COURSEWORK: SCHOOL EVENT PLANNER AND MANAGER

Samuel Barrett

Candidate Number: 2317

Royal Grammar School High Wycombe

H446

1 CONTENTS

1 ANALYSIS

1.1 PROBLEM DEFINITION

Within a school environment it is very easy for messages regarding a multitude of events to be lost between the organizer and the attendees. This could be down to human or software error. A form tutor could forget to pass on a message or check their emails. An email could fail to send or be blocked by an over sensitive school firewall. I intend to address this issue with a piece of software specifically designed to deliver messages regarding school events to specific members of staff or the pupils themselves.

1.2 INTENDED MARKET/AUDIENCE

The intended user-base for my program would be a school as a whole. The program would be available for use by both students and staff alike with varying amounts of functionality depending on their station. I feel that schools such as my own would have a vested interest in a software solution such as this as it would allow for a smoother running school where perhaps notices are not needed to be passed via assembly and form tutors allowing for this time to be focused on other aspects of the student's education.

1.3 STAKEHOLDERS

The success of my program will be assessed by the final customer. In my case this will be my school. If my final program does not meet their requirements it will not be utilised within the school.

The final customer's name is The Royal Grammar School in High Wycombe; they are a state funded grammar school that employs around 200 staff to teach and manage the school for the 1300 students. They are always expanding, building new facilities and gradually increasing their student capacity and thus, their staff.

My contact at this business will be the IT systems coordinator, Paul Berkman. Who works 5 days a week and will be accessible via email throughout the development process.

Use of my program will aid the growth of the school as it will allow for much more efficient management of events within the school. with a school this size there are multiple events taking place every day involving many students and members of staff. This has become too much to be managed any way other than computationally

To research their needs, I will perform interviews looking to see what different demographics within the school want out of such a program. These demographics will include:

- Students

    - From these interviews, I will find out what the student based functions of the program need to encompass
- Teachers
    - The teachers will be the main users of this program so their feedback will perhaps be the most key. I hope to find out the current issues with the used system so that I can address this with my final solution
- Managerial staff
    - This is an interesting perspective as they want the system to run as smoothly as possible so their ideas for the program may be more focused on the backend or confidentiality of data

I will also perform research into the methods currently used within the school and other similar organizations in the area using this research to form my initial ideas addressing some of the areas that require improvement or gaps in the market.

1.4 WHY A COMPUTATIONAL SOLUTION IS JUSTFIED

To effectively manage events within a school environment you must be able to enter data and have it outputted in the following formats:

1. As detention forms
2. As personalized user agendas
3. As analytics data based on the data collected during the running of the school

1. This can be achieved very easily using a computational approach as it requires a form to be filled out with user inputted data. These inputs will be known as they will be entered into various parts of the GUI form during (detention) event creation.
2. This can be solved computationally as it requires data to be gathered from storage based on various factors such as the current user and date/time frame required. This can be solved computationally as it requires iterative data searching to find certain records. Doing this via a computer program is much more time efficient than personally going through each record to search for something
3. This data is useful to the school's managerial staff as it allows them to monitor how events are running within the school in different scopes (specific student, forms, years etc.)
   It will be easy to create as it will need to perform calculations on data collected during the use of the program and stored in the database. The database must be fully normalized so the data stored will be consistent throughout allowing the program to perform statistical analysis on data with relative ease

   My solution however will only be a prototype version as the python language I intend to use will not be able to run on the student devices as they run iOS. I am prototyping in

python before developing in Xcode or C++ as it is a very easy language to porotype in due to its pseudocode like syntax. Hopefully this prototype version will also highlight areas that need further work and areas that I may need to enlist help to get working properly.

My program is amenable to a computational approach as it requires the efficient storage and random access of a large amount of data, with data relevance constantly changing. A system of this scale would require several people to operate if it were not computerized. Some of the analytical functions would also require a lot of time to execute and would be prone to human error. To perform even simple calculations on such a vast amount of data without a computer system (i.e. pen and paper) would take far too long

1.5 PROJECT IDEA

- To create a better management system for school events including internal events and detentions and external events
- This is a necessary concept as events are currently poorly tracked with students often missing meetings or events due to form tutors failing to inform them of the details.
- This program will provide the infrastructure necessary for students and teachers alike to check any events their presence may be required at.
- Provide advanced analytics data for students and Teachers regarding amount of detentions a student has and how much time out of lessons they have for various events
- Program could post notifications if certain thresholds are exceeded
- Could plot graphs using statistical data and algorithms
- Program should notify form tutors of students in their form's events
    - When they are given a detention
    - When they may be absent for an external event
    - To remind the student of an internal event on the day

1.6 INTERVIEWS

1.6.1 QUESTIONS

1. Is there any software currently in use at your school that attempts to perform a similar task to the one I have outlined?
    a. If yes: do you feel it performs well?
        i. If not what improvements do you feel are necessary
    b. If no: do you feel that a piece of software like this is necessary or would be useful?
        i. What features do you think would be needed for it to catch on?
2. Do you often find that you miss events or find out about them late?

   a. Do you feel that my proposed program could alleviate that issue?
3. Do you ever find that rooms are double booked in your school?
   a. Would you use my program to resolve this issue if it were available to do so?

I feel these questions will give me insight into the kinds of things the end user would like out of my program. These things will then be incorporated into my objectives.

1.6.2 RESPONSE 1

1) "Yes I believe the SIMS software package performs these functions"
   a. "I believe that SIMS is a useful tool but unreliable and confusing as many teachers find it difficult to operate"
      i. "I wish it could send emails to all participants of events to improve attendance. It should also have a way of informing parents if their students are not attending compulsory events as this is currently a manual process"
2) "Yes I am often informed about events by my form tutor late or even on the day of the event"
   a. "Yes I believe it would"

3) "Yes all the time. For example, in RAF CCF in year 10 we often went to rooms where lessons were being held"
   a. "Yes I would as this issue wastes valuable time that could be better spent if rooms were efficiently organized"

- Mansoor Wooding RGS Student

1.6.3 RESPONSE 2

1) "Our school currently uses a software package called SIMS it provides a lot of the software based infrastructure for the school's day to day running "
   a. "I feel that a lot of its functions work to an extent but some require improvement or replacement. The user interface is also difficult to navigate requiring many hours of teacher training before a lot of its basic functions can be used effectively by the staff. Some of its more complex features are not used at all by the school as too few people properly understand the package"

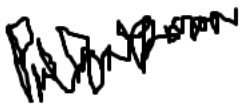
        i. "I wish it had a cleaner user interface as was able to connect and be used by the students to an extent as well as the staff. "
2) "I find that as a form tutor I often find out about things late myself and this then creates a concertina effect making the students find out about things even later"
    a. "I think a software package such as this with a student focused front end as well as a teacher and admin interface would solve this particular problem well if adopted by my school"
3) "As a teacher I find that if I have to move my class to a room for certain events they are often double booked and due to the poor use of the clunky SIMS program I am not notified when I book the event"
    a. "This sort of issue wastes class time and makes events overrun"

- Daniel Thomas, Head of RGS computing

1.6.4 RESPONSE 3

1) "The school currently the Capita software package SIMS, we maintain a database and run its admin tools on our school servers."
    a. "I feel that it has the potential to perform well and has many useful features but is being held back by poor development and use of user feedback."
    b. "I feel that due to the poor progress with the improvement of SIMS small software packages such as the one you propose could be key in fixing the issues and perhaps alerting Capita to the issues"
        i. "The sorts of features I would expect out of a software package such as this would be similar to those provided by SIMS coupled with advanced analytics of the data regarding student event attendance etc."
2) "As I am not a teacher or student I do not suffer from these issues although I am aware that they happen frequently due to poor planning via SIMS "
    a. "if used properly I feel that your proposed program could solve these issues"
3) "Again I do not suffer from these issues but I am aware of them"
    a. "Yes I feel that your program would be used to solve these issues if it were available to the staff and students"

- Paul Berkman, RGS IT systems coordinator.



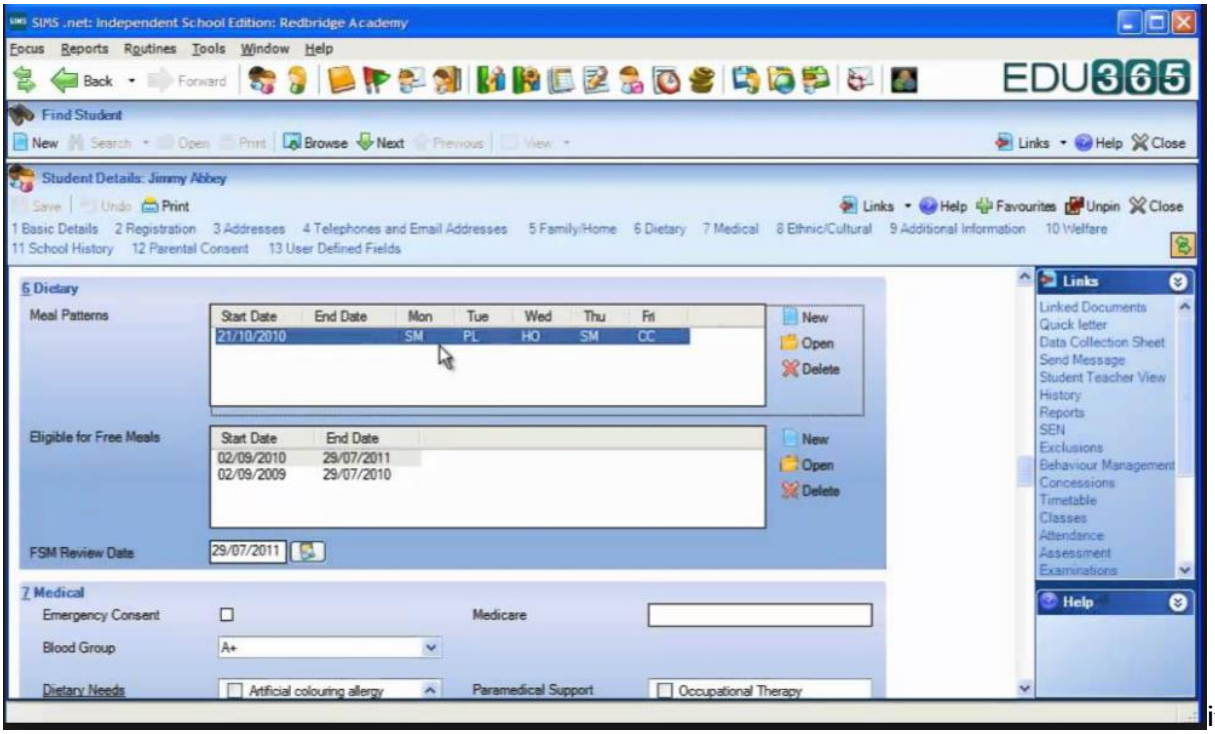
1.7 RESEARCHING THE PROBLEM

1.7.1 SIMILAR PRODUCTS

1.7.1.1 SIMS

SIMS is the software currently in use in the vast majority of schools in the UK holing over 80% of the market share across primary and secondary schools; RGSHW is among these.

It provides an antiquated GUI designed during the windows XP era



it provides a teacher and management focused GUI with no student facing side to the program. Due to poor GUI design, many of its features are underutilized. I aim to learn from this, making my GUI intuitive and well designed.

SIMS has many good features such as a networking capability allowing it to run on all staff computers and a robust backend leading to very few crashes and errors

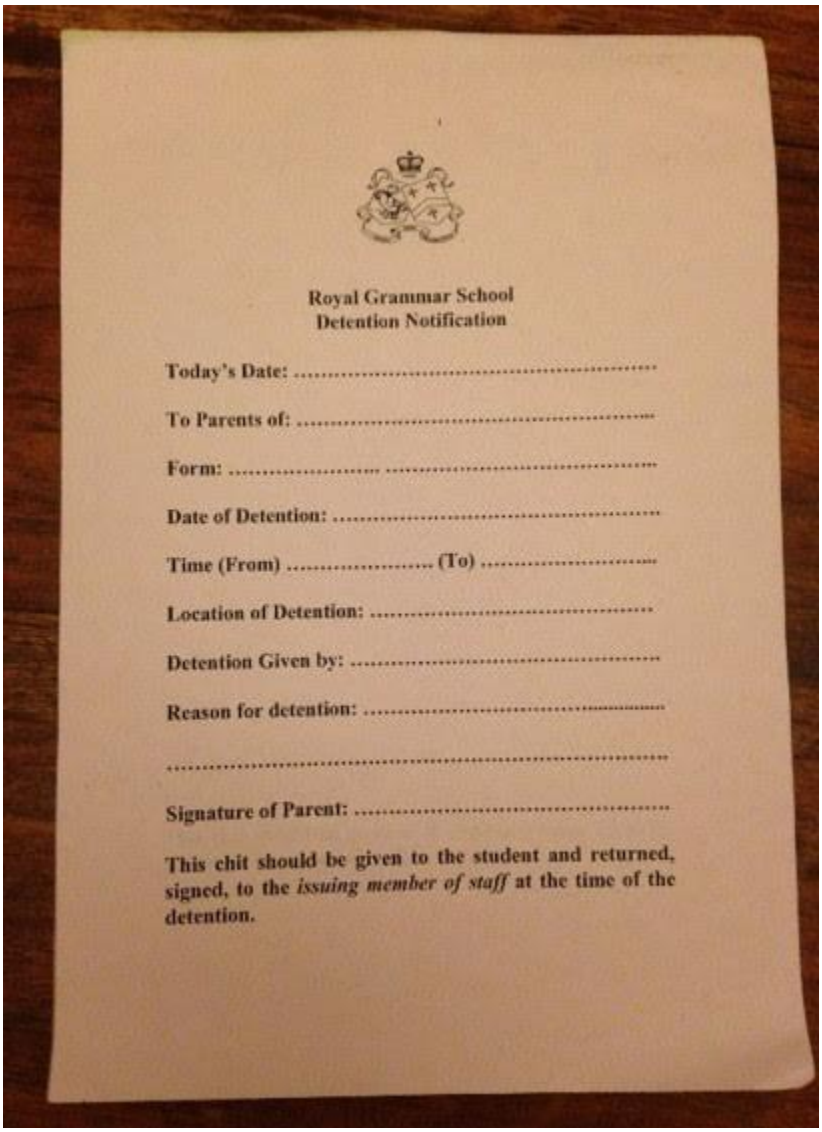
SIMS utilizes a client-server setup with the server side being run on Microsoft SQL Server and a .NET framework module

1.7.1.2 CURRENT SCHOOL SYSTEM

The current system in use at my school seems fairly archaic with all events being displayed on a spreadsheet, edited by one member of staff. This spreadsheet is read only and only accessible

by staff. It doesn't display all events going on in the school instead displaying large events that affect many students or staff. This sort of system is prone to mistakes and leads to many school events being overlooked leading to most day-to-day events being managed via emails. This system, I feel, is not an efficient use of time and doesn't produce the best results.

Currently at my school detention forms are filled out manually by teachers for after school and Saturday detentions only. They require special forms to be bought as it needs to create 3 copies. My program will allow the teachers to print off forms for all detention levels and save time by not having to write them out themselves. It will also produce much more legible results



Royal Grammar School
Detention Notification

Today's Date: .....

To Parents of: .....

Form: .....

Date of Detention: .....

Time (From) ..... (To) .....

Location of Detention: .....

Detention Given by: .....

Reason for detention: .....

.....

Signature of Parent: .....

This chit should be given to the student and returned, signed, to the *issuing member of staff* at the time of the detention.

There doesn't seem to be many programs that address this issue within most schools

1.8 ESSENTIAL FEATURES

1.8.1 CORE FUNCTIONALITY

1. Allow the user to log in and out of different user accounts on the same machine
2. To create an intuitive user interface with which any user can fully utilize the program
3. The program must be able to have multiple access levels
    a. This must be used to prevent users from accessing areas of the program
4. Must be able to recognize when a room is double booked and prompt the second user to change venue
5. Must be able to take input of data
6. Must be able to enter inputted data into the DB
7. Database must be fully normalized
8. Must provide statistical analysis of the data recorded based on group
    a. Specific student
    b. Specific teacher
    c. A form
    d. A year
9. Must be able to provide analysis based on event type
    a. Internal events
    b. External events
    c. Detentions
10. Analysis should include
    a. Time missed due to events
    b. Attendance of events
11. Must be able to notify a form tutor if an attendance rate of a pupil falls below a certain level
12. Must be able to display events based on a given time frame when showing analytics data
13. Must be able to display events based on a given time frame when showing personal events of a user
14. Detentions should take priority over other event types if events overlap
15. Has a method of being broadcast to screens around the school

1.8.2 STUDENT INTERACTION

16. Must be able to print out list of events specific to student based on their form or year etc.

17. Program must be able to recognize when a student has clashing commitments and not allow two events to overlap
18. Program can be accessed from mobile devices such as iPads

1.8.3 STAFF INTERACTON

19. Program should not allow events to be booked for strange times such as late at night or very early in the morning
20. Must be able to create new users with secure login details (hashing) this feature should be restricted to administrative users
21. The program must be able to add new detentions
22. The program must be able to add new internal events
23. The program must be able to add new external events
24. Must be able to print out list of events specific to teacher
25. Must be able to register attendance of users
26. Program must be able to recognize when a teacher has clashing commitments and not allow two events to overlap
27. Must prompt user to delete event if events overlap

1.9 HARDWARE & SOFTWARE REQUIREMENTS

My program would have relatively low requirements. My intended language to create this program is Python which has very low memory and CPU usage since it also runs on mobile devices. Memory usage will depend on the finished product however. I estimate that a minimum of one GB of RAM will be needed. And around 5GB of storage depending on the size of the database.

The software requirements of my program will be a python interpreter and any of the external libraries utilised in the writing of the code, such as PyQt4 or SQLite3. And a OS that supports

1.10 POSSIBLE LIMITATIONS

Due to various hardware and software factors there will be some limitations to my final solution

One major source of limitations comes from the use of SQLite instead of more feature-full versions such as MySQL. This reduces the amount of networking potential and scalability the final program will have.

It would be possible to move to a MySQL database later but it would require significant restructuring of the code related to database queries. Currently by using SQLite this limits us to a local database file which will not be sustainable for a system in use by many people at once. Cementing the fact that this will be a prototype version.

Due to the limitation of python's interpreter some functions may be performed slowly such as the higher intensity calculations that may be performed for the analytics of student attendance. As it does not utilise GPU cores or multiple threads on CPUs

Points 15 and 18, above, will not be able to be implemented in the first version of the program as it requires skills (iOS coding and networking) that I do not possess. The porting of the program to iOS will also be too much work for just me in my allotted time frame. These features could be implemented later, once the program is in use around the school.

However this feature would be worth adding later on as it would allow all students to access the program on their personal devices (iPad etc.) allowing the schools existing computer infrastructure to remain unchanged.

1.11 FOLLOW UP INTERVIEW

1.11.1 QUESTIONS

1) Do you think these objectives satisfy your needs?
2) What if anything would you want it to do beyond these outlined objectives
3) Is a clean user interface paramount to you in regards to the final product or does functionality count for more?

1.11.2 RESPONSE

1) "Yes, I feel that they do. The outlined objectives seem to encompass all the discussed elements."
2) "I would not add anything as I feel that my input from the previous interview summed up my ideas towards the requirements "
3) "I feel that due to the broad range of possible end users including students and teachers who may not be terribly good with computers a cleaner User interface is needed over a clunky feature packed program "

- Mansoor Wooding, RGS Student

1.11.3 RESPONSE 2

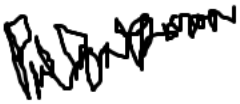
1) "I feel that my feedback on SIMS has been noted as these objectives seem to remedy all the major issues with SIMS regarding its event planning system"
2) "I would make sure that the program is properly optimized for the school teacher and student hardware making sure load times are kept to a minimum"
3) "As stated in my original interview one of the main issues with the SIMS program is that many teachers cannot fully utilise it due to the bad UI therefore I feel that a clean user interface is paramount to our school adopting your software"

- Daniel Thomas, Head of RGS computing

1.11.4 RESPONSE 3

1) "I feel that the issues I spoke of to do with the SIMS system have been considered and fixed with these proposed objectives"
2) "I would make sure that the database integration with this program is easily integrated with the existing school systems"
3) "As I said SIMS is currently underutilized in no small part to its poor usability therefore a well-designed user interface is necessary for this program to be used within our school by students and staff as many are not technologically minded"

- Paul Berkman, RGS IT systems coordinator.



1.12 SUCCESS CRITERIA

1.12.1 PROGRAM INITIALISATION

1) Loads all PyQt GUI files from a designated folder upon starting the program and throws a contextual error if it cannot
2) Connects to a SQLite file stored along-side the program and throws a contextual error if it cannot

3) Finds and imports all required python libraries
   a) Such as PyQt4 and SQLite3
4) All data inputted by the user is validated by the program to prevent errors
5) Use of Object oriented code for constructing all GUI forms as objects
6) Allow the user to move between different GUI forms without lag

1.12.2 LOGIN SCREEN

7) Allows user to enter username and password via lineEdits
8) Password line edit does not display characters to keep password secure
9) Passwords must be stored in a hashed format
10) Entered passwords must be hashed in the same format as stored passwords to allow them to be compared
11) Be able to log in both student and teacher users from the same screen
12) Must be able to create an object to store key user data upon successful login
13) Must be able to notify the user if their username is incorrect
14) Must be able to notify the user if their password is incorrect
15) Alert the user when details are entered correctly
16) Must be able to search the database for teachers based on username
17) Must be able to search the database for students based on their username
18) Allow the user to close the program upon a button press

1.12.3 SPLASH SCREEN

19) Display all required GUI objects
20) Must not allow student users to access the create new user screen
21) Must not allow student users to access the create new event screen
22) Must not allow student users to access the advanced analytics screen
23) Must not allow student users to access the registration screen
24) Must allow the users to logout of the program
25) Must not leave data remnants after logout
26) Must allow the user to exit the program
27) Must allow teacher users to access the create new user screen
28) Must allow teacher users to access the create new event screen
29) Must allow teacher users to access the advanced analytics screen
30) Must allow teacher users to access the registrations screen
31) Must allow all users to access the personal events screen
32) Must allow all users to access the modify users screen

1.12.4 CREATE NEW USER SCREEN

33) Must run presence checks on all fields upon user creation
34) Must alert the user to any empty fields that need to be filled out
35) Must allow for the creation of student users
36) Must allow for the creation of teacher users
37) Must allow for the entry of all relevant data for each user type
38) Must visually show the user which fields need to be filled out for each user type
39) Must fill out all information in the database
40) Must use corresponding foreign keys where necessary to keep database fully normalized
    a) Eg. Find associated FormID from a user entered form name
41) Must return errors if any data cannot be found in the database, prompting the user to correct the mistake

1.12.5 CREATE NEW EVENT SCREEN

42) Must run presence checks on all fields upon event creation
43) Must alert the user to any empty fields that need to be filled out
44) Must allow for the creation of detention events
45) Must allow for the creation of internal events
46) Must allow for the creation of external events
47) Must use corresponding foreign keys where necessary to keep database fully normalized
    a) Eg. Find associated RoomID from a user entered Room name
48) Must return errors if any data cannot be found in the database, prompting the user to correct the mistake
49) Allows the user to return to the splash screen
50) Must check the time to be suitable
51) Must check to see if the room is double booked
52) Must check to see if the attendees have a prior engagement

1.12.6 SEE PERSONAL EVENTS SCREEN

53) Allows the user to specify the start date of events to be shown
54) Allows the user to specify the time range
    a) A month or a week
55) Allows the user to seek through events by time in increments of 1 month or 1 week
56) Allows the user to filter events to be shown by type
57) Allows the user to return to the splash screen
58) Shows all relevant event data in their most user-friendly form
    a) i.e. all foreign keys stored in the events table are translated to their full forms stored in their separate table
59) program only displays events relevant to the currently logged in user

1.12.7 ADVANCED ANALYTICS SCREEN

60) Must allow the user to specify the group type they would like to analyse
    a) Form
    b) Year
    c) Student
61) Must allow the user to enter the group's name
62) Must search the database for the group based on the entered name and type
63) Must be able to find data related to the group
64) Must be able to analyse this data for attendance and number of events
65) Must be able to display fetched and calculated data in a table
66) Must be able to display calculated values in a pie chart format

1.12.8 MODIFY USER SCREEN

67) Must allow student users to change their own passwords
68) Must allow teacher users to search for specific users via a lineEdit
69) Must display all data related to users, fetched from the database
70) All foreign keys must be translated to more user friendly, corresponding values
71) Must allow teacher user to edit data of any student

1.12.9 REGISTRATION SCREEN

72) Show all students who are supposed to attending a specific event
73) Must display only one event at a time based on current time and date
74) Must allow teachers to register events they are supposed to be attending
75) Must update the database with the completed register upon a button press

2 DESIGN

2.1 PROBLEM DECOMPOSITION

For my program to fulfill its overall purpose there are many contributing smaller requirements it must satisfy

2.1.1 USER OBJECT

- This construct must have the ability to be used to create an instance from within the login screen
- It must have attributes that can be set to important user information for use throughout the program

2.1.2 LOGIN SCREEN/OBJECT

- My login screen involves checking a user inputted username and password against stored values in the database.
    - It must de-hash the stored passwords using the salt and compares it with the inputted password only allowing the user to continue to the 'splash screen' if both username and password is correct
    - As usernames must be unique to save computational time I first search the database for the username only then if it exists do I compare the passwords.
    - This allows me to have more helpful error messages telling the user if the issue is with their username or password rather than like in some instances where it will output the generic "username/password incorrect"
    - This program also should create an instance of the next program if the requisites are met.

2.1.3 SPLASH SCREEN/OBJECT

- This screen has little functionality beyond allowing the user to navigate to other parts of the program via push-buttons
    - It will do this by having multiple methods that allow it to create instances of various other objects, or programs, hiding or closing itself as it does this. These programs will have the ability to send the user back to this screen if they desire to access a different part of the program

2.1.4 CREATE NEW USER SCREEN/OBJECT

- This program will allow an administrative user to create new users of the same 'status' or lower
    - It will do this by having many input fields for the necessary data to create a row in the required database tables.
    - It will then perform queries on the database to enter this data where necessary before saving its changes
    - It must be able to reject data and tell the user the issues with it such as a blank field

2.1.5 CREATE NEW EVENT SCREEN/OBJECT

- This program must be able to perform the same tasks as the 'Create New User Screen' as it performs a similar function but for different data
    - It must be able to accept the relevant data and ONLY the relevant data (no data redundancy)
    - If creating a detention, it must prompt the user to print out a detention form which has been pre-filled in with the required data
    - It must also add these events to the 'calendar' of each registered attendee
    - It must be able to enter this data into the required columns in database tables

2.1.6 SEE EVENTS SCREEN/OBJECT

- This is perhaps the most integral program within this project as it allows the user (admin or student) to see a sequential calendar view of all the events they are expected to attend from detentions to sports fixtures
    - This must show events for a specific user (the one currently logged in)
    - Must be able to show events in varying time frames in the future or over a longer time span (in the next day/week/month)

2.1.7 ADVANCED ANALYTICS SCREEN/OBJECT

- This program must only be accessible by administrative users
    - It should have the ability to show detailed data on student attendance to events
    - User must be able to specify the size of the population for use on the statistical analysis eg a specific form/ year etc.
    - Must display graphs and charts of the data found in the database including student attendance of event and time spent in events instead of in lessons (ie time of event lies between 9.15->11.45 ,12.15->1.30, 2.20->3.40)

2.1.8 MODIFY USER SCREEN/OBJECT

- This program simply allows admin users to call up data on other users and edit it such as changing passwords or making admins
    - Must only allow admins to make changes
    - Must make sure that integrity of data is conserved ie changes must still adhere to the rules that were in place when originally created

2.1.9 REGISTRATION SCREEN/OBJECT

- This program must only be accessible by administrative (teacher) users
    - It should allow the teacher to see all students enrolled in the event currently taking place

- It should find the current event by using the system clock and then query the database for events in that time range
- It must allow the teacher to enter present or missing into the register for each enrolled student
- It must commit back to the database the attendance data once submitted by the teacher

2.2 STRUCTURE OF SOLUTION

The way in which I have laid out my decomposition shows the basic flow of the program

The first screen will be the login screen and its related code; this will then send the user to the splash screen. From here they can navigate to all the other screens within the program. Before I can create these screens however the backend must be completed. This includes:

- Database setup
  - Table creation
  - Foreign key setup
  - Data population
- User object creation
- Miscellaneous function creation
- User interface design and file import

The 'See Events Screen' works in tandem with the 'Create New Event Screen' as it allows all the users to view the output of that previous screen. Without one the other would be useless

By the same token the 'Modify User Screen' and 'Create New User Screen' are linked as the latter allows admins to create users whilst the second allows the admins to alter their previous work.

The advanced analytics screen utilised all the data inputted by the other screens to show graphs and charts

All these 'screens' will have code to add the functionality to the GUI form. Without it the program would not work as buttons would not trigger functions.

2.3 INPUTS, PROCESSES, OUTPUTS AND STORAGE

Input Type	Entered Via	Processing

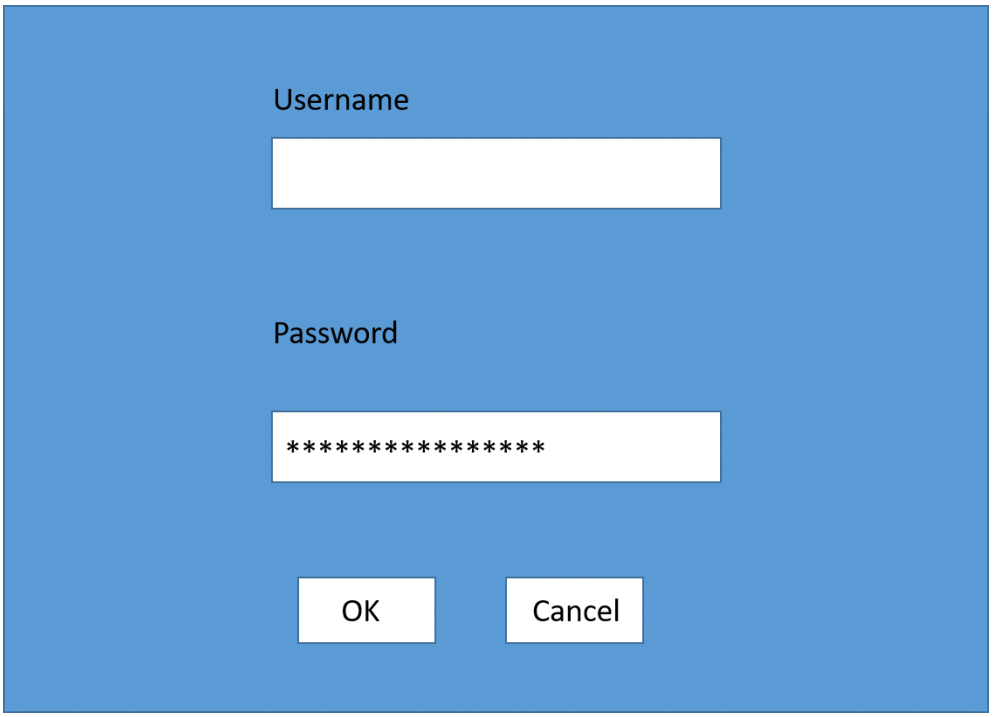
String Value	Text Field	Used to search database for validation or for finding related data such as a primary/foreign key Also, stored in database in some instances such as during user creation
Integer Value	Text Field or Double-Box	Used again for searching DB or storage in DB but also for mathematical operations etc.
Option	Radio Button	Usually translated to string values for use but entered as an option out of mutually exclusive options. Used to aid usability
Date	Date edit box	Translated to string but entered via date box as validation
Time	Time edit box	Translated to string but entered via time box as validation

Output Type	Outputted Via	Description
Error Message Type1	Text Label	This is used to alert user to minor errors such as an input not being long enough etc
Error Message Type 2	Popup Window	Used to alert user to major errors such as the function they just tried to perform didn't work. These require immediate attention and make sure they acknowledge the issue
Detention Form	Popup Window then printed into physical copy	When detentions issued from within the program a

		detention form is filled out digitally the user is then prompted to print out the completed form to give to the offending student

2.4 USABILITY FEATURES

2.4.1 LOGIN SCREEN MOCKUP

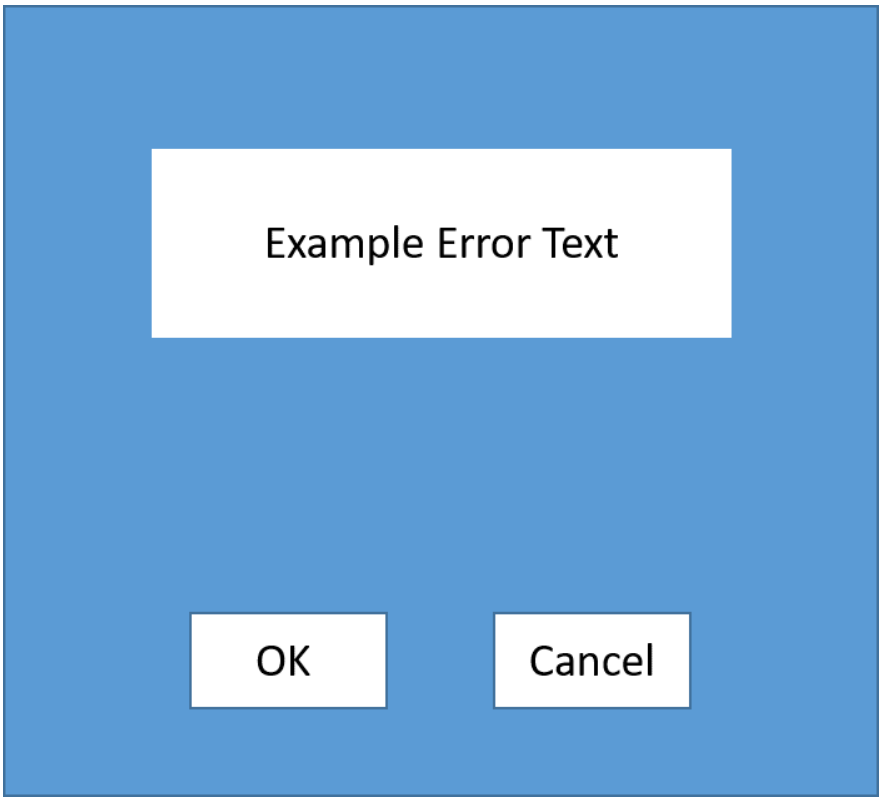


This is the design for my login screen. It is a very basic design as it requires very little user interaction beyond typing in their credentials and hitting the 'ok' button

The password entry field will not show the characters typed into it for data security reasons it will show black dots in their stead

This design in its simplicity is very simple and familiar to most users.

2.4.2 ERROR SCREEN MOCKUP

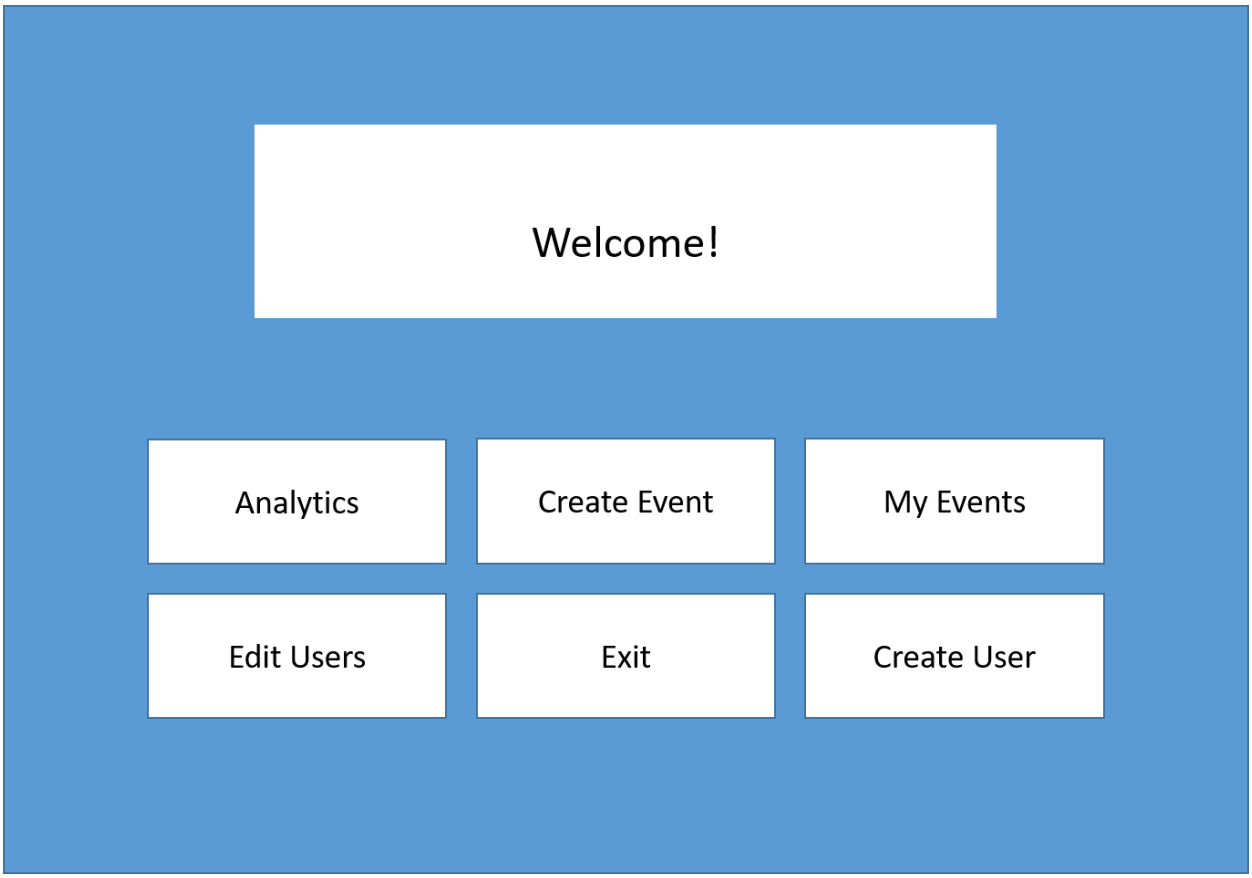


This screen is one that will crop up throughout the program and alerts the user to various critical errors or mistakes they have made

This screen takes inspiration from many other programs and is a standard design for an error popup, very similar to those used in Windows and OS X.

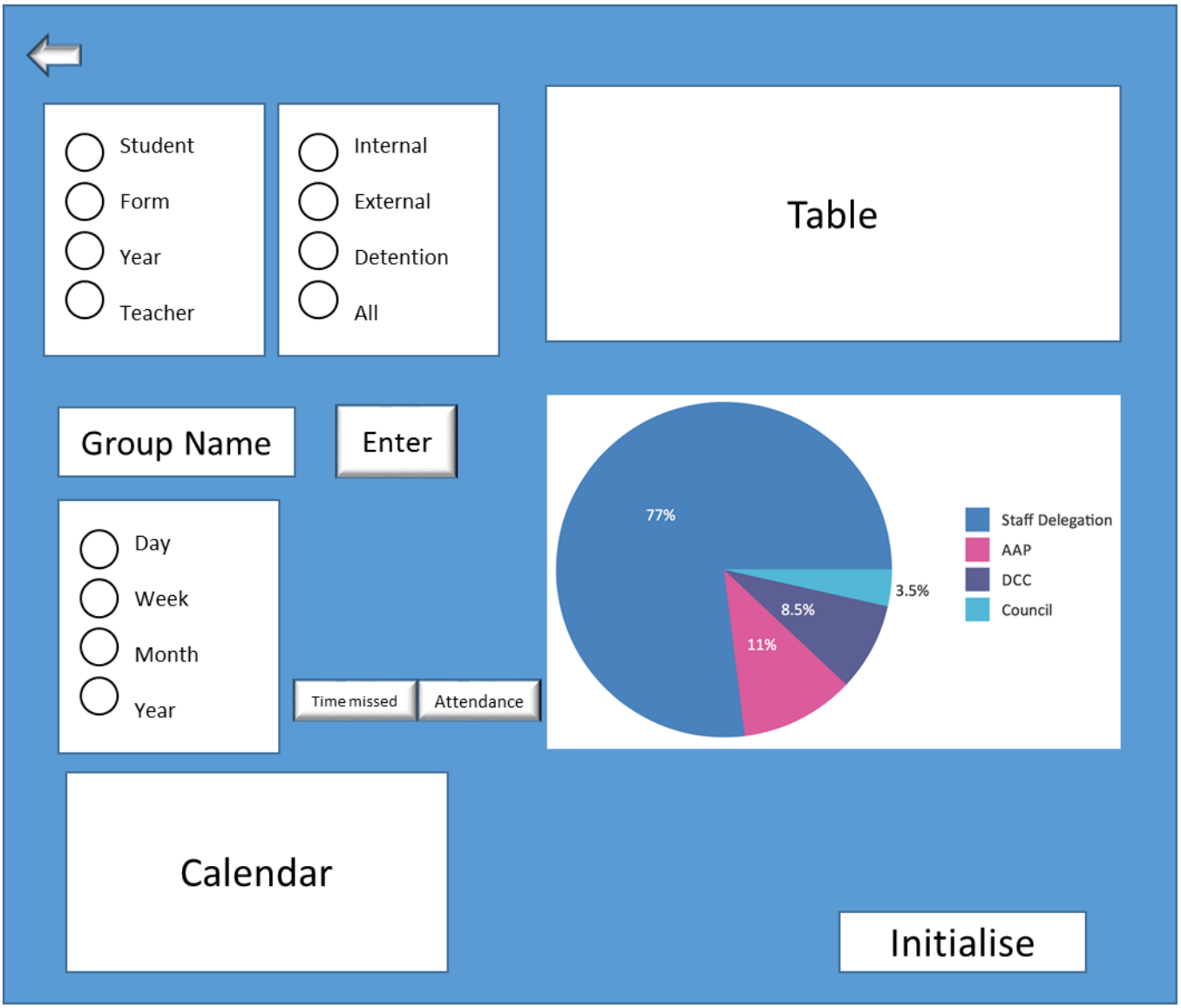
I have not cluttered the design as its purpose it to get a message to the user and unnecessary items on the screen only detract from that

2.4.3 SPLASH/ START SCREEN MOCKUP

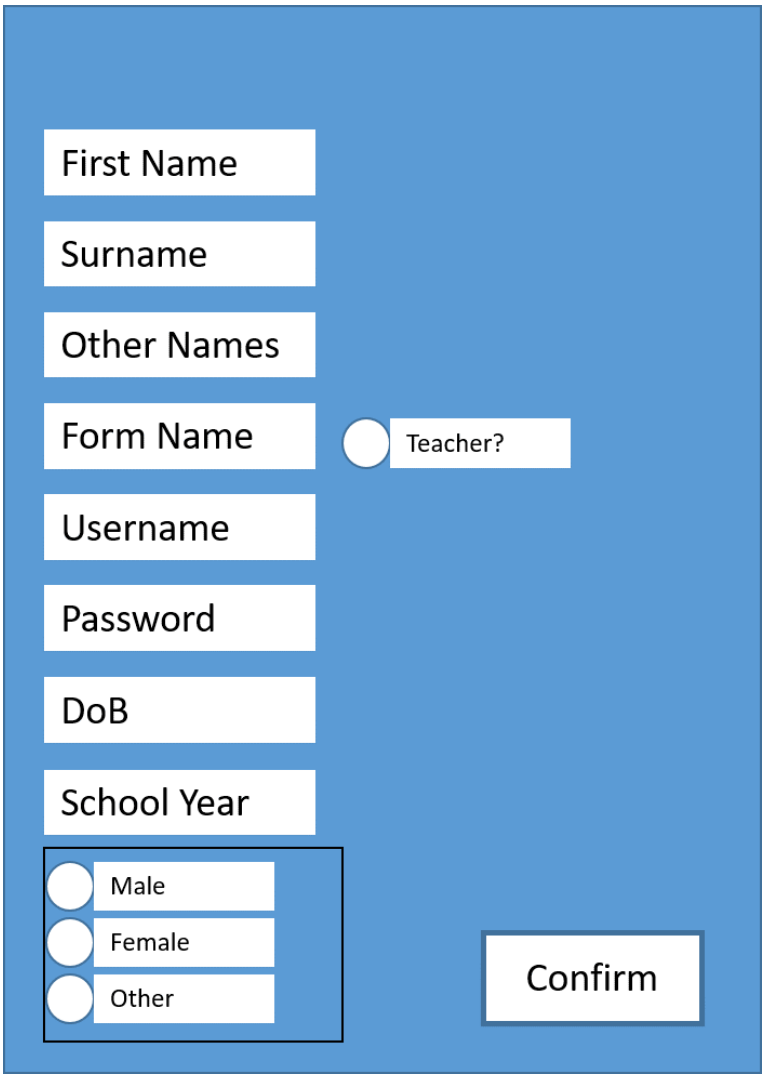


This is the hub screen of the program and I have designed it to be very spacious to make sure the text is big enough to be read by any user and so that the chance of navigating to the wrong screen is minimized

2.4.4 ADVANCED ANALYTICS SCREEN MOCKUP



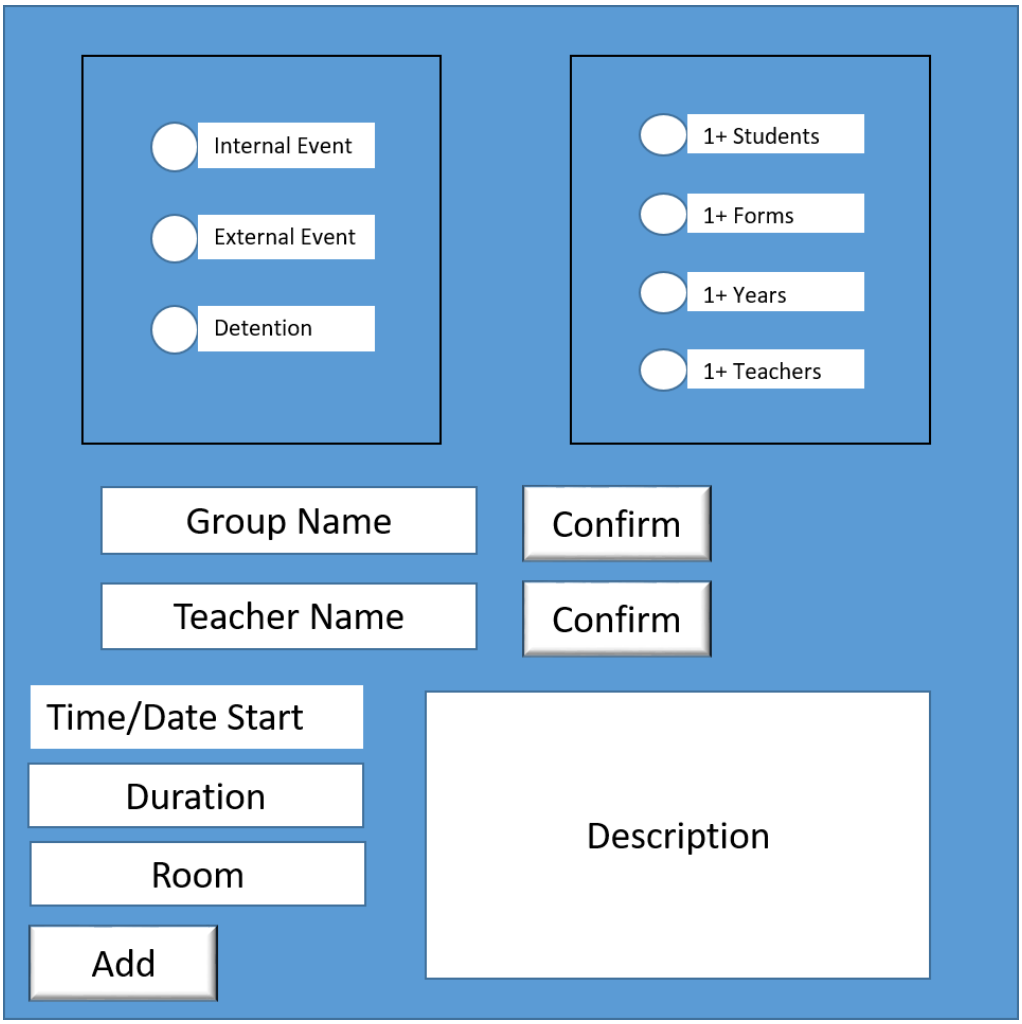
2.4.5 CREATE NEW USER MOCKUP



This screen allows an administrative user to add new users to the database system.

It has input fields to allow data entry to satisfy all DB fields. It will take in the password unhashed but hash it before it stores it in the database. I will utilise radio buttons for set value inputs and a date/time entry widget for the DoB.

2.4.6 CREATE EVENT SCREEN MOCKUP



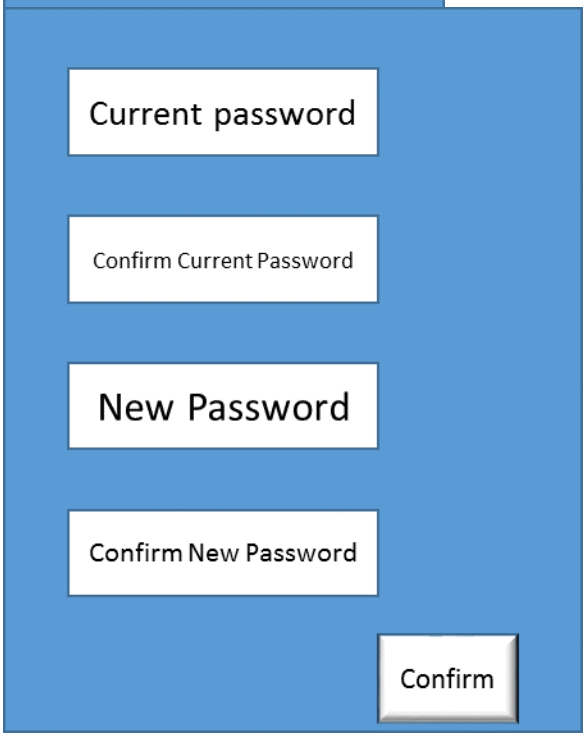
This screen allows administrative users such as teachers to add and enroll student and teachers in events of the 3 types. It allows the user to add users in different groups or individually. All the data entered via this screen is stored in different database fields and tables all related to each other.

I will utilise 2 groups of radio buttons for group and event type selection as well as a date time entry widget for the start date and time of the event.

2.4.7 MODIFY USERS MOCKUPS

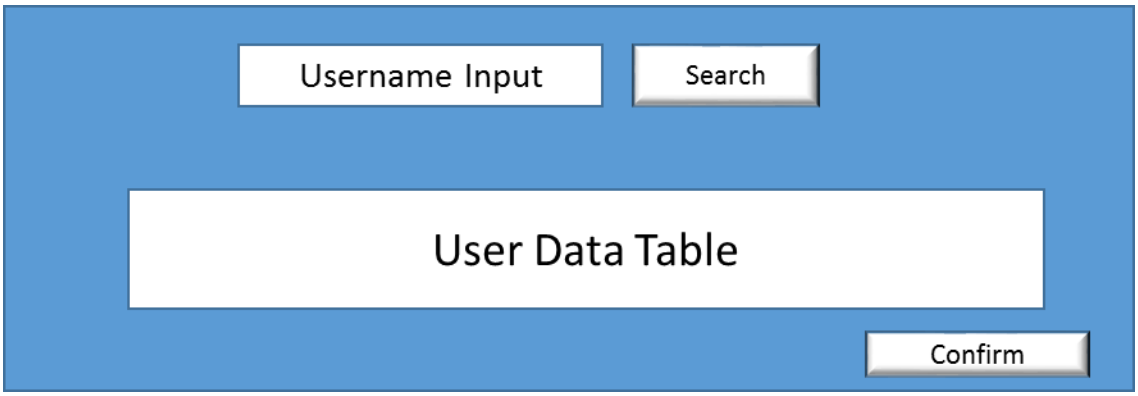
This screen allows users to direct to either of the following two screens. It should prevent student users from accessing the top option. It will use buttons with clear Calibri font. The text should take up as much of the button as possible

2.4.7.1 CHANGE PASSWORD



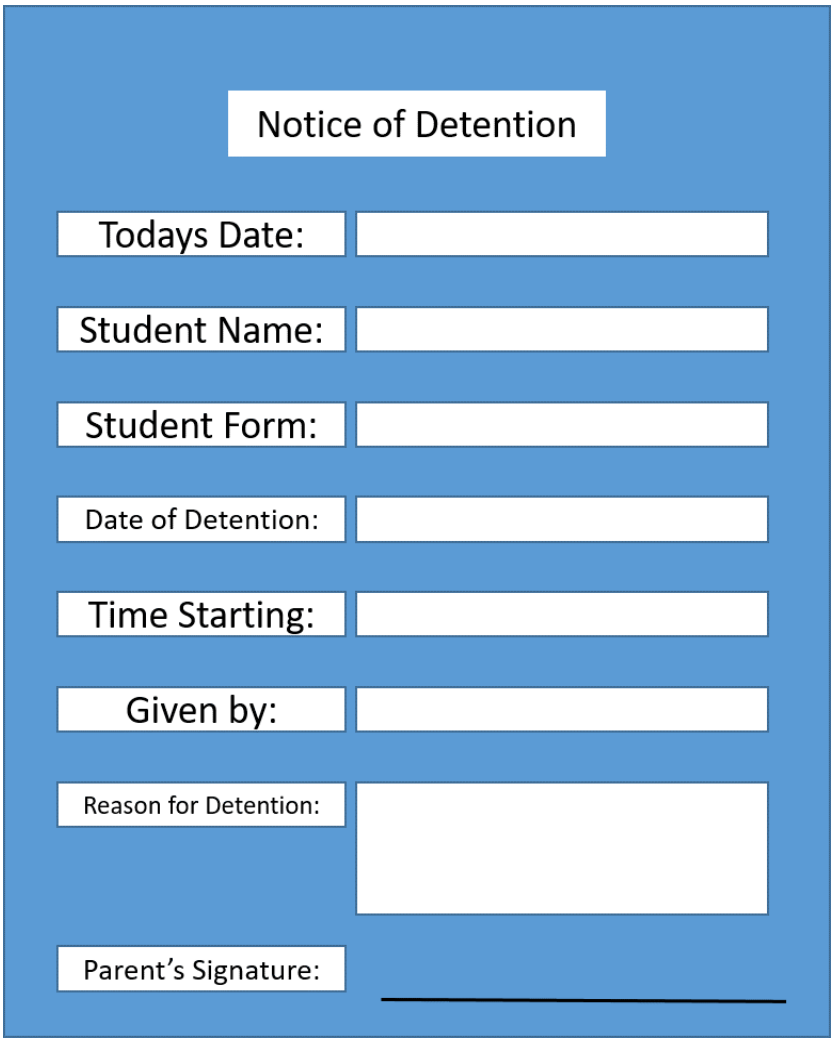
This screen uses 4 input boxes and a button to confirm input. The screen prompts the user for the current password with a confirmation entry. And then their new password with a confirmation entry, the passwords will be checked to be matching and follow the regex rules for the system.

2.4.7.2 EDIT USER DATA (ADMIN)



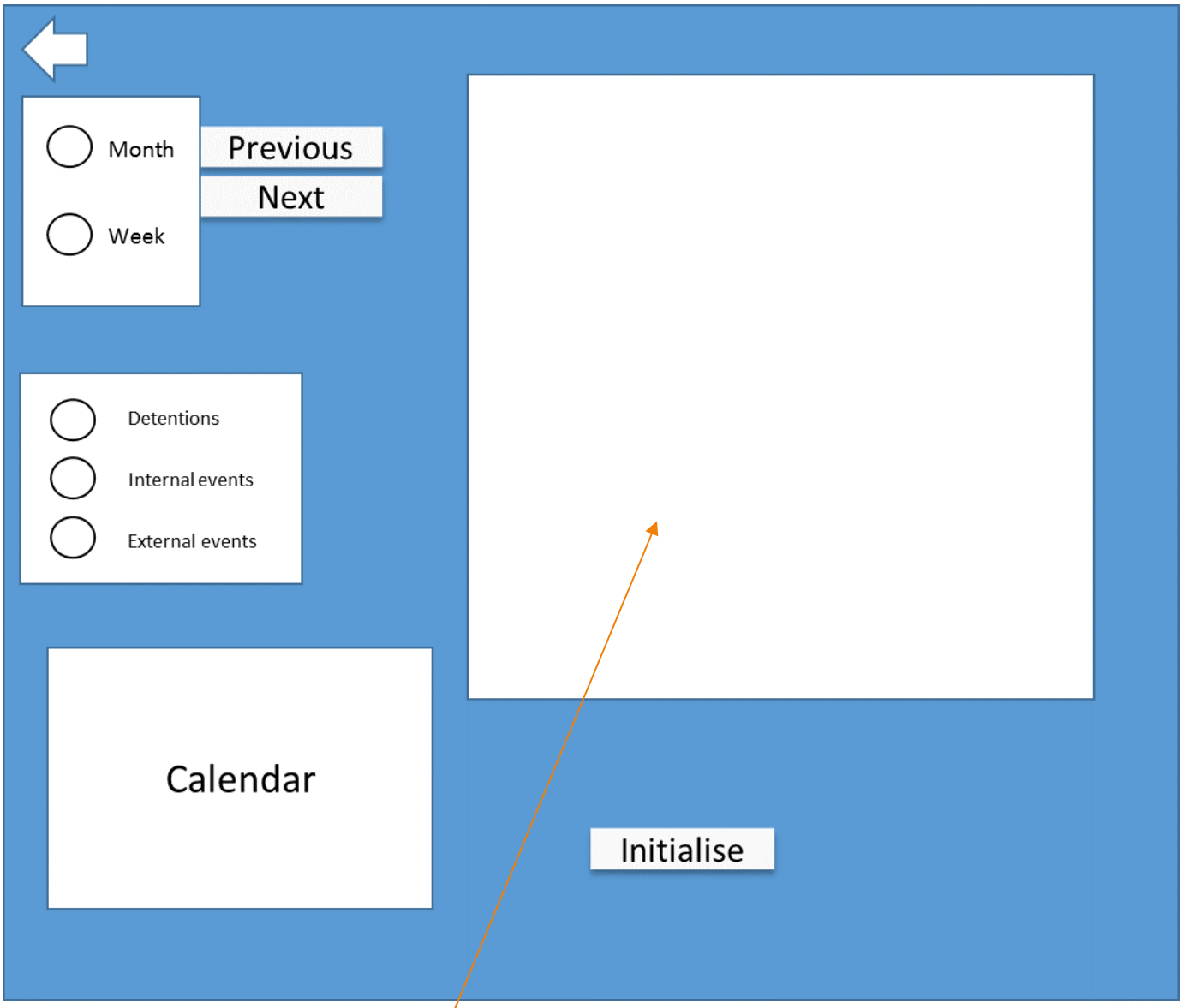
This screen allows administrative users to search for any user using their username. All data relating to that user will then be displayed in a table below. The passwords will not be shown in plain text. The admin can then change any of the detail and they will be updated back into the database.

2.4.8 DETENTION SCREEN MOCKUP



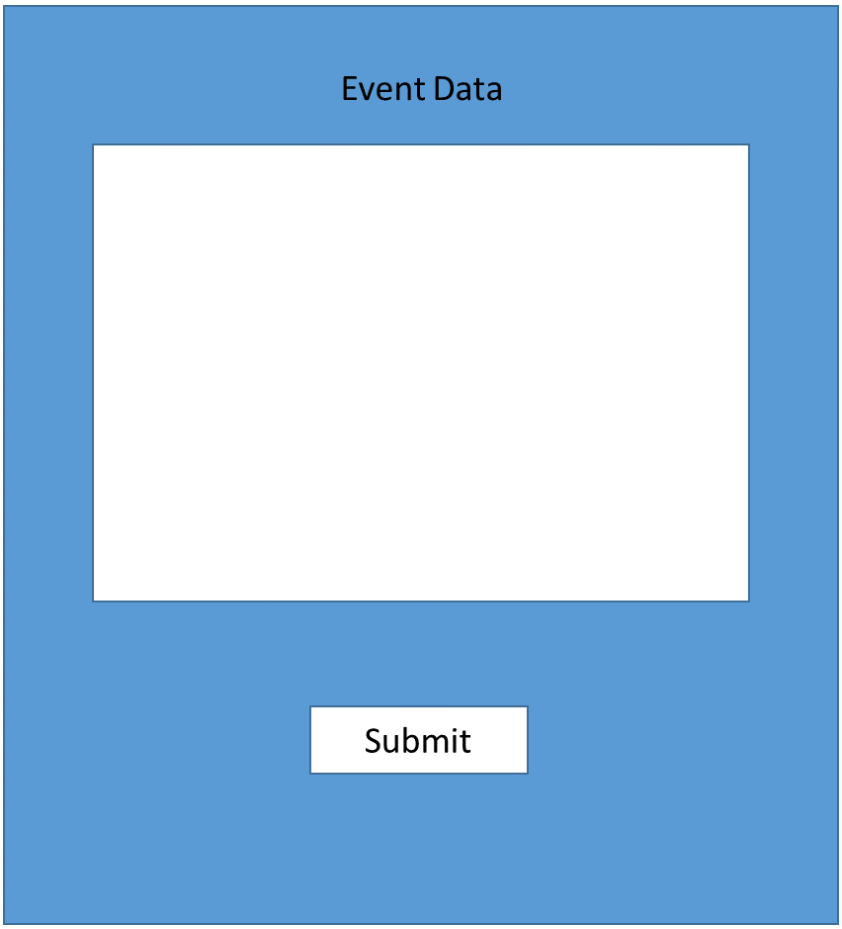
This form is not presented digitally to the user. Instead it takes input from the event screen and generates a detention form to be printed and given to the student. The fields will be filled out with the data entered via the previous screen using pythons string replace functionality

2.4.9 PERSONAL EVENTS SCREEN MOCKUP



This form will display events that the currently logged in user is supposed to be attending. It will allow them to search through them based on a time frame of either a week or a month. It will display all relevant event data in the table

2.4.10 REGISTRATION SCREEN MOCKUP



this screen needs to be simple yet efficient at displaying data on many students

it must also allow for user input to register attendance

it should display key details of the current event at the top of the form

it should allow the user to submit the completed register at the press of a button

to achieve this I will utilise 3 objects:

- A tableView for student data
- A label for event data
- A button for user submission

2.5 ALGORITHMS

To see clearer text of the algorithms please see the attached documents in the 'Pseudocode' folder, the formatting of the plaintext algorithms doesn't work well with word so to see formatting also see the pseudocode folder.

2.5.1 USER OBJECT

```
START UserCreation

      class User()
            public userid
            public accesslevel

            public procedure new()
                  userid = None
                  accesslevel = None
            endprocedure
      endclass
HALT
```

```
START UserCreation

    class User()
        public userid
        public accesslevel

        public procedure new()
            userid = None
            accesslevel = None
        endprocedure
    endclass
HALT
```

This is a abstract class that holds data needed throughout the program relating to the currently logged in user. It is created at the login stage and is destroyed upon logout

2.5.2 LOGIN SCREEN OBJECT

```
START UserLogin **Uses the User Object Defined by User() class**

      class Login_Screen_Gui
            new btn_login //defines GUI objects
            new btn_exit
            new inp_username //text boxes
            new inp_password

            public procedure new()
                  inp_username.clearText
                  inp_password.clearText
                  instance.constructGui()
                  instance.display()
            endprocedure

      endclass




      class Login_Screen inherits Login_Screen_Gui
            private username //sets up variables to be used (private as
contain sensitive data)
            private password
            private teachers
            private students
            private user
            public currentuser
            private temp_accessLevel

            public procedure new()
                  super.new()
                  currentuser = User()

                  if super.btn_login.pressed()
                        _main()
                  else super.btn_exit.pressed()
                        HALT
                  endif
            endprocedure

            private procedure _main() //private as password password must
be secure
                  username = super.inp_username.getText()
                  if username is None
                        print("Please enter a username")
                  endif
                  password = super.inp_password.getText()
                  if password is None
```

```
                    print("please enter a password")
                endif
                teachers = openRead("tbl_teachers")
                students = openRead("tbl_students") //reads file from
storage
                hashed_password = hashfunc.hashed(password) //where
hashfunc is a inbuilt function
                for i = 0 to length(students)
                    if students[i][1] = username //second column of
each user row is username
                        user =students[i].read_line
                        temp_accessLevel = "Student"
                        break
                    else
                        next i
                    endif
                endfor
                if user is none
                    for i = 0 to length(teachers)
                        if teachers[i][1] = username
                            user = teachers[i].read_line
                            temp_accessLevel = "Teacher"
                        else
                            next i
                        endif
                    endfor

                    if user is none
                        print("no such user")
                    else
                        if user[3] =  hashed_password //third column
of each user is a hashed password
                            //login successful
                            //set up user object with key user
information (access level etc.)
                            currentuser.userid = user[0] //first
column of each user is a unique userid
                            currentuser.accesslevel =
temp_accessLevel
                            print("login successful")
                            splash_Window = new Splash_Screen()
//constructs the splash screen
                            super.close()
                        elseif
                            //password incorrect
                            print("password is incorrect")
                endif
            endprocedure
        endclass

login_Window = new Login_Screen()

HALT
```

```
START UserLogin **Uses the User Object Defined by User() class**

    class Login_Screen_Gui
        new btn_login //defines GUI objects
        new btn_exit
        new inp_username //text boxes
        new inp_password

        public procedure new()
            inp_username.clearText
            inp_password.clearText
            instance.constructGui()
            instance.display()
        endprocedure

    endclass



    class Login_Screen inherits Login_Screen_Gui
        private username //sets up variables to be used (private as contain sensitive data)
        private password
        private teachers
        private students
        private user
        public currentuser
        private temp_accessLevel

        public procedure new()
            super.new()
            currentuser = User()

            if super.btn_login.pressed()
                _main()
            else super.btn_exit.pressed()
                HALT
            endif
        endprocedure

        private procedure _main() //private as password password must be secure
            username = super.inp_username.getText()
            password = super.inp_password.getText()
            teachers = openRead("tbl_teachers")
            students = openRead("tbl_students") //reads file from storage
            hashed_password = hashfunc.hashed(password) //where hashfunc is a inbuilt function
            for i = 0 to length(students)
                if students[i][1] = username //second column of each user row is username
                    user =students[i].read_line
                    temp_accessLevel = "Student"
                    break
                else
                    next i
                endif
            endfor
            if user is none
                for i = 0 to length(teachers)
                    if teachers[i][1] = username
                        user = teachers[i].read_line
                        temp_accessLevel = "Teacher"
                    else
                        next i
                    endif
                endfor

                if user is none
                    print("no such user")
                        //set up user object with key user information (access level etc.)
                        currentuser.userid = user[0] //first column of each user is a unique userid
                        currentuser.accesslevel = temp_accessLevel
                        print("login successful")
                        splash_Window = new Splash_Screen() //constructs the splash screen
                        super.close()
```

2.5.3 SPLASH SCREEN OBJECT

```
START SplashScreen **Uses predefined User Object, defined by User() class
and created in Login() class**
**Also will import classes from the rest of the program as it constructs
all other objects**

      class Splash_Screen_Gui
            new btn_create_new_user
            new btn_analytics
            new btn_add_event
            new btn_personal_events
            new btn_modify_user
            new btn_registration
            new btn_exit
            new btn_logout

            public procedure new()
                  instance.constructGui()
                  instance.display()
            endprocedure


      endclass

      class Splash_Screen inherits from Splash_Screen_Gui //collection of
constructor methods
            super.new()
            public create_new_user_Window


            public procedure new()
                  if super.btn_create_new_user.pressed()
                        if currentuser.accesslevel != 1 //where a teacher
has accesslevel of 1
                              print("ERROR Access level not high enough")
                        else
                              create_new_user_Window = new
Create_New_User_Screen()
                        endif

                  elseif super.btn_analytics.pressed()
                        if currentuser.accesslevel != 1
                              print("ERROR Access level not high enough")
                        else
                              analytics_Window = new Analytics_Screen()

                        endif

                  elseif super.btn_add_event.pressed()
                        if currentuser.accesslevel != 1
                              print("ERROR Access level not high enough")
                        else
```

```
                              add_event_Window = new Add_Event_Screen()
                        endif
                  elseif super.btn_personal_events.pressed()
                        personal_events_Window = new
Personal_Events_Screen()
                  elseif super.btn_modify_user.pressed()
                        if currentuser.accesslevel != 1
                              print("ERROR Access level not high enough")
                        else
                              modify_user_Window = new Modify_User_Screen()
                        endif
                  elseif super.btn_registration.pressed()
                        if currentuser.accesslevel != 1
                              print("ERROR Access level not high enough")
                        else
                              registration_Window = new Registration_Screen
                  elseif super.btn_exit.pressed()
                        HALT
                  elseif super.btn_logout.pressed()
                        _logout()

                  endif

            endprocedure

            private procedure _logout()
                  currentuser = None
                  login_Window = new Login_Screen()
                  super.close()
            endprocedure

      endclass

      splash_Window = new Splash_Screen()

HALT
```

```
START SplashScreen **Uses predefined User Object, defined by User() class and created in Login() class**
**Also will import classes from the rest of the program as it constructs all other objects**

    class Splash_Screen_Gui
        new btn_create_new_user
        new btn_analytics
        new btn_add_event
        new btn_personal_events
        new btn_modify_user
        new btn_registration
        new btn_exit
        new btn_logout

        public procedure new()
            instance.constructGui()
            instance.display()
        endprocedure


    endclass

    class Splash_Screen inherits from Splash_Screen_Gui //collection of constructor methods
        super.new()
        public create_new_user_Window


        public procedure new()
            if super.btn_create_new_user.pressed()
                if currentuser.accesslevel != 1 //where a teacher has accesslevel of 1
                    print("ERROR Access level not high enough")
                else
                    create_new_user_Window = new Create_New_User_Screen()
                endif

            elseif super.btn_analytics.pressed()
                if currentuser.accesslevel != 1
                    print("ERROR Access level not high enough")
                else
                    analytics_Window = new Analytics_Screen()
                endif

            elseif super.btn_add_event.pressed()
                if currentuser.accesslevel != 1
                    print("ERROR Access level not high enough")
                else
                    add_event_Window = new Add_Event_Screen()
                endif
            elseif super.btn_personal_events.pressed()
                personal_events_Window = new Personal_Events_Screen()
            elseif super.btn_modify_user.pressed()
                if currentuser.accesslevel != 1
                    print("ERROR Access level not high enough")
                else
                    modify_user_Window = new Modify_User_Screen()
                endif
            elseif super.btn_registration.pressed()
                if currentuser.accesslevel != 1
                    print("ERROR Access level not high enough")
                else
                    registration_Window = new Registration_Screen
            elseif super.btn_exit.pressed()
                HALT
            elseif super.btn_logout.pressed()
                _logout()

            endif

        endprocedure
            currentuser = None
            login_Window = new Login_Screen()
            super.close()
        endprocedure

    endclass

```

2.5.4 CREATE NEW USER OBJECT

```
START Create_New_user  **Uses predefined User Object, defined by User()
class and created in Login() class**

      class Create_New_User_Screen_Gui
            new inp_Forename
            new inp_Surname
            new inp_OtherNames
            new inp_FormName
            new inp_Username
            new inp_Password
            new inp_DateOfBirth
            new inp_SchoolYear
            new radGroup_Gender //Group of radio buttons
            new Gender.rad_Male //adds a radio button to the group
            new Gender.rad_Female
            new Gender.rad_Other
            new checkbox_Teacher  //checkbox if user is a teacher
            new btn_AddUser

            public procedure new()
                  inp_Forename.clearText
                  inp_Surname.clearText
                  inp_OtherNames.clearText
                  inp_FormName.clearText
                  inp_Username.clearText
                  inp_Password.clearText
                  inp_DateOfBirth.clearText
                  inp_SchoolYear.clearText
                  instance.constructGui()
                  instance.display()
            endprocedure
      endclass


      class Create_New_User_Screen inherits Create_New_User_Screen_Gui

            private forename
            private surname
            private otherName
            private formName
            private username
            private password
            private hashed_password
            private dateOfBirth
            private schoolYear
            private gender
            private array teachers
            private array students
            public procedure new()
                  super.new()
                  if super.btn_AddUser.pressed()
                        _adduser()
```

```
            endprocedure

            private procedure _getdata()
                  forename = super.inp_Forename.getText()
                  surname = super.inp_Surname.getText()
                  otherName = super.inp_OtherNames.getText()
                  formName = super.inp_FormName.getText()
                  username = super.inp_Username.getText()
                  password = super.inp_Password.getText()
                  dateOfBirth = super.inp_DateOfBirth.getText()
                  schoolYear = super.inp_SchoolYear.getText()
                  if super.Gender.rad_male.selected()
                        gender = 0 //where male has the gender id 0
                  elseif super.Gender.rad_Female.selected()
                        gender = 1 //where female has the gender id 1
                  else
                        gender = 2 //where 'other' has the gender id 2
                  endif
            endprocedure

            private procedure _adduser()
                  _getdata()
                  hashed_password = hashfunc.hashed(password) //where
hashfunc is a inbuilt function
                  if super.checkbox_Teacher.checked()
                        teachers = openReadWrite("tbl_teachers")
                        id = teachers[length(teachers)][0] + 1 //where the
first row in a column is the id
                        str_input =
string(id,forename,otherName,surname,schoolYear,formName,username,hashed_p
assword,dateOfBirth,gender)
                        teachers.writeLine(str_input)
                        teachers.save()
                        teachers.close()

                  else
                        students = openReadWrite("tbl_students")
                        id = students[length(students)][0] + 1
                        str_input =
string(id,forename,otherName,surname,schoolYear,formName,username,hashed_p
assword,dateOfBirth,gender)
`                       students.writeLine(str_input)
                        students.save()
                        students.close()

                  endif
            endprocedure
      endclass

      create_new_user_Window = new Create_New_User_Screen
HALT
```

```
START Create_New_user  **Uses predefined User Object, defined by User() class and created in Login() class**

    class Create_New_User_Screen_Gui
        new inp_Forename
        new inp_Surname
        new inp_OtherNames
        new inp_FormName
        new inp_Username
        new inp_Password
        new inp_DateOfBirth
        new inp_SchoolYear
        new radGroup_Gender //Group of radio buttons
        new Gender.rad_Male //adds a radio button to the group
        new Gender.rad_Female
        new Gender.rad_Other
        new checkbox_Teacher  //checkbox if user is a teacher
        new btn_AddUser

        public procedure new()
            inp_Forename.clearText
            inp_Surname.clearText
            inp_OtherNames.clearText
            inp_FormName.clearText
            inp_Username.clearText
            inp_Password.clearText
            inp_DateOfBirth.clearText
            inp_SchoolYear.clearText
            instance.constructGui()
            instance.display()
        endprocedure
    endclass


    class Create_New_User_Screen inherits Create_New_User_Screen_Gui

        private forename
        private surname
        private otherName
        private formName
        private username
        private password
        private hashed_password
        private dateOfBirth
        private schoolYear
        private gender
        private array teachers
        private array students
        public procedure new()
            super.new()
            if super.btn_AddUser.pressed()
                _adduser()
            endprocedure

        private procedure _getdata()
            forename = super.inp_Forename.getText()
            surname = super.inp_Surname.getText()
            otherName = super.inp_OtherNames.getText()
            formName = super.inp_FormName.getText()
            username = super.inp_Username.getText()
            password = super.inp_Password.getText()
            dateOfBirth = super.inp_DateOfBirth.getText()
            schoolYear = super.inp_SchoolYear.getText()
            if super.Gender.rad_male.selected()
                gender = 0 //where male has the gender id 0
            elseif super.Gender.rad_Female.selected()
                gender = 1 //where female has the gender id 1
            else
                gender = 2 //where 'other' has the gender id 2
            endif
        endprocedure

        private procedure _adduser()
            _getdata()
            hashed_password = hashfunc.hashed(password) //where hashfunc is a inbuilt function
            if super.checkbox_Teacher.checked()
                teachers = openReadWrite("tbl_teachers")
                id = teachers[length(teachers)][0] + 1 //where the first row in a column is the id
                str_input = string(id,forename,otherName,surname,schoolYear,formName,username,hashed_password,dateOfBirth,gender)
                teachers.writeLine(str_input)
                teachers.save()
                teachers.close()

            else
                students = openReadWrite("tbl_students")
                id = students[length(students)][0] + 1
                str_input = string(id,forename,otherName,surname,schoolYear,formName,username,hashed_password,dateOfBirth,gender)
                students.writeLine(str_input)
                students.save()
            formName = super.inp_FormName.getText()
            username = super.inp_Username.getText()
            password = super.inp_Password.getText()
            dateOfBirth = super.inp_DateOfBirth.getText()
            schoolYear = super.inp_SchoolYear.getText()
            if super.Gender.rad_male.selected()
                gender = 0 //where male has the gender id 0
            elseif super.Gender.rad_Female.selected()
                gender = 1 //where female has the gender id 1
            else
                gender = 2 //where 'other' has the gender id 2
            endif
        endprocedure

        private procedure _adduser()
            _getdata()
            hashed_password = hashfunc.hashed(password) //where hashfunc is a inbuilt function
            if super.checkbox_Teacher.checked()
                teachers = openReadWrite("tbl_teachers")
                id = teachers[length(teachers)][0] + 1 //where the first row in a column is the id
                str_input = string(id,forename,otherName,surname,schoolYear,formName,username,hashed_password,dateOfBirth,gender)
                teachers.writeLine(str_input)
                teachers.save()
                teachers.close()

            else
                students = openReadWrite("tbl_students")
                id = students[length(students)][0] + 1
                str_input = string(id,forename,otherName,surname,schoolYear,formName,username,hashed_password,dateOfBirth,gender)
                students.writeLine(str_input)
                students.save()
                students.close()

            endif
        endprocedure
    endclass

    create_new_user_Window = new Create_New_User_Screen
HALT
```

2.5.5 CREATE NEW EVENT OBJECT

```
START Create_New_Event

      class Create_New_Event_Screen_Gui
            new radGroup EventType
            new radGroup GroupType
            new EventType.rad_InternalEvent
            new EventType.rad_ExternalEvent
            new EventType.rad_Detention
            new GroupType.rad_Students
            new GroupType.rad_Forms
            new GroupType.rad_Years
            new GroupType.rad_Teachers
            new inp_GroupName
            new inp_TeacherName
            new btn_GroupIn
            new btn_TeacherIn
            new inp_TimeDateIn
            new inp_Duration
            new inp_Room
            new inp_Description
            new btn_AddEvent
            public procedure new()
                  inp_GroupName.clearText
                  inp_TeacherName.clearText
                  inp_TimeDateIn.clearText
                  inp_Duration.clearText
                  inp_Room.clearText
                  inp_Description.clearText
                  instance.constructGui()
                  instance.display()
            endprocedure

      endclass


      class Create_New_Event_Screen inherits Create_New_Event_Screen_Gui
            private eventType
            private timeDate
            private duration
            private room
            private description
            private array particiapant_array
            private array teacher_array
            private participants
            private teachers
            private array user
            private groupname
            private array appdata
            private events
            private comitTeacher
            private comitStudent
            private eventId
```

```
            public procedure new()
                  super.new()
                  if super.btn_GroupIn.pressed()
                        _addpart()
                  endif
                  if super.btn_TeacherIn.pressed()
                        _addteach()
                  endif
                  if super.btn_AddEvent.pressed()
                        _addevent()
                  endif
            endprocedure

            private procedure _addpart()
                  user = None
                  groupname = super.inp_GroupName.getText()
                  if super.GroupType.rad_Students.selected()
                        participants = openReadWrite("tbl_students")
                        for i = 0 to length(participants)
                              if participants[i][1] = groupname //second
column of each user row is username
                                    user.append(participants[i].read_line)
                                    break
                              else
                                    next i
                              endif
                        endfor
                        if user is none
                              print("No such student")
                        endif
                        particiapant_array.append(user)
                  elseif super.GroupType.rad_Forms.selected()
                        participants = openReadWrite("tbl_students")
                        for i = 0 to length(participants)
                              if participants[i][5] = groupname //6th
column of each user row is their form
                                    user.append(participants[i].read_line)
                                    next i
                              else
                                    next i
                              endif
                        endfor
                        if user is none
                              print("No such form group")
                        endif
                        particiapant_array.append(user)
                  elseif super.GroupType.rad_Years.selected()
                        participants = openReadWrite("tbl_students")
                        for i = 0 to length(participants)
                              if participants[i][4] = groupname //5th
column of each user row is their year
                                    user.append(participants[i].read_line)
                                    next i
                              else
```

```
                              next i
                        endif
                  endfor
                  if user is none
                        print("No such year group")
                  endif
                  particiapant_array.append(user)
            elseif super.GroupType.rad_Teachers.selected()
                  print("Please use teacher input to enter teacher
IDs")
            endif
      endprocedure

      private procedure _addteach()
            user = None
            teacherName = super.inp_TeacherName.getText()
            teachers = openReadWrite("tbl_teachers")
            for i = 0 to length(teachers)
                  if teachers[i][1] = teacherName //1st column in
each row is their username
                        user.append(teachers[i].read_line)
                        break
                  else
                        next i
                  endif
            endfor
            if user is none
                  print("No such year group")
            endif
            teacher_array.append(user)
      endprocedure



      private procedure _addevent()
            if super.EventType.rad_ExternalEvent.selected()
                  eventType = "External"
            elseif super.EventType.rad_InternalEvent.selected()
                  eventType = "Internal"
            elseif super.EventType.rad_Detention.selected()
                  eventType = "Detention"
            endif
            timeDate = super.inp_TimeDateIn.getText()
            duration = super.inp_Duration.getText()
            room = super.inp_Room.getText()
            description = super.inp_Description.getText()
            events = openReadWrite("tbl_events")
            comitStudent = openReadWrite("tbl_comStudents")
            comitTeacher = openReadWrite("tbl_comTeachers")
            eventId = events[length(events)][0] + 1
            for i = 0 to length(particiapant_array)
                  appdata = None
```

```
                        appdata =
[(comitStudent[length(comitStudent)][0]+1), particiapant_array[i][0],
eventId]
                        comitStudent.append(appdata)
                        next i
                  endfor
                  comitStudent.save()
                  comitStudent.close()

                  for i = 0 to length(teacher_array)
                        appdata = None
                        appdata =
[(comitTeacher[length(comitTeacher)][0]+1), teacher_array[i][0], eventId]
                        comitTeacher.append(appdata)
                        next i
                  endfor
                  comitTeacher.save()
                  comitTeacher.close()

                  appdata = None
                  appdata = [eventId, room, eventType, timeDate,
description, duration]
                  events.append(appdata)
                  events.save()
                  events.close()
            endprocedure
      endclass

create_new_event_Window = new Create_New_Event_Screen

HALT
```

```
START Create_New_Event

    class Create_New_Event_Screen_Gui
        new radGroup EventType
        new radGroup GroupType
        new EventType.rad_InternalEvent
        new EventType.rad_ExternalEvent
        new EventType.rad_Detention
        new GroupType.rad_Students
        new GroupType.rad_Forms
        new GroupType.rad_Years
        new GroupType.rad_Teachers
        new inp_GroupName
        new inp_TeacherName
        new btn_GroupIn
        new btn_TeacherIn
        new inp_TimeDateIn
        new inp_Duration
        new inp_Room
        new inp_Description
        new btn_AddEvent
        public procedure new()
            inp_GroupName.clearText
            inp_TeacherName.clearText
            inp_TimeDateIn.clearText
            inp_Duration.clearText
            inp_Room.clearText
            inp_Description.clearText
            instance.constructGui()
            instance.display()
        endprocedure

    endclass


    class Create_New_Event_Screen inherits Create_New_Event_Screen_Gui
        private eventType
        private timeDate
        private duration
        private room
        private description
        private array participant_array
        private array teacher_array
        private participants
        private teachers
        private array user
        private groupname
        private array appdata
        private events
        private comitTeacher
        private comitStudent
        private eventId
        public procedure new()
            super.new()
            if super.btn_GroupIn.pressed()
                _addpart()
            endif
            if super.btn_TeacherIn.pressed()
                _addteach()
            endif
            if super.btn_AddEvent.pressed()
                _addevent()
            endif
        endprocedure

        private procedure _addpart()
            user = None
            groupname = super.inp_GroupName.getText()
            if super.GroupType.rad_Students.selected()
                participants = openReadWrite("tbl_students")
                for i = 0 to length(participants)
                    if participants[i][1] = groupname //second column of each user row is username
                        user.append(participants[i].read_line)
                        break
                    else
                        next i
                    endif
                endfor
                if user is none
                    print("No such student")
                endif
                participant_array.append(user)
            elseif super.GroupType.rad_Forms.selected()
                participants = openReadWrite("tbl_students")
                for i = 0 to length(participants)
                    if participants[i][5] = groupname //5th column of each user row is their form
                        user.append(participants[i].read_line)
                        next i
                    else
                        next i
                    endif
                endfor
                if user is none
                    print("No such form group")
                endif
                participant_array.append(user)
            elseif super.GroupType.rad_Years.selected()
                participants = openReadWrite("tbl_students")
                for i = 0 to length(participants)
                    if participants[i][4] = groupname //5th column of each user row is their year
                        user.append(participants[i].read_line)
                        next i
                    else
                        next i
                    endif
                endfor
                if user is none
                    print("No such year group")
                endif
                participant_array.append(user)
            elseif super.GroupType.rad_Teachers.selected()
                print("Please use teacher input to enter teacher IDs")
            endif
        endprocedure

            for i = 0 to length(teachers)
                if teachers[i][1] = teacherName //1st column in each row is their username
                    user.append(teachers[i].read_line)
                    break
                else
                    next i
                endif
            endfor
            if user is none
                print("No such year group")
            endif
            teacher_array.append(user)
        endprocedure


        private procedure _addevent()
            if super.EventType.rad_ExternalEvent.selected()
                eventType = "External"
            elseif super.EventType.rad_InternalEvent.selected()
                eventType = "Internal"
            elseif super.EventType.rad_Detention.selected()
                eventType = "Detention"
            endif
            timeDate = super.inp_TimeDateIn.getText()
            duration = super.inp_Duration.getText()
            room = super.inp_Room.getText()
            description = super.inp_Description.getText()
            events = openReadWrite("tbl_events")
            comitStudent = openReadWrite("tbl_comStudents")
            comitTeacher = openReadWrite("tbl_comTeachers")
            eventId = events[length(events)][0] + 1
            for i = 0 to length(participant_array)
                appdata = None
                appdata = [(comitStudent[length(comitStudent)][0]+1), participant_array[i][0], eventId]
                comitStudent.append(appdata)
                next i
            endfor
            comitStudent.save()
            comitStudent.close()

            for i = 0 to length(teacher_array)
                appdata = None
                appdata = [(comitTeacher[length(comitTeacher)][0]+1), teacher_array[i][0], eventId]
                comitTeacher.append(appdata)
                next i
            endfor
            comitTeacher.save()
            comitTeacher.close()

            appdata = None
            appdata = [eventId, room, eventType, timeDate, description, duration]
            events.append(appdata)
            events.save()
            events.close()
        endprocedure
    endclass

create_new_event_Window = new Create_New_Event_Screen

HALT
```

2.5.6 SEE EVENTS OBJECT

```
START Personal_Events **Uses predefined User Object, defined by User()
class and created in Login() class**

      class Personal_Events_Screen_Gui
            new radGroup TimeFrame
            new radGroup EventType
            new TimeFrame.rad_Month
            new TimeFrame.rad_Week
            new EventType.rad_InternalEvent
            new EventType.rad_ExternalEvent
            new EventType.rad_Detention
            new btn_Next
            new btn_Previous
            new btn_Init
            new cal_DateIn //a calendar object externally defined
            new tbl_tableView //a table object externally defined

            public procedure new()

                  instance.constructGui()
                  instance.display()

            endprocedure


      endclass


      class Personal_Events_Screen inherits Personal_Events_Screen_Gui
            private eventType
            private timeFrame
            private events
            private comits
            private lowerbound
            private upperbound
            private userCommits
            private array userCommitsPassA
            private array userCommitsPassB
            private array tableData
            private tempbound
            public procedure new()
                  if super.TimeFrame.rad_Month.selected()
                        timeFrame = 1
                  elseif super.TimeFrame.rad_Week.selected()
                        timeFrame = 0
                  endif

                  lowerbound = None
                  upperbound = None
                  super.new()
```

```
        if super.btn_Init.pressed()
              lowerbound = super.cal_DateIn.date()
              if timeFrame = 1
                    upperbound = lowerbound[month] + 1
              elseif timeFrame = 0
                    upperbound = lowerbound[day] + 7
              endif
        elseif super.btn_Next.pressed()
              if lowerbound is not None
                    lowerbound
              else
                    print("please initialse first")
              endif
        elseif super.btn_Previous.pressed()
              if lowerbound is not None
                    if timeFrame = 1
                          upperbound = upperbound[month] - 1
                          lowerbound = tempbound[month] - 1
                    elseif timeFrame = 0
                          upperbound = upperbound[day] - 7
                          lowerbound = lowerbound[day] - 7
                    endif

              else
                    print("please initialse first")
              endif
        endif


  private procedure _main(lb,ub)
        events = openRead("tbl_events")
        if super.EventType.rad_Detention.selected()
              eventType = "Detention"
        elseif super.EventType.rad_ExternalEvent.selected()
              eventType = "External"
        elseif super.EventType.rad_InternalEvent.selected()
              eventType = "Internal"
        else
              eventType = None
        endif



        if currentuser.accesslevel = "Teacher"
              userCommits = _teacher()
        elseif currentuser.accesslevel = "Student"
              userCommits = _student()
        endif

        for i = 0 to length(userCommits)
              tableData.append(events[userCommits[i][2])
              next i
        endfor
```

```
                super.tbl_tableView.setData(tableData)



        endprocedure


        private comitTeacher

        private function _teacher(lb,ub)
              comitTeacher = openRead("tbl_comTeachers")
              for i = 0 to length(comitTeacher)
                        if comitTeacher[i][1] = currentuser.userid

    userCommitsPassA.append(comitTeacher[i])
                              next i
                        else
                              next i
                        endif
              endfor

              if eventType not None
                    for i = 0 to length(userCommitsPassA)
                        if events[userCommitsPassA[i][2]][2] =
eventType

    userCommitsPassB.append(userCommitsPassA[i])
                              next i
                        else
                              next i
                        endif
                    endfor
              else
                    userCommitsPassB = userCommitsPassA
              endif
              return userCommitsPassB
        endfunction


        private comitStudent
        private function _student(lb,ub)
              comitStudent = openRead("tbl_comStudents")
              for i = 0 to length(comitStudent)
                    if comitStudent[i][1] = currentuser.userid
                          userCommitsPassA.append(comitStudent[i])
                          next i
                    else
                          next i
                    endif
              endfor

              if eventType not None
                    for i = 0 to length(userCommitsPassA)
```

```
                              if events[userCommitsPassA[i][2]][2] =
eventType

      userCommitsPassB.append(userCommitsPassA[i])
                                    next i
                              else
                                    next i
                              endif
                        endfor
                  else
                        userCommitsPassB = userCommitsPassA
                  endif
                  return userCommitsPassB
            endfunction
      endclass

HALT
```

```
START Personal_Events **Uses predefined User Object, defined by User() class and created in Login() class**

    class Personal_Events_Screen_Gui
        new radGroup TimeFrame
        new radGroup EventType
        new TimeFrame.rad_Month
        new TimeFrame.rad_Week
        new EventType.rad_InternalEvent
        new EventType.rad_ExternalEvent
        new EventType.rad_Detention
        new btn_Next
        new btn_Previous
        new btn_Init
        new cal_DateIn //a calendar object externally defined
        new tbl_tableView //a table object externally defined

        public procedure new()

            self.constructGui()
            self.show()

        endprocedure


    endclass


    class Personal_Events_Screen inherits Personal_Events_Screen_Gui
        private eventType
        private timeFrame
        private events
        private comits
        private lowerbound
        private upperbound
        private userCommits
        private array userCommitsPassA
        private array userCommitsPassB
        private array tableData
        private tempbound
        public procedure new()
            if super.TimeFrame.rad_Month.selected()
                timeFrame = 1
            elseif super.TimeFrame.rad_Week.selected()
                timeFrame = 0
            endif

            lowerbound = None
            upperbound = None
            super.new()
            if super.btn_Init.pressed()
                lowerbound = super.cal_DateIn.date()
                if timeFrame = 1
                    upperbound = lowerbound[month] + 1
                elseif timeFrame = 0
                    upperbound = lowerbound[day] + 7
                endif
            elseif super.btn_Next.pressed()
                if lowerbound is not None
                    lowerbound
                else
                    print("please initalse first")
                endif
            elseif super.btn_Previous.pressed()
                if lowerbound is not None
                    if timeFrame = 1
                        upperbound = upperbound[month] - 1
                        lowerbound = tempbound[month] - 1
                    elseif timeFrame = 0
                        upperbound = upperbound[day] - 7
                        lowerbound = lowerbound[day] - 7
                    endif

                else
                    print("please initalse first")
                endif
            endif


        private procedure _main(lb,ub)
            events = openRead("tbl_events")
            if super.EventType.rad_Detention.selected()
                eventType = "Detention"
            elseif super.EventType.rad_ExternalEvent.selected()
                eventType = "External"
            elseif super.EventType.rad_InternalEvent.selected()
                eventType = "Internal"


            if currentuser.accesslevel = "Teacher"
                userCommits = _teacher()
            elseif currentuser.accesslevel = "Student"
                userCommits = _student()
            endif

            for i = 0 to length(userCommits)
                tableData.append(events[userCommits[i][2]])
                next i
            endfor

            super.tbl_tableView.setData(tableData)


        endprocedure

        private comitTeacher

        private function _teacher(lb,ub)
            comitTeacher = openRead("tbl_comTeachers")
            for i = 0 to length(comitTeacher)
                    if comitTeacher[i][1] = currentuser.userid
                        userCommitsPassA.append(comitTeacher[i])
                        next i
                    else
                        next i
                    endif
            endfor

            if eventType not None
                for i = 0 to length(userCommitsPassA)
                    if events[userCommitsPassA[i][2]][2] = eventType
                        userCommitsPassB.append(userCommitsPassA[i])
                        next i
                    else
                        next i
                    endif
                endfor
            else
                userCommitsPassB = userCommitsPassA
            endif
            return userCommitsPassB
        endfunction

        private comitStudent
        private function _student(lb,ub)
            comitStudent = openRead("tbl_comStudents")
            for i = 0 to length(comitStudent)
                if comitStudent[i][1] = currentuser.userid
                    userCommitsPassA.append(comitStudent[i])
                    next i
                else
                    next i
                endif
            endfor

            if eventType not None
                for i = 0 to length(userCommitsPassA)
                    if events[userCommitsPassA[i][2]][2] = eventType
                        userCommitsPassB.append(userCommitsPassA[i])
                        next i
                    else
                        next i
                    endif
                endfor
            else
                userCommitsPassB = userCommitsPassA
            endif
            return userCommitsPassB
        endfunction
    endclass
```

2.5.7 ANALYTICS OBJECT

```
START Advanced Analytics


      class Advanced_Analytics_Screen_Gui


            new radGroup GroupType
            new radGroup EventType
            new radGroup TimeFrame
            new GroupType.rad_Student
            new GroupType.rad_Form
            new GroupType.rad_Year
            new GroupType.rad_Teacher
            new EventType.rad_Internal
            new EventType.rad_External
            new EventType.rad_Detention
            new EventType.rad_All
            new TimeFrame.rad_Day
            new TimeFrame.rad_Week
            new TimeFrame.rad_Month
            new TimeFrame.rad_Year
            new inp_GroupName
            new btn_GroupIn
            new cal_DateIn
            new tbl_DataOut
            new cht_ChartDataOut //Creates chart object for data output
            new btn_TimeMissed
            new btn_Attendance

            public procedure new()
                  GroupType.clearText
                  instance.constructGui()
                  instance.display()

            endprocedure

      endclass



      class Advanced_Analytics_Screen inherits
Advanced_Analytics_Screen_Gui

            private groupName
            private date
            private endDate
            private groupType
            private eventType
            private timeFrame
            public procedure new()
                  super.new()
                  date = super.cal_DateIn.date()
```

```
                  if super.rad_Day.selected()
                        endDate = date[day] + 1
                  elseif super.rad_Week.selected()
                        endDate = date[day] + 7
                  elseif super.rad_Month.selected()
                        endDate = date[month] + 1
                  elseif super.rad_Year.selected()
                        endDate = date[year] + 1
                  endif
                  if super.btn_GroupIn.pressed()
                        groupName = super.inp_GroupName.getText()
                        if groupName.getType() != "String" //Data type
check
                              print("Please enter a valid group name")
                              groupName = None
                        endif
                  endif
                  if super.btn_Attendance.pressed()
                        if groupName is not None //presence check for a
groupName having been entered
                              _attendance()
                        else
                              print("Enter a group name first")
                        endif
                  elseif super.btn_TimeMissed.pressed()
                        if groupName is not None
                              _eventCount()
                        else
                              print("Enter a group name first")
                        endif
                  endif
            endprocedure

            private events
            private comits
            private comitsPassA
            private comitsPassB
            private students
            private teachers
            private curuserid
            private anadata

            private procedure _getdata()
                  events = openRead("tbl_events")
                  if super.rad_Student.selected()
                        students = openRead("tbl_students")
                        for i = 0 to length(students) //finds userid from
username
                              if students[i][1] = groupName
                                    curuserid = students[i][0]
                              else
                                    next i
                              endif
                        endfor
```

```
                        if curuserid is None
                              print("Invalid user")
                        comits = openRead("tbl_ComStudents")
                        for i = 0 to length(comits)//selects comits
relevant to group
                              if comits[i][1] = curuserid
                                    comitsPassA.append(comits[i])
                                    next i
                              else
                                    next i
                              endif
                        endfor

                        if super.rad_All.selected()
                              eventType = None
                        elseif super.rad_Internal.selected()
                              eventType = "Internal"
                        elseif super.rad_External.selected()
                              eventType = "External"
                        elseif super.rad_Detention.selected()
                              eventType = "Detention"
                        endif

                        if eventType is not None
                              for i = 0 to length(comitsPassA)
                                    for j = 0 to length(events)
                                          if events[j][0] =
comitsPassA[i][2]
                                                if events[j][2] = eventType
//where third column is the eventType of the event

      comitsPassB.append(events[j],comitsPassA[i])
                                                      next j
                                                else
                                                      next j
                                                endif
                                          else
                                                next j
                                          endif
                                    endfor
                                    next i
                              endfor
                        endif

                        for i = 0 to lenght(comitsPassB)
                              if comitsPassB[i][0][3] = date
                                    anadata.append(comitsPassB[i])
                                    next i
                              else
                                    next i
                              endif
                        endfor

                  elseif super.rad_Form.selected()
```

```
                        students = openRead("tbl_students")
                        comits = openRead("tbl_comStudents")
                        **FETCHES ALL DATA ON EVENTS THAT MEMBERS OF THE
GROUP ARE DUE TO BE ATTENDING STORED IN 'ANADATA'**
                  elseif super.rad_Year.selected()
                        students = openRead("tbl_students")
                        comits = openRead("tbl_comStudents")
                        **FETCHES ALL DATA ON EVENTS THAT MEMBERS OF THE
GROUP ARE DUE TO BE ATTENDING STORED IN 'ANADATA'**
                  elseif super.rad_Teacher.selected()
                        teachers = openRead("tbl_teachers")
                        comits = openRead("tbl_comTeachers")
                        **FETCHES ALL DATA ON EVENTS THAT MEMBERS OF THE
GROUP ARE DUE TO BE ATTENDING STORED IN 'ANADATA'**
                  endif
                  return anadata

                  **anadata contruction:  [[[event
data],[comitmentdata]],[[event data],[comitmentdata]],.....]**



            endprocedure

            private array data
            private attendcount
            private misscount
            private procedure _attendance()
            private iecount
            private eecount
            private dcount
            private temp_Item
                  attendcount = 0
                  misscount = 0
                  data = _getdata()
                  for i = 0 to length(data)
                        if data[i][1][3] = 0
                              misscount = misscount + 1
                        elseif data[i][1][3] = 1
                               attendcount = attendcount + 1
                        endif
                        next i
                  endfor

      super.tbl_DataOut.setData([groupName,GroupType,attendcount,misscount
])
                  super.cht_ChartDataOut.setData(attendcount:misscount)

            endprocedure

            private procedure _eventCount()
                  data = _getdata()
                  for i = 0 to length(data)
                        temp_Item = data[i][0][2]
```

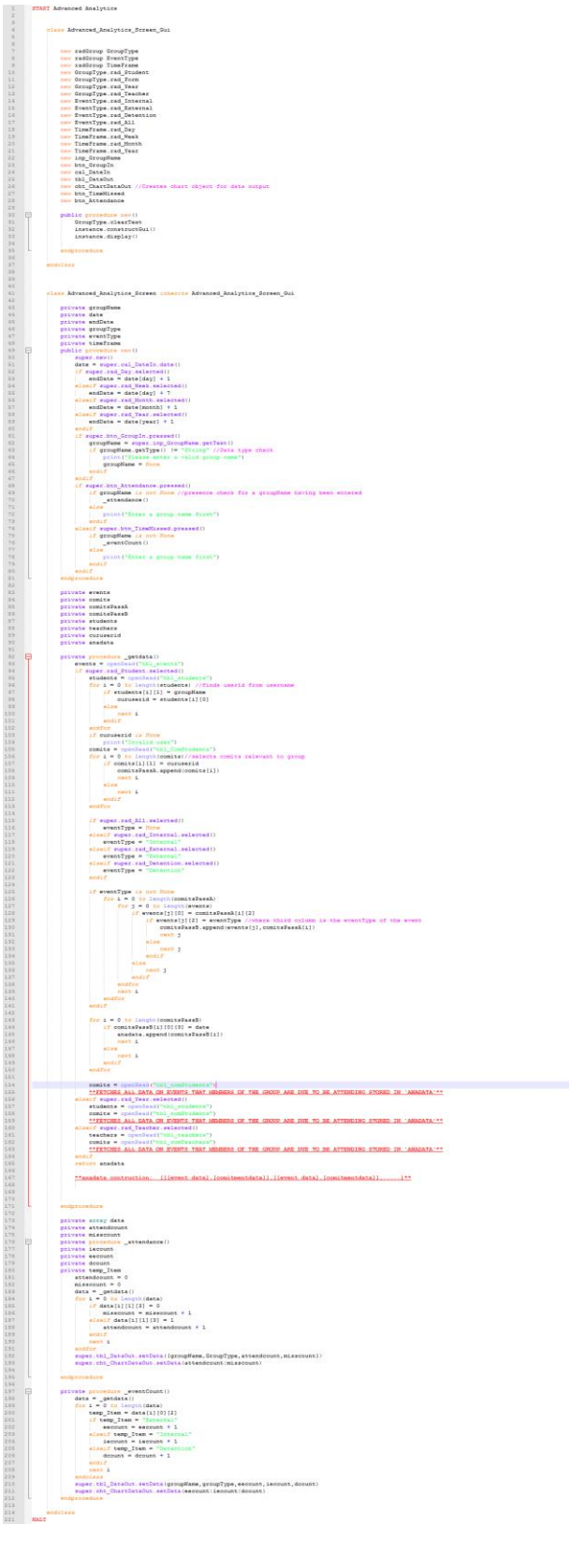
```
                        if temp_Item = "External"
                              eecount = eecount + 1
                        elseif temp_Item = "Internal"
                              iecount = iecount + 1
                        elseif temp_Item = "Detention"
                              dcount = dcount + 1
                        endif
                        next i
                  endclass

      super.tbl_DataOut.setData(groupName,groupType,eecount,iecount,dcount
)
                  super.cht_ChartDataOut.setData(eecount:iecount:dcount)
            endprocedure

      endclass




HALT
```



```
START Advanced_Analytics

    class Advanced_Analytics_Screen_Gui


        new radGroup GroupType
        new radGroup EventType
        new radGroup TimeFrame
        new GroupType.rad_Student
        new GroupType.rad_Form
        new GroupType.rad_Year
        new GroupType.rad_Teacher
        new EventType.rad_Internal
        new EventType.rad_External
        new EventType.rad_Detention
        new EventType.rad_All
        new TimeFrame.rad_Day
        new TimeFrame.rad_Week
        new TimeFrame.rad_Month
        new TimeFrame.rad_Year
        new lng_GroupName
        new btn_GroupIn
        new cal_DateIn
        new tbl_DataOut
        new cht_ChartDataOut //Creates chart object for data output
        new btn_TimeMissed
        new btn_Attendance

        public procedure new()
            GroupType.clearText
            instance.constructGui()
            instance.display()

        endprocedure

    endclass


    class Advanced_Analytics_Screen inherits Advanced_Analytics_Screen_Gui

        private groupName
        private date
        private endDate
        private groupType
        private eventType
        private timeFrame
        public procedure new()
            super.new()
            date = super.cal_DateIn.date()
            if super.rad_Day.selected()
                endDate = date[day] + 1
            elseif super.rad_Week.selected()
                endDate = date[day] + 7
            elseif super.rad_Month.selected()
                endDate = date[month] + 1
            elseif super.rad_Year.selected()
                endDate = date[year] + 1
            endif
            if super.btn_GroupIn.pressed()
                groupName = super.lng_GroupName.getText()
                if groupName.getType() != "String" //Data type check
                    print("Please enter a valid group name")
                    groupName = None
                endif
            endif
            if super.btn_Attendance.pressed()
                if groupName is not None //presence check for a groupName having been entered
                    _attendance()
                else
                    print("Enter a group name first")
                endif
            elseif super.btn_TimeMissed.pressed()
                if groupName is not None
                    _eventCount()
                else
                    print("Enter a group name first")
                endif
            endif
        endprocedure

        private events
        private comits
        private comitsPassA
        private comitsPassB
        private students
        private teachers
        private curuserid
        private anadata

        private procedure _getdata()
            events = openRead("tbl_events")
            if super.rad_Student.selected()
                students = openRead("tbl_students")
                for i = 0 to length(students) //finds userid from username
                    if students[i][1] = groupName
                        curuserid = students[i][0]
                    else
                        next i
                    endif
                endfor
                if curuserid is None
                    print("Invalid user")
                comits = openRead("tbl_comStudents")
                for i = 0 to length(comits)//selects comits relevant to group
                    if comits[i][1] = curuserid
                        comitsPassA.append(comits[i])
                        next i
                    else
                        next i
                    endif
                endfor

                if super.rad_All.selected()
                    eventType = None
                elseif super.rad_Internal.selected()
                    eventType = "Internal"
                elseif super.rad_External.selected()
                    eventType = "External"
                elseif super.rad_Detention.selected()
                    eventType = "Detention"
                endif

                if eventType is not None
                    for i = 0 to length(comitsPassA)
                        for j = 0 to length(events)
                            if events[j][0] = comitsPassA[i][2]
                                if events[j][2] = eventType //where third column is the eventType of the event
                                    comitsPassB.append(events[j],comitsPassA[i])
                                    next j
                                else
                                    next j
                                endif
                            else
                                next j
                            endif
                        endfor
                        next i
                    endfor
                endif

                for i = 0 to length(comitsPassB)
                    if comitsPassB[i][0][3] = date
                        anadata.append(comitsPassB[i])
                        next i
                    else
                        next i
                    endif
                endfor

                comits = openRead("tbl_comStudents")
                **RETURNS ALL DATA ON EVENTS THAT MEMBERS OF THE GROUP ARE DUE TO BE ATTENDING STORED IN 'ANADATA'**
            elseif super.rad_Year.selected()
                students = openRead("tbl_students")
                comits = openRead("tbl_comStudents")
                **RETURNS ALL DATA ON EVENTS THAT MEMBERS OF THE GROUP ARE DUE TO BE ATTENDING STORED IN 'ANADATA'**
            elseif super.rad_Teacher.selected()
                teachers = openRead("tbl_teachers")
                comits = openRead("tbl_comTeachers")
                **RETURNS ALL DATA ON EVENTS THAT MEMBERS OF THE GROUP ARE DUE TO BE ATTENDING STORED IN 'ANADATA'**
            endif
            return anadata

            **anadata construction:  [[event data],[comitsentdata]],[event data] [comitsentdata],.....]**


        endprocedure

        private array data
        private attendcount
        private misscount
        private procedure _attendance()
        private iecount
        private eecount
        private dcount
        private temp_Item
            attendcount = 0
            misscount = 0
            data = _getdata()
            for i = 0 to length(data)
                if data[i][1][3] = 0
                    misscount = misscount + 1
                elseif data[i][1][3] = 1
                    attendcount = attendcount + 1
                endif
                next i
            endfor
            super.tbl_DataOut.setData([groupName,GroupType,attendcount,misscount])
            super.cht_ChartDataOut.setData(attendcount:misscount)

        endprocedure

        private procedure _eventCount()
            data = _getdata()
            for i = 0 to length(data)
                temp_Item = data[i][0][2]
                if temp_Item = "External"
                    eecount = eecount + 1
                elseif temp_Item = "Internal"
                    iecount = iecount + 1
                elseif temp_Item = "Detention"
                    dcount = dcount + 1
                endif
                next i
            endfor
            super.tbl_DataOut.setData([groupName,groupType,eecount,iecount,dcount)
            super.cht_ChartDataOut.setData(eecount:iecount:dcount)
        endprocedure

    endclass
HALT
```

2.5.7.1 EXAMPLES OF CALCULATIONS

2.5.7.1.1 ATTENDANCE CALCULATOR BETWEEN BOUNDED DATA

```
  private array data
  private attendcount
  private misscount
  private procedure _attendance()
  private iecount
  private eecount
  private dcount
  private temp_Item
      attendcount = 0
      misscount = 0
      data = _getdata()
      for i = 0 to length(data)
          if data[i][1][3] = 0
              misscount = misscount + 1
          elseif data[i][1][3] = 1
              attendcount = attendcount + 1
          endif
          next i
      endfor
      super.tbl_DataOut.setData([groupName,GroupType,attendcount,misscount])
      super.cht_ChartDataOut.setData(attendcount:misscount)

  endprocedure
```

2.5.7.1.2 EVENT TYPE COUNTER

```
private procedure _eventCount()
    data = _getdata()
    for i = 0 to length(data)
        temp_Item = data[i][0][2]
        if temp_Item = "External"
            eecount = eecount + 1
        elseif temp_Item = "Internal"
            iecount = iecount + 1
        elseif temp_Item = "Detention"
            dcount = dcount + 1
        endif
        next i
    endclass
    super.tbl_DataOut.setData(groupName,groupType,eecount,iecount,dcount)
    super.cht_ChartDataOut.setData(eecount:iecount:dcount)
endprocedure
```

2.5.8 MODIFY USER OBJECT

2.5.8.1 SPLASH SCREEN

```
START Modify user splash screen **Uses predefined User Object, defined by User() class and created in Login() class**

    class Modify_Users_Splash_Screen_Gui
        new btn_adminModify
        new btn_changePassword
        new btn_return
        public procedure new()
            instance.constructGui()
            instance.display()
        endprocedure
    endclass


    class Modify_Users_Splash_Screen_Gui inherits Modify_Users_Splash_Screen_Gui

        public procedure new()
            super.new()
            if super.btn_changePassword.pressed()
                **STARTS THE Change password screen**
            elseif super.btn_adminModify.pressed()
                if currentuser.accesslevel = "Teaacher"
                    **Starts the modify user (admin) screen**
                else
                    print("You must be a teacher to access this ")
                endif
            elseif super.btn_return.pressed()
                **closes current window snd shows the splash screen**
            endif
        endprocedure



    endclass


HALT
```

2.5.8.2 EDIT PASSWORD

```
START edit password screen **Uses predefined User Object, defined by
User() class and created in Login() class**

      class Edit_Password_Screen_Gui
            new btn_confirm
            new inp_curpassword
            new inp_curpasswordcheck
            new inp_newpassord
            new inp_newpassordcheck
            public procedure new()

                  inp_curpassword.clearText
                  inp_curpasswordcheck.clearText
                  inp_newpassord.clearText
                  inp_newpassordcheck.clearText
                  instance.constructGui()
                  instance.display()
            endprocedure

      endclass



      class Edit_Password_Screen inherits Edit_Password_Screen_Gui


            public procedure new()
                  super.new()
                  if super.btn_confirm.pressed()
                        _main()
                  endif

            endprocedure

            private curPassword
            private curPasswordCheck
            private curHashedPassword
            private newPassword
            private newPasswordCheck
            private users
            private newHashedPassword
            private procedure _main()
                  curPassword = super.inp_curpassword.getText()
                  curPasswordCheck = super.inp_curpasswordcheck.getText()
                  if curPassword = curPasswordCheck
                        newPassword = super.inp_newpassord.getText()
                        newPasswordCheck =
super.inp_newpassordcheck.getText()
                        if newPassword = newPasswordCheck
                              if currentuser.accesslevel = "Student"
                                    users = openReadWrite("tbl_students")
                                    for i = 0 to length(users)
```

```
                                        if users[i][0] =
currentuser.userid
                                              curHashedPassword =
hashfunc.hashed(curPassword)
                                              if users[i][7] =
curHashedPassword
                                                    newHashedPassword =
hashfunc.hashed(newPassword)
                                                    users[i][7] =
newHashedPassword
                                                    users.save()
                                                    users.close
                                              else
                                                    print("Current
Password is not correct")
                                              endif
                                        else
                                              next i
                                        endif
                                  endfor
                            elseif currentuser.accesslevel = "Teacher"
                                  users = openReadWrite("tbl_teachers")

                            endif
                      else
                            print("ERROR New password do not match")
                      endif
                else
                      print("ERROR Current passwords do no match")
                endif


          endprocedure

     endclass

HALT
```

```
START edit password screen **Uses predefined User Object, defined by User() class and created in Login() class**

    class Edit_Password_Screen_Gui
        new btn_confirm
        new inp_curpassword
        new inp_curpasswordcheck
        new inp_newpassord
        new inp_newpassordcheck
        public procedure new()

            inp_curpassword.clearText
            inp_curpasswordcheck.clearText
            inp_newpassord.clearText
            inp_newpassordcheck.clearText
            instance.constructGui()
            instance.display()
        endprocedure

    endclass



    class Edit_Password_Screen inherits Edit_Password_Screen_Gui


        public procedure new()
            super.new()
            if super.btn_confirm.pressed()
                _main()
            endif

        endprocedure

        private curPassword
        private curPasswordCheck
        private curHashedPassword
        private newPassword
        private newPasswordCheck
        private users
        private newHashedPassword
        private procedure _main()
            curPassword = super.inp_curpassword.getText()
            curPasswordCheck = super.inp_curpasswordcheck.getText()
            if curPassword = curPasswordCheck
                newPassword = super.inp_newpassord.getText()
                newPasswordCheck = super.inp_newpassordcheck.getText()
                if newPassword = newPasswordCheck
                    if currentuser.accesslevel = "Student"
                        users = openReadWrite("tbl_students")
                        for i = 0 to length(users)
                            if users[i][0] = currentuser.userid
                                curHashedPassword = hashfunc.hashed(curPassword)
                                if users[i][7] = curHashedPassword
                                    newHashedPassword = hashfunc.hashed(newPassword)
                                    users[i][7] = newHashedPassword
                                    users.save()
                                    users.close
                                else
                                    print("Current Password is not correct")
                                endif
                            else
                                endif
                        endfor
                    elseif currentuser.accesslevel = "Teacher"
                        users = openReadWrite("tbl_teachers")

                    endif
                else
                    print("ERROR New password do not match")
                endif
            else
                print("ERROR Current passwords do no match")
            endif


        endprocedure

    endclass
                        endfor
                    elseif currentuser.accesslevel = "Teacher"
                        users = openReadWrite("tbl_teachers")

                    endif
                else
                    print("ERROR New password do not match")
                endif
            else
                print("ERROR Current passwords do no match")
            endif


        endprocedure

    endclass

HALT
```

2.5.8.3 EDIT USER DATA

```
START Edit User Data (admin)

      class Edit_User_Data_Gui

            new inp_userIn
            new btn_search
            new tbl_dataOut
            new btn_confirm

            public procedure new()

                  inp_userIn.clearText
                  instance.constructGui()
                  instance.display()

            endprocedure



      endclass



      class Edit_User_Data inherits Edit_User_Data_Gui

            public procedure new()
                  super.new()
                  if super.btn_search.pressed()
                        _search()
                  endif
                  if super.btn_confirm.pressed()
                        _main()
                  endif
            endprocedure
            private username
            private user
            private teachers
            private type
            private id
            private students
            private procedure _search()
                  username = super.inp_userIn.getText()
                  teachers = OpenRead("tbl_teachers")
                  students = OpenRead("tbl_students")
                  for i = 0 to length(teachers)
                        if teachers[i][3] = username
                              user = teachers[i]
                              type = "Teacher"
                              id = teachers[i][0]
                        else
                              next i
                        endif
```

```
                endfor
                if user is None
                      for i = 0 to length(students)
                            if students[i][8] = username
                                  user = students[i]
                                  type = "Student"
                            else
                                  next i
                            endif
                      endfor
                if user is None
                      print("User not found")
                else
                      super.tbl_dataOut.setData(user)
                endif

          endprocedure
          private newData
          private procedure _main()
                newData = super.tbl_dataOut.getData()
                if type = "Teacher"
                      for i = 0 to length(teachers)
                            if teachers[i][0] = id
                                  teachers[i] = newData
                                  teachers.save()
                                  break
                            else
                                  next i
                            endif
                      endfor
                elseif type = "Student"
                      for i = 0 to length(students)
                            if students[i][0] = id
                                  students[i] = newData
                                  students.save()
                                  break
                            else
                                  next i
                            endif
                      endfor
                endif
                teachers.close()
                students.close()
          endprocedure

    endclass



HALT
```

```
START Edit User Data (admin)

    class Edit_User_Data_Gui

        new inp_userIn
        new btn_search
        new tbl_dataOut
        new btn_confirm

        public procedure new()

            inp_userIn.clearText
            instance.constructGui()
            instance.display()

        endprocedure


    endclass


    class Edit_User_Data inherits Edit_User_Data_Gui

        public procedure new()
            super.new()
            if super.btn_search.pressed()
                _search()
            endif
            if super.btn_confirm.pressed()
                _main()
            endif
        endprocedure
        private username
        private user
        private teachers
        private type
        private id
        private students
        private procedure _search()
            username = super.inp_userIn.getText()
            teachers = OpenRead("tbl_teachers")
            students = OpenRead("tbl_students")
            for i = 0 to length(teachers)
                if teachers[i][3] = username
                    user = teachers[i]
                    type = "Teacher"
                    id = teachers[i][0]
                else
                    next i
                endif
            endfor
            if user is None
                for i = 0 to length(students)
                    if students[i][8] = username
                        user = students[i]
                        type = "Student"
                    else
                        next i
                    endif
                endfor
            if user is None
                print("User not found")
            else
                super.tbl_dataOut.setData(user)
            endif

        endprocedure
        private newData
        private procedure _main()
            newData = super.tbl_dataOut.getData()
            if type = "Teacher"
                for i = 0 to length(teachers)
                    if teachers[i][0] = id
                        teachers[i] = newData
                        teachers.save()
                        break
                    else
                        next i
                    endif
                endfor
            elseif type = "Student"
                for i = 0 to length(students)
                    if students[i][0] = id
                        students[i] = newData
                        students.save()
                        break
            students.close()
        endprocedure

        endprocedure

    endclass



HALT
```

2.5.9 REGISTRATION SCREEN OBJECT

```
START Registration Screen **Uses predefined User Object, defined by User()
class and created in Login() class**


      class Registration_Screen_Gui

            new tbl_dataOut
            new btn_submit
            public procedure new()
                  instance.constructGui()
                  instance.display()

            endprocedure

      endclass



      class Registration_Screen inherits Registration_Screen_Gui
            private teachers
            private comitTeachers
            private comitStudents
            private students
            private date
            private time
            private events
            private array curComitTeacherPassA
            private array curComitTeacherPassB
            private array curComitTeacherPassC
            public procedure new()
                  super.new()
                  date = getCurrentDate()
                  time = getCurrentTime()
                  events = openRead("tbl_events")
                  for i = 0 to length(comitTeachers)
                        if comitTeachers[i][1] = currentuser.userid
                              curComitTeacherPassA.append(comitTeachers[i])
                              next i
                        else
                              next i
                        endif
                  endfor

                  for i = 0 to length(curComitTeacherPassA)
                        for j = 0 to length(events)
                              if curComitTeacherPassA[i][2] = events[j][0]


      curComitTeacherPassB.append(curComitTeacherPassA[i],events[j])
                                    next j
                              else
                                    next j
```

```
                        endif
                    endfor
                    next i
                endfor

                for i = 0 to length(curComitTeacherPassB)
                    if curComitTeacherPassB[i][1][3] = date
                        if curComitTeacherPassB[i][1][4] is between
time and (time[mins] + curComitTeacherPassB[i][1][6])

    curComitTeacherPassC.append(curComitTeacherPassB[i][0])
                            next i
                        endif
                    else
                        next i
                    endif
                endfor

                super.tbl_dataOut.setData(curComitTeacherPassC)
                if super.btn_submit.pressed()
                    _update()
                endif
            endprocedure

            private register
            private procedure _update()
                comitStudents = openReadWrite("tbl_comitStudent")
                register = super.tbl_dataOut.getData()
                if register = curComitTeacherPassC
                    print("No data changed")
                endif
                for i = 0 to length(register)
                    for j = 0 to length(comitStudents)
                        if register[i][0] = comitStudents[j][0]
                            comitStudents[j] = register[i]
                            next i
                        else
                            next i
                        endif
                    endfor
                    next i
                endfor
                comitStudents.save()
                comitStudents.close()
            endprocedure



    endclass


HALT
```

```
START Registration Screen **Uses predefined User Object, defined by User() class and created in Login() class**


    class Registration_Screen_Gui

        new tbl_dataOut
        new btn_submit
        public procedure new()
            instance.constructGui()
            instance.display()

        endprocedure

    endclass


    class Registration_Screen inherits Registration_Screen_Gui
        private teachers
        private comitTeachers
        private comitStudents
        private students
        private date
        private time
        private events
        private array curComitTeacherPassA
        private array curComitTeacherPassB
        private array curComitTeacherPassC
        public procedure new()
            super.new()
            date = getCurrentDate()
            time = getCurrentTime()
            events = openRead("tbl_events")
            for i = 0 to length(comitTeachers)
                if comitTeachers[i][1] = currentuser.userid
                    curComitTeacherPassA.append(comitTeachers[i])
                    next i
                else
                    next i
                endif
            endfor

            for i = 0 to length(curComitTeacherPassA)
                for j = 0 to length(events)
                    if curComitTeacherPassA[i][2] = events[j][0]
                        curComitTeacherPassB.append(curComitTeacherPassA[i],events[j])
                        next j
                    else
                        next j
                    endif
                endfor
                next i
            endfor

            for i = 0 to length(curComitTeacherPassB)
                if curComitTeacherPassB[i][1][3] = date
                    if curComitTeacherPassB[i][1][4] is between time and (time[mins] + curComitTeacherPassB[i][1][6])
                        curComitTeacherPassC.append(curComitTeacherPassB[i][0])
                        next i
                    endif
                else
                    next i
                endif
            endfor

            super.tbl_dataOut.setData(curComitTeacherPassC)
            if super.btn_submit.pressed()
                _update()
            endif
        endprocedure

        private register
        private procedure _update()
            comitStudents = openReadWrite("tbl_comitStudent")
            register = super.tbl_dataOut.getData()
            if register = curComitTeacherPassC
                print("No data changed")
            endif
            for i = 0 to length(register)
                for j = 0 to length(comitStudents)
                    if register[i][0] = comitStudents[j][0]
                        comitStudents[j] = register[i]
                        next i
                    else
                        next i
                    endif
                endfor
                next i
            comitStudents.close()
        endprocedure


    endclass

HALT
```

2.5.10 VALIDATION EXAMPLES

```
if lowerbound is not None
    lowerbound
else
    print("please initialse first")
endif
```

here it checks to see if a variable has been set externally in the program and prompts the user to do this before continuing

```
username = super.inp_username.getText()
if username is None
    print("Please enter a username")
endif
password = super.inp_password.getText()
if password is None
    print("please enter a password")
endif
```

this is an example of a presence check. In the final code these will be used whenever data is inputted by the user to avoid crashes. It simply checks to see if there has been an input into the program where there should have been.

2.6 KEY VARIABLES AND STRUCTURES

2.6.1 KEY VARIABLES

My program will include many variables many of which are only used a handful of times for specific tasks in the following table are the core variables that will be used throughout the program

Name	Type	Where stored and why needed
Database Cursor	Sqlite3.Cursor (class object)	Needed to interact with my SQLite database containing most of the important data for the program
UserID	Integer (unique)	This variable is created when the user logs in using their username and password. It is then used for various functions to search the database for data related to the current user

Account_Type	Boolean	This is a variable created at login and is used globally through the program to determine if the user is eligible to use of certain functions or screens
Login Screen UI Variable	PyQt UI file	This variable reads a locally stored PyQt UI file and loads it for use in a python program as an object. This variable is specifically for the Login Screen
Splash Screen UI variable	PyQt UI file	This variable reads a locally stored PyQt UI file and loads it for use in a python program as an object. This variable is specifically for the Splash Screen

There will be a PyQt UI file variable for use in constructing each screen they all serve the same function for their different screens

2.6.2 KEY DATA STRUCTURES

Whenever the cursor variable browses the database to access the retrieved data a list has to be created. Therefore, due to the many times for many different functions that the database is accessed there are many lists that are used a few times

Name	Used to Store	Where stored and why needed
Create_Event_Data	Data to fill out the fields in the Event Table in the database	Stored locally before being emptied into the database It is needed to hold data related to the creation of a new event whilst data is still being collected before being moved to the DB
Create_User_Data	Data to fill out the fields in the Users Table in the database	Stored locally before being emptied into the database It is needed to hold data related to the creation of a new user whilst data is still

		being collected before being moved to the DB
Analytics_Data	A list used to store data fetched from the database whilst it is being operated on to find statistics about events	Stored locally before being outputted via different widgets on the screen.

2.7 TESTING

2.7.1 TEST DATA

Whilst testing my program to start I will not be performing destructive testing. Initially I will just be making sure the core functionality of the program works

To do this I will use the following users to test each function of the program.

One is an administrator to allow it to access all functionality, it has a one-character username, something that will later not be allowed for data security reasons and a password of '123' (hashed) to make it quick for me to log in when testing.

I will also have a test user of below admin level to make sure that student users do not have access to any unauthorized areas and no crashes occur for student based functions.

Each user (Test teacher and test student) will get a set of test events which span the 3 event types.

2.7.1.1 TEACHER

User data:

UserID	Forename	Surname	Username	Password (un-hashed)	User_Level
5	Sam	Barrett	w	123	1

Test event data:

Commitments table:

CommitmentID	TeacherID (foreign key)	EventID (foreign key)
3	5	17
4	5	18
5	5	19

2.7.1.2 STUDENT

User data:

UserID	Forename	Other Names	Surname	YearID	FormID	Gender	DoB	Username	Password (un-hashed)	User_Level
7	A	Test	Student	7	1	Male	07/06/1999	e	123	0

Test event data:

Commitments table:

CommitmentID	StudentID (foreign key)	EventID (foreign key)	Attended?
1	7	1	Null
2	7	2	Null
3	7	3	Null

The test users will both me members of the same test events just in different roles therefore the Events Table will be the same for both

Events table:

EventID	RoomID	TypeID	Date – Variable on testing date	Time- variable with testing time	Desc	Duration
1	3	1	2017-02-26	13:40:00	Test Event: Internal Event	60
2	3	2	2017-02-27	13:40:00	Test Event: External Event	60
3	3	3	2017-02-28	13:40:00	Test Event: Detention	60

Later in the process I will perform destructive testing by using a student account and attempting various functions and trying to enter illegal data into all fields in the program to make sure it rejects it and doesn't crash. Such illegal data includes events at inappropriate times or clashing events or double booked rooms. Currently this data will be used for testing other functions that operate on correctly entered data

I will also get different people of differing technical abilities to use the program to make sure all my success criteria are met

2.7.2 TEST PLAN

Upon the completion of each class (screen), I will test that it meets the success criteria related to that screen (see Success Criteria). For the most part I will simply run the program and provide it with acceptable data, to test the functionality of the data, before performing destructive testing with illegal data entry. If the program fails either of the tests I will then have to go back to the code to find the source of the error and fix it. Any fixes I perform will be documented with the issue and fix I used to solve it.

2.7.2.1 TESTING OF SUCCESS CRITERIA

2.7.2.1.1 PROGRAM INITIALISATION

Success Criteria Number(s)	Test	Destructive Test	Required Result
1	Run Program	Move/delete a UI file	Program starts correctly or notifies the user that a UI file cannot be found
2	Run Program	Move/delete the Database file	Program starts correctly or notifies the user that a DB file cannot be found
3	Run Program	Removes needed Library File	Program starts correctly or notifies the user that an external library is missing
4	n/a	Entry of illegal data in every field	Program doesn't crash and tells user that data isn't of the correct format
5	Completed code should feature classes and methods	n/a	GUI forms should be generated using class based objects
6	All screens accessed via related buttons	Buttons pressed in quick succession	Program should open all forms when required. Only one form should be open at one time. Program should not crash upon multiple button presses

2.7.2.1.2 LOGIN SCREEN

7	Test data entered into the form	Illegal data entered into the form	Program should allow entry of both (at this stage)
8	Test data Password entered into the password field	n/a	Password field should not display actual characters, rather dots or asterisks
9/10	Have program print passwords that are being compared for login	n/a	Printed values should not be in plaintext format, should be in a hashed format
11	Log in using student and teacher test data	n/a	Program should login and print usertype to console for verification
12	Log in using student and teacher testdata	n/a	Print out User object attributes after attempted creation
13	n/a	Attempt to login using incorrect Username	Program should return an error message telling the user that the password is username
14	n/a	Attempt to login using incorrect password	Program should return an error message telling the user that the password is incorrect
15	Attempt to login using student and teacher test data	n/a	Program should tell the user that their details are correct and open the splash screen
16	Login using student and teacher test data	n/a	Print out sql queries to be executed and results. Proof of this working can be seen from the successful login tested above

17			Proof of this working can be seen from the successful login tested above
18	Press the cancel button on the form	n/a	Program should exit to desktop

2.7.2.1.3 SLASH SCREEN

19	Login to show the form	n/a	Program should display the form as designed
20,21,22,23	Whist logged in using the test student account, attempt to access screen	n/a	Program should tell the user that they are not able to access this feature
24	Whilst logged in, press the logout button	n/a	Program should close the splash screen and return the user to the login screen, with all fields blank
25	After logging out, print the user attributes	n/a	They should all hold None values
26	Whilst logged in, press the exit button	n/a	Program should exit to the desktop
27,28,29,30	Whilst logged in using the teacher test user, attempt to access the screen	n/a	Program should open the screens and close the splash screen
31,32	Whilst logged in, attempt to access the screen	n/a	Program should open the screen and close the splash screen

2.7.2.1.4 CREATE NEW USER SCREEN

33,34	Fill out all fields with valid data, press the confirm button	Leave some fields blank	the program should run as normal if all fields hold values. If any are empty the

			program should return an error message to the user
35	Fill out all data for student user creation and press the confirm button	n/a	The program should enter all data into the database in a fully normalized format
36	Fill out all data for teacher user creation and press the confirm button		The program should enter all data into the database in a fully normalized format
37	Fill out all fields with relevant data	Fill fields with illegal data	Program should run if all data entered meets the requirements, otherwise it should return an error to the user.
38	By looking at the form you should be able to tell which fields much be filled out for different user creation	n/a	A group of test users should easily be able to tell, respond to feedback if necessary
39,40,41	Create test users	Create test user with incorrect names for forms etc.	If all data can be found in the database a new user with the correct foreign keys should be created. Else the user should be notified that a name doesn't check out

2.7.2.1.5 CREATE NEW EVENT SCREEN

42,43	Create event with all fields filled out	Create event with some empty fields	If all fields are filled the program should continue. Otherwise it should alert the user to any empty fields

44,45,46	Attempt to create an event of the three different types	N/A	Program should take in all input and create a new fully normalized DB entry
47,48	Using valid event data Print values to entered into DB	Using illegal event data. Print all values to be entered into the DB	All values should be primary or foreign keys. Otherwise it should return an error that a something doesn't exist in the DB
49	Press the return button	n/a	Program should close current window and display the splash screen
50	Create event with valid time frame	Create event with invalid time frame	Program should register whether the event being created runs between accepted time frames (break, lunch, after school) depending on event type. Returning an error if the event is outside of these times
51	Create event with free room	Create event with known booked room	Program should return an error if the room is already booked. Otherwise it should continue with the process
52	Create event with free users	Create an event with known double booked users	If there is a double booked user: Program should assess level of importance of prior engagement and overwrite if possible. Otherwise it should alert the user to the error

			Otherwise it should add the user to the event

2.7.2.1.6 SEE PERSONAL EVENTS SCREEN

53	Use the calendar widget to enter a date. Program prints it	n/a	Program should print the date in a valid format
54	Use the radio buttons to select a time frame	n/a	Print the name of which radio button is selected
55	Press the next or previous buttons once events have been initialized	n/a	Program should display different events depending on the current time range selected
56	Use radio button to select event type	n/a	Program should only allow one button to be selected at once. Program should then only display events of the selected type
57	Click the return button	n/a	Program closes current window and shows the splash screen
58	Initialize data	n/a	Program should not show any foreign keys, instead displaying their related secondary keys
59	Initialize data	n/a	All events shown should be relevant to the user. Can be checked against the commitmentstudent DB table

2.7.2.1.7 ADVANCED ANALYTICS SCREEN

60	Analyse data specifying different group types	n/a	Program should output data relevant to each group
61,62,63	Enter group name	Enter invalid group name	Program should show an error message if invalid group name is inputted. Otherwise it should search the corresponding database table for the group, returning the resulting data to the program
64,65,66	Instruct the program to show attendance data	n/a	Program should go through all fetched data and find the attendance ratio of the data set. It should then return these results with the data set to the table and pie chart

2.7.2.1.8 MODIFY USER SCREEN

67	Attempt to change password as a student user	n/a	Program should check confirmation passwords to match and current password to be correct before entering it into the DB
68	Attempt to change password das a teacher user	n/a	Program should check confirmation passwords to match and current password to be correct before entering it into the DB
69,70	Search for user	Search for non existent data	program should search for user and

			display all data, with all foreign keys converted, in the table. If the user cannot be found it should alert the user
71	Change values and hit confirm button	No changes made	Program should return an error if no changes are made. Otherwise it should change the database entry with new *fully normalized* values

2.7.2.1.9 REGISTRATION SCREEN

72,73,74	Open the window	n/a	Program should show the details of all enrolled users. Should only display attendees of current event
75	Press confirm button after completing register	n/a	Program should check to see if the register is complete before entering it into the DB. Return an error if students haven't been registered

3 DEVELOPMENT

3.1 DATABASE CREATION AND POPULATION

3.1.1 PROJECT DECOMOSITION

The database will need to store all data required throughout the program. This will include sensitive user data and critical system data.

It will need to be fully normalized to avoid any possible data redundancy. As when interacting with the database any errors stemming from redundancy can cause catastrophic errors in the program and lead to mass data loss which in a critical system such as this is not optimal

3.1.1.1 DATA REQUIRED

Data will be stored in various tables, below I have outlines the various pieces of data that will be required to be stored for the program to function. These may be stored in different tables and linked to using foreign keys. The actual database layout can be found in the ERD

3.1.1.1.1 STUDENT DATA

The students table will require various key pieces of data used throughout the program for various processes from logging in to event creation

It will require the following to be stored:

- Student ID (int)
- Forename (str)
- Surnames (str)
- Other names (str)
- Password (hashed) (str)
- Gender (str)
- Year group (int)
- Form group (str)
- Date of Birth (str)
- Username (str)
- Access level (int)

3.1.1.1.2 TEACHER DATA

The teachers table will store similar data to that held on the students but is used to separate the two user types for easier use of the DB within the program

The data held on the teachers will be:

- Teacher ID (int)
- Forename (str)
- Surname (str)
- Username (str)
- Password (hashed) (str)
- Access level (int)

3.1.1.1.3 EVENTS DATA

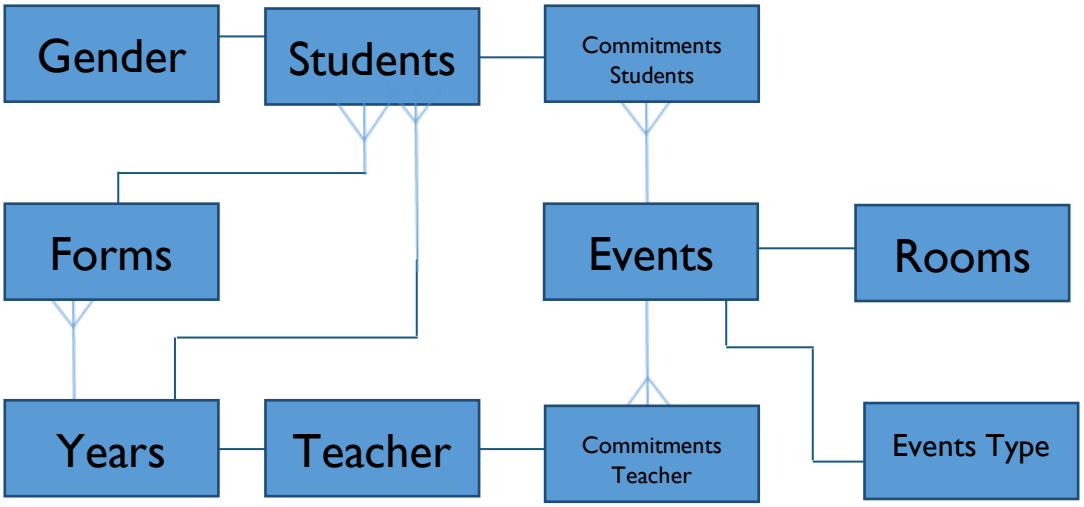
Each event that is created will have a lot of associated data that forms its details

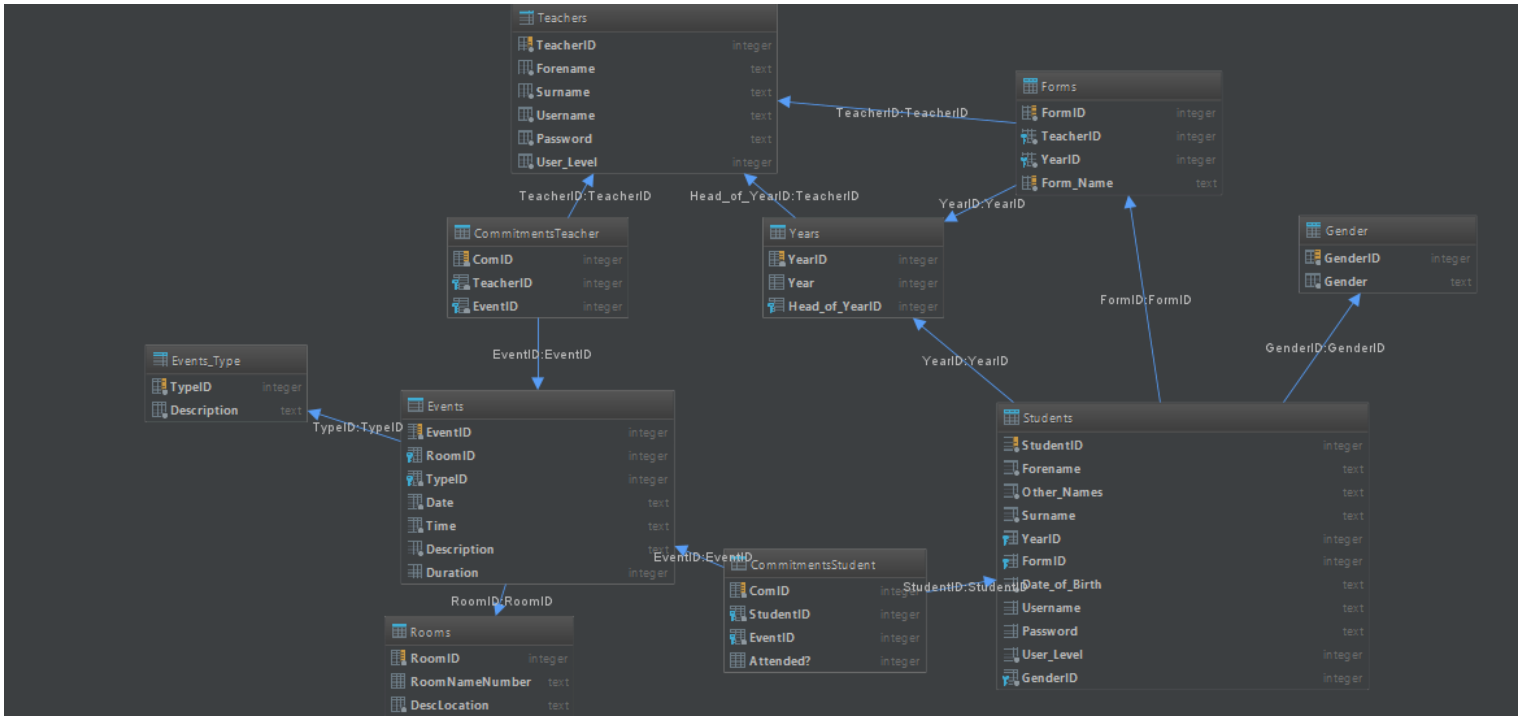
Much of this information will be linked to via foreign keys as it will be data already stored elsewhere in the database

The data held on events will be:

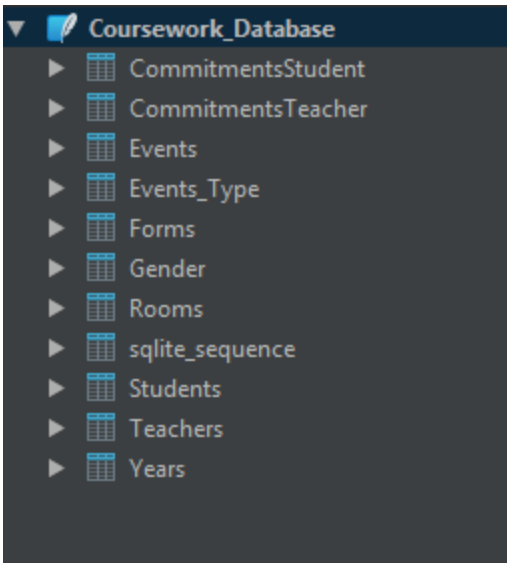
- Event ID (int)
- IDs of both teachers and students attending (int)
- Room where it is being held (if applicable) (str)
- Type of Event (str)
- Date of occurrence (str)
- Time to start (str)
- Duration (int)
- Description (str)

3.1.1.2 ENTITY RELATIONSHIP DIAGRAM

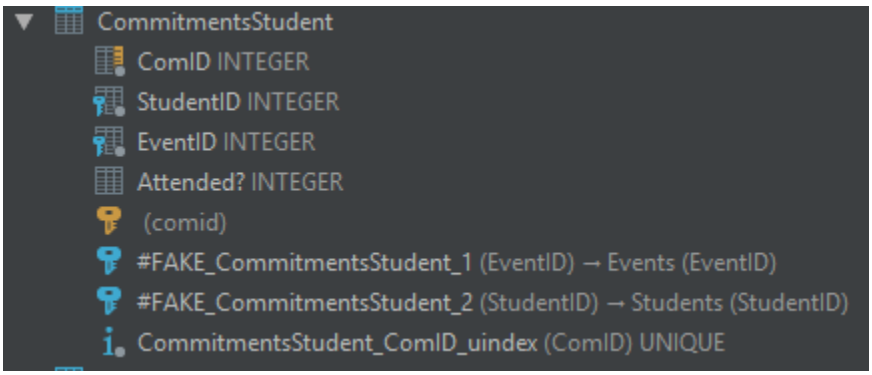




3.1.1.3 TABLE LAYOUT IN DBMS

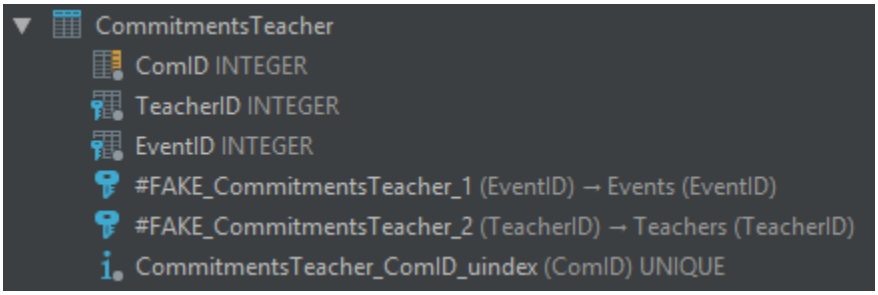


3.1.1.3.1 COMMITENTSSTUDENT



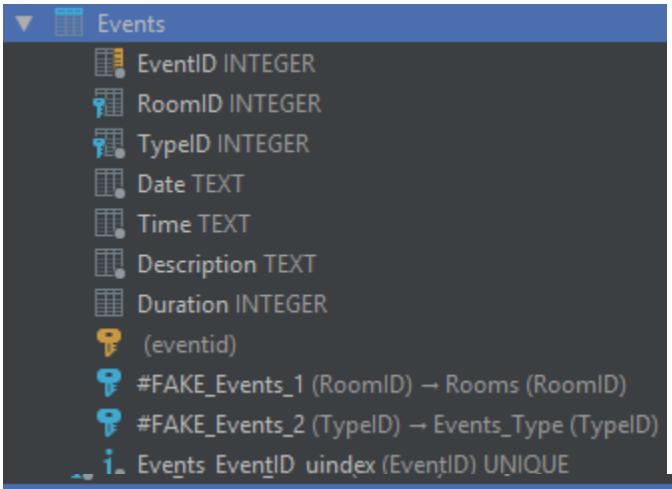
this table is a go-between for the Students and Events tables. It allows the database to be fully normalized by holding event instances for each student registered to attend an event. It has foreign keys pointing to the Students Table (studentID) and to the Events table (EventID)

3.1.1.3.2 COMMITMENTSTEACHER



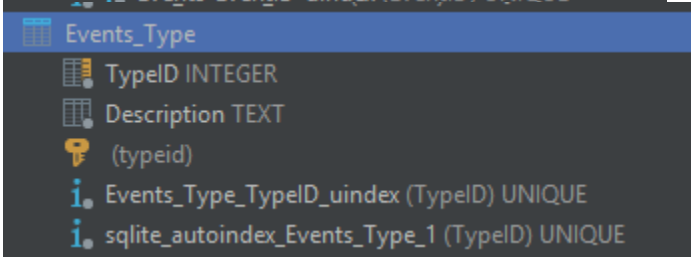
this table performs the same action as the last and is a go between for the Teachers and Events table with foreign keys from Teachers (TeacherID) and Events table (EventID)

3.1.1.3.3 EVENTS



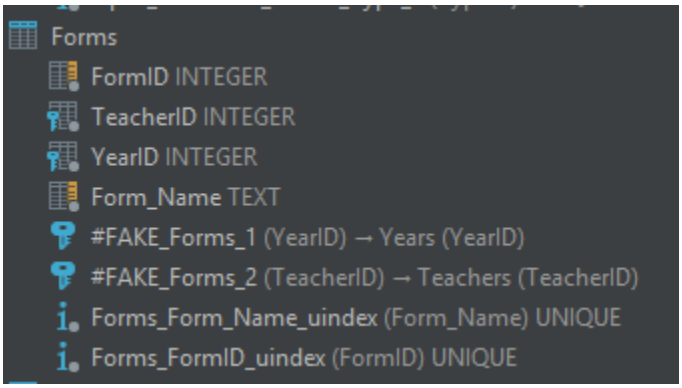
this table holds all the key data for every event created and managed within the program. It has foreign keys pointing to the Rooms (RoomID) and Events_Type (TypeID) Tables. This holds the most important data in the database

3.1.1.3.4 EVENT_TYPE



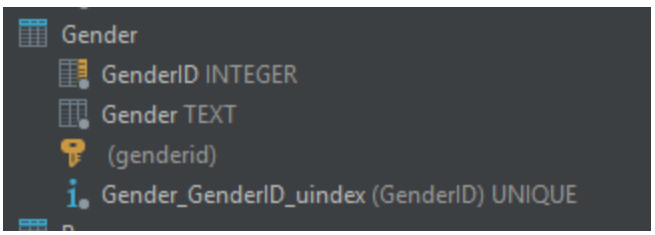
this is a small table to aid in the normalization of the database. It holds TypeIDs which are referenced elsewhere and the corresponding names of those event types along with descriptions of the different types

3.1.1.3.5 FORMS



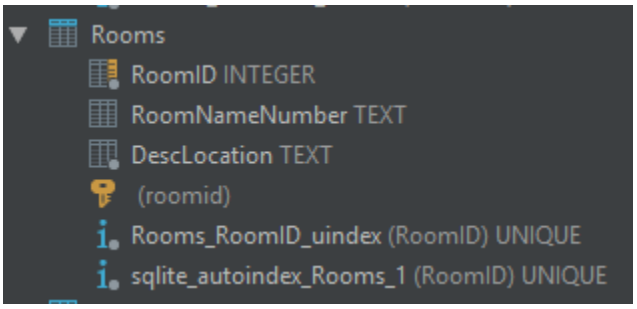
this table holds the data on each form group. It is referenced in each student entry. It has foreign keys to the teachers (TeacherID) and Years (YearID) tables.

3.1.1.3.6 GENDER



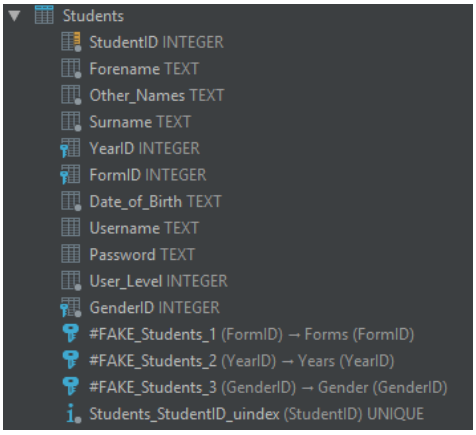
this is another table used to fully normalize the database. It is referenced by every student entry and gives each gender type a unique identifier to be referenced by

3.1.1.3.7 ROOMS



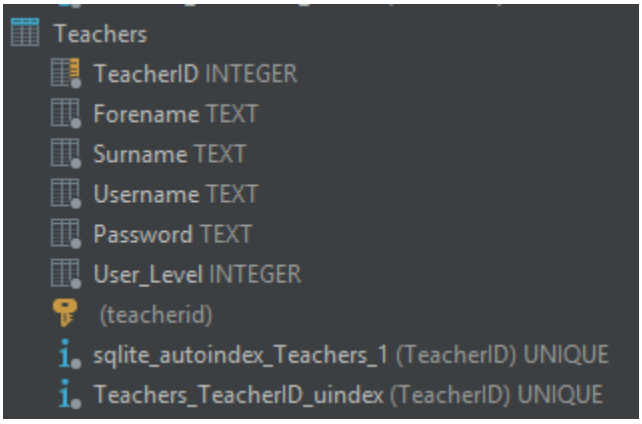
this table is referenced by every internal event or detention as it gives a unique identifier to each room in the school along with a description of its location.

3.1.1.3.8 STUDENTS



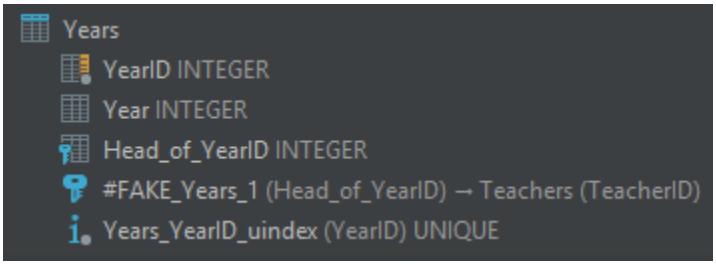
this table stores all the data related to the registered student accounts in the program. From their name to their date of birth. This data is used throughout the program and entities in this table are referenced in the CommitmentsStudent table upon event creation. It has foreign keys pointing to the Forms, Years and Gender tables

3.1.1.3.9 TEACHERS



this table stores all the data related to teacher users within the program from their name to their password. This table is referenced in the CommitmentsTeacher Table upon event creation.

3.1.1.3.10 YEARS



this table stores data relating to the year groups in the school and is referenced in the Forms table and each Student Table. It has foreign keys pointing to the Teachers table to reference the Head of Year for each entity

3.2 USER OBJECT

3.2.1 PROJECT DECOMOSITION

3.2.1.1 CLASS REQUIREMENTS

This is a small yet important class. This is used to create an object upon login possessing all attributes the user needs to navigate the program. It saves unnecessary searching of the DB for the same information related to current UserID etc.

3.2.2 DEVELOPMENT

To fulfill the requirements of this class the code had to be accessible from everywhere in the program and hold three key pieces of data. These must be able to be defined at login and cleared upon logout

Therefore, I simply created a class with an initialization method which defined three attributes originally holding Nonetype values

```
class User:
    def __init__(self):
        self.userid = None
        self.usertype = None
        self.acctype = None
```

3.2.3 ITERATIVE TESTING

This object doesn't require much testing as its purpose it to hold values to be accessed throughout the program. Its functionality can be tested by isolating the class, creating an instance, setting attribute values and having it print them back.

```
class User:
    def __init__(self):
        self.userid = None
        self.usertype = None
        self.acctype = None


def test():
    t_user = User()
    t_user.userid = "TEST USERID"
    t_user.usertype = "TEST USERTYPE"
    t_user.acctype = "TEST ACCTYPE"
    print(type(t_user), "\n", t_user.userid, t_user.usertype, t_user.acctype)

test()
```

This testing code produces the following result:

```
C:\WinPy\python-3.4.3.amd64\python.exe "H:/A-Level Coursework/current_git_code_rep/CourseworkGIT/documentation_code/tdir/user_t.py"
<class '__main__.User'>
 TEST USERID TEST USERTYPE TEST ACCTYPE

Process finished with exit code 0
```

This result shows that the object can be constructed and its attributes changed and used in further functions

3.2.4 FINAL CODE

```
class User:
    def __init__(self):
        self.userid = None
        self.usertype = None
        self.acctype = None


currentuser = User

# Creates Object called
# current user currently
# attributes are set as
# 'None' but are then
# Chanegd to values found
# in the DB

currentuser.userid = 1
currentuser.usertype = "Student"
currentuser.acctype = 2
```

This small section of code shows the definition of the 'User' class and an example of its construction and setup

This section sets up the objects attributes, by default they are set to None

The object is then created with these predefined Nonetype attributes

The object is then created with these predefined Nonetype attributes

The Nonetype attributes are then overwritten to be valid data types

3.3 LOGIN SCREEN

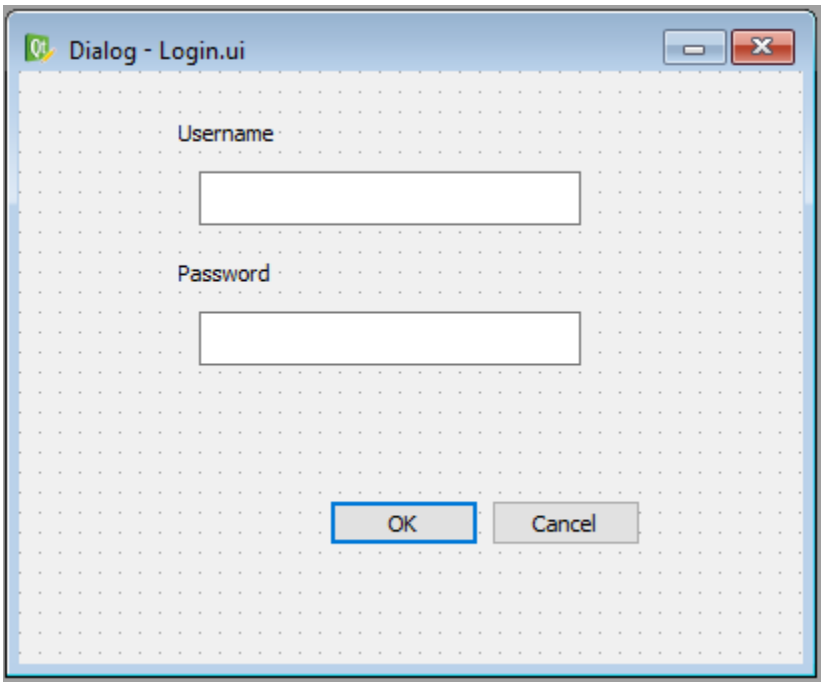
3.3.1 PROJECT DECOMOSITION

3.3.1.1 CLASS REQUIREMENTS

This screen will be the first one encountered by the user upon running the program. It will need to be able to take input of the user's username and password (un hashed) and check them against the username and password (hashed) stored in the database. It will need to do this securely so as not to allow any sensitive user data to be displayed. It must also create the user object upon successful login and launch the splash screen window

3.3.2 DEVELOPMENT

To fulfill the requirements of this class I first designed the User interface in QtDesigner



- I used labels to show what data should be entered into which field
- I used QLineEdits for username and password input with the password field being set up to show dots in place of entered characters
- I used a button box to allow the user to enter the details or exit out of the program

I firstly set up the user interface before taking input from the GUI, making sure not to globalize any sensitive data.

```
class LoginScreen(QtGui.QWidget, Ui_Dialog):
    def __init__(self, parent=None):
        QtGui.QWidget.__init__(self,parent)
        self.setupUi(self)
        self.buttonBox.rejected.connect(lambda: exit(0))
        self.buttonBox.accepted.connect(self.checkpassword)


        def checkpassword():
            username = self.User_Field.text()
            if username == "":
                QtGui.QMessageBox.information(self, "ERROR", "No username entered")
            password = self.Pass_Field.text()
```

After having completed this section I needed to check the password against that stored in the database. Since the database has a hashed password stored I needed to hash the entered password into the same format to compare them. The function I used to hash the entered password I wrote in a separate file for miscellaneous functions (See _MiscFunctions_ section)

I also had to select the data from the database, if any, that corresponded to the entered username. The username could be that of a student or teacher so I must search both for whichever gives me a value.

```
def checkpassword(self):
    username = self.User_Field.text() # TAKES INPUT OF Username AND Password
    if username == "":
        QtGui.QMessageBox.information(self, "ERROR", "No username entered")
    password = self.Pass_Field.text()


    cmd = 'SELECT TeacherID,User_Level,Password FROM Teachers WHERE Username = ?' # CHECKS Teachers TABLE FOR CORRESPONDING USER
    cur.execute(cmd, (username, ))
    data = cur.fetchall()

    if not data: #IF NO TEACHER FOUND, LOOK FOR STUDENT
        cmd = 'SELECT StudentID,User_Level,Password FROM Students WHERE Username = ?'
        cur.execute(cmd, (username, ))
        data = cur.fetchall()
        if not data:
            QtGui.QMessageBox.information(self, "ERROR", "No such User")
        else:
            data.append("Student")
    else:
        data.append("Teacher")

    if __miscfuncitons__.verify_hash(self, password, data[0][2]):
        self.Output_Label.setText("Logging in.....")
```

I have added validation in the form of presence checks which output to the user via QMessageboxes with corresponding error messages

Once I have ascertained whether the password is correct I need to either launch the Start Screen or return an error message

```
        if __miscfuncitons__.verify_hash(self, password, data[0][2]):
            self.Output_Label.setText("Logging in.....")
            user_id = data[0][0]
            account_type = data[0][1]
            usertype = data[1]
            currentuser.userid = user_id
            currentuser.acctype = account_type
            currentuser.usertype = usertype
            self.start()

        else:
            QtGui.QMessageBox.information(self,"Password ERROR","Password is incorrect")
            pass


    def start(self):
        if self.StartWindow is None:
            self.StartWindow = StartScreen()

        self.StartWindow.show()
        self.hide()
```

This completes the code for the login screen

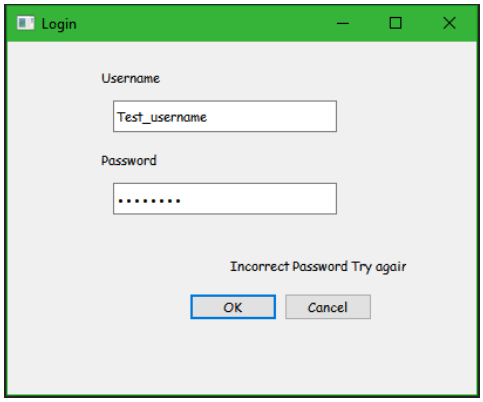
3.3.3 ITERATIVE TESTING

First I tested that this program can take in the data needed for logging in upon a button press. The code responsible for this is:

```
self.buttonBox.accepted.connect(self.checkpassword)
```

```
def checkpassword(self):
    username = self.User_Field.text()
    if username == "":
        QtGui.QMessageBox.information(self, "ERROR", "No username inputted")
    print(username)
    Password = self.Pass_Field.text()
    print(Password)
```

This code and the following input produces this result:




```
C:\WinPy\python-3.4.3.amd64\python.exe "H:/A-Le
Test_username
password
```

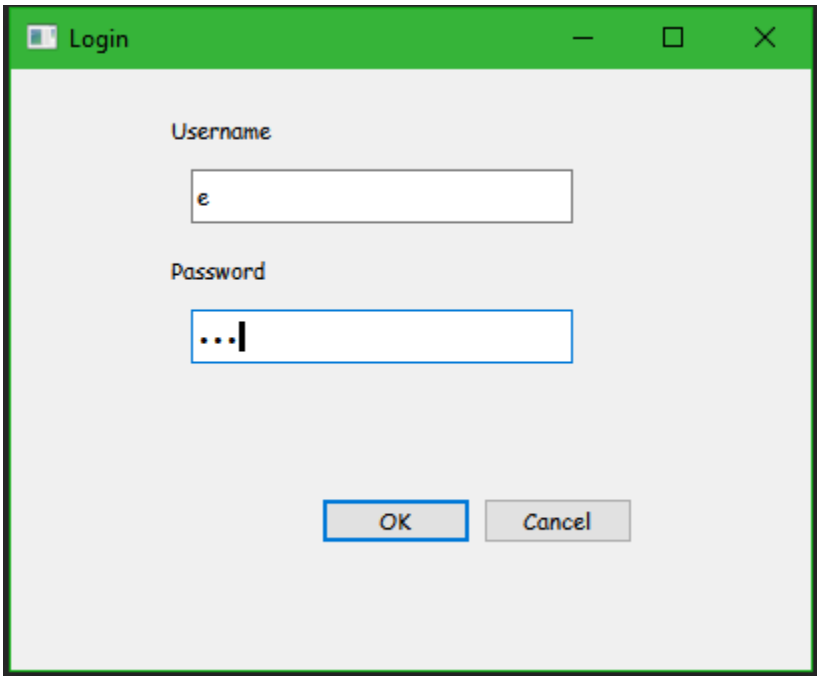
Therefore, I can conclude that this function works and the program can take input from the GUI text boxes in a useable format

The next function of this screen is to properly hash the password and check it against the hashed password stored in the DB. It must be able to do this for student and teacher users.

```
cmd = 'SELECT TeacherID,User_Level,Password FROM Teachers WHERE username =?'
cur.execute(cmd, (username,))
data = cur.fetchall()
print("related teacher data:", data)

if not data:
    cmd = 'SELECT StudentID, User_Level,Password FROM Students WHERE username =?'
    cur.execute(cmd, (username,))
    data = cur.fetchall()
    print("related student data", data)
    data.append("Student")
    if not data:
        self.Output_Label.setText("Incorrect username Try again")
        QtGui.QMessageBox.information(self, "ERROR", "Incorrect username please try again")
        pass
else:
    data.append("Teacher")
if __miscfuncitons__.verify_hash(self, Password, data[0][2]):
    print("PASSWORD CORRECT")
```

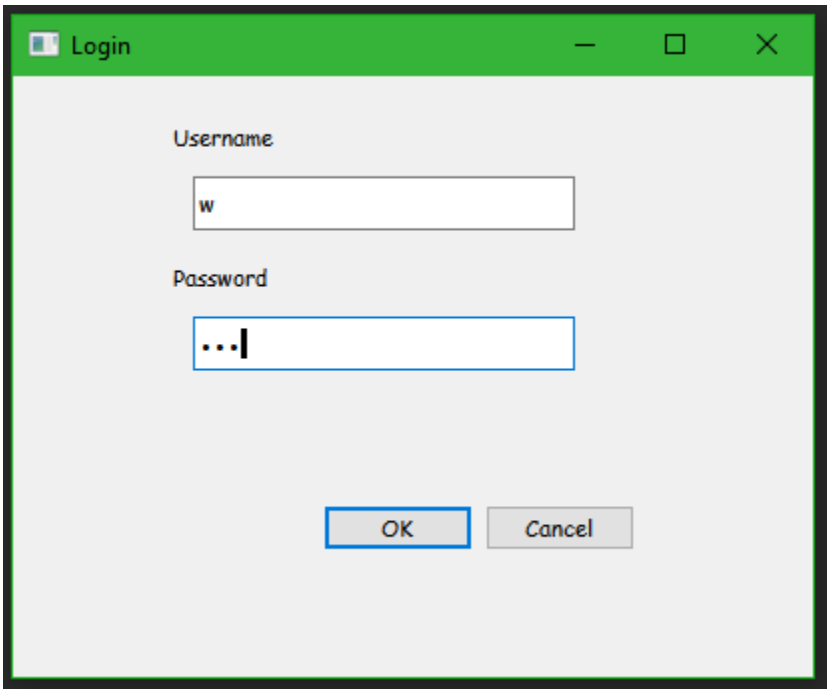
The output of this code for the Student test data is:




```
related teacher data: []
related student data [(7, 0, 'cfbaa60a71942fcdc9d08ffdd3a809dc8daf6f89427049c372260c7df955ab35:9997c6200a3e4ca4bba10c263fcdef89
cfbaa60a71942fcdc9d08ffdd3a809dc8daf6f89427049c372260c7df955ab35 dd 9997c6200a3e4ca4bba10c263fcdef89 ""dddddd
PASSWORD CORRECT
7 Student 0
```

This shows that the code functions as intended for student users with it finding the corresponding data in the database and checking the passwords using my verifyhash() function

The output of this code for the teacher test data is:




```
related teacher data: [(5, 1, '2fcaf63a759a18b2d617cbaa1650252ec01fa845428df2658b2a429bdeaade29:8a039676fffa4d219842401419919db0')]
2fcaf63a759a18b2d617cbaa1650252ec01fa845428df2658b2a429bdeaade29 dd 8a039676fffa4d219842401419919db0 ""dddddd
PASSWORD CORRECT
5 Teacher 1
```

As you can see from the console result the program correctly hashes and checks the password against that stored in the database

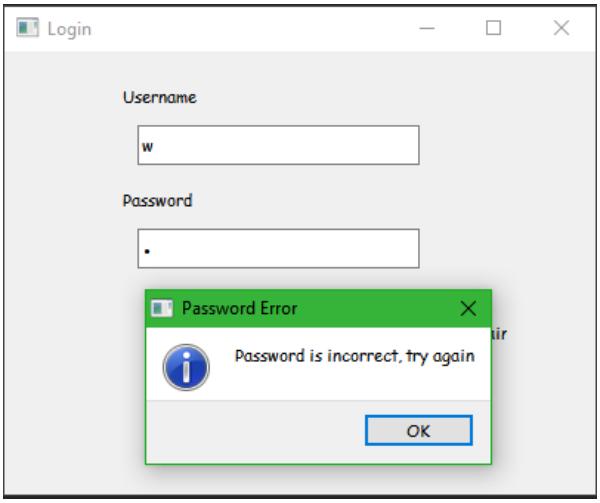
Using the following destructive test data:

1) Username: w

   Password: 3

Should return incorrect password error

Result:

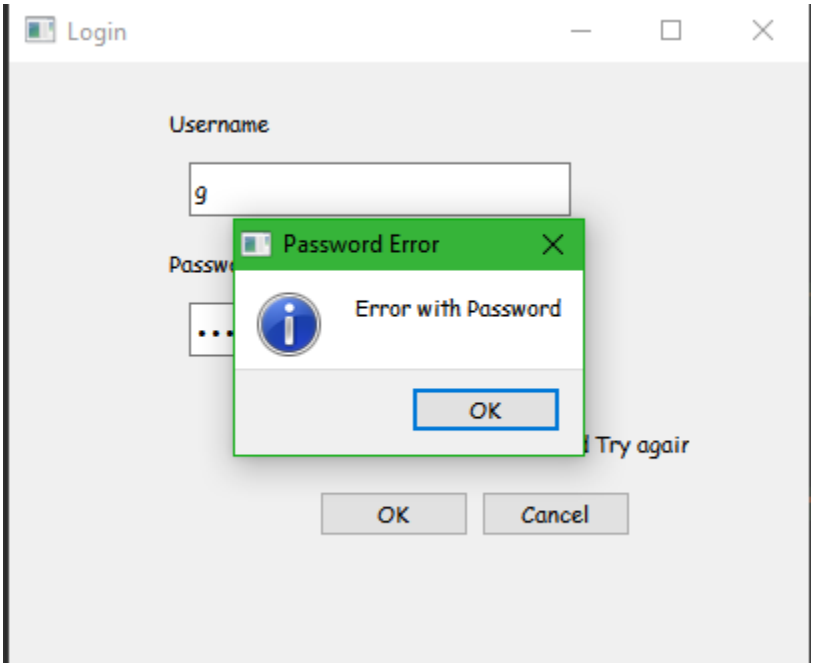


It correctly reponds that the password is incorrect after checking the stored password against the entered one

2) Username: g
   Password: 123

Show return user non-existent error

Result:



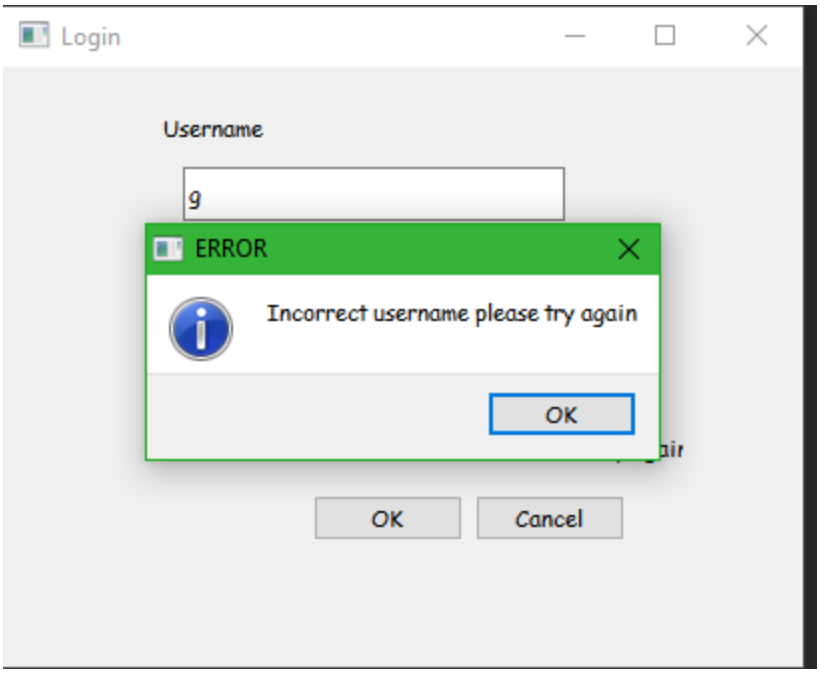

```
related teacher data: []
related student data []
```

This highlights an error in my program as it shows an error with the password instead of telling me that the user does not exist

Upon further inspection, I found that a user was classed as nonexistent if a list containing details of the searched for user was empty. However, I was adding a value to this list before checking its state therefore causing it to never be empty when checked. To fix this I added an else clause and added the user type data to it within there.

```
data.append("Student")
if not data:
    self.Output_Label.setText("Incorrect username Try again")
    QtGui.QMessageBox.information(self, "ERROR", "Incorrect username please try again")
    pass
```

```
if not data:
    self.Output_Label.setText("Incorrect username Try again")
    QtGui.QMessageBox.information(self, "ERROR", "Incorrect username please try again")
else:
    data.append("Student")
```



Now that the error is resolved the program produces the expected result. Alerting the user to the fact that a user with the username they tried does not exist.

3.3.4 FINAL CODE

Within this class there are several functions and procedures each doing smaller subtasks

3.3.4.1.1 __INIT__()

This is a standardized format for the initialization of the class. Within this procedure the class attributes are setup as either null or empty lists. It also listens for button presses which then launch the other functions within the class

```
def __init__(self, parent=None):

    QtGui.QWidget.__init__(self, parent)
    self.upcorrect = True
    self.setupUi(self)
    self.buttonBox.rejected.connect(lambda: exit(0))
    self.buttonBox.accepted.connect(self.start)
    self.StartWindow = None
    self.QueryWindow = None
    self.userpass = None
    self.storedpass = None
    self.password = None
    self.salt = None
```

declaration of attributes

listening for button presses

functions run on button press

3.3.4.1.2 CHECKPASSWORD()

This function is the main function in this class. It links to the other methods to fetch, hash and check the user entered password and username against those stored in the database. It does this using a SQLite DB connection and an external library hashing function. The usage of this hashing function is found in my _miscfunctions_ file which has any reusable, non-class specific function in the project.

in this class the username and password are fetched from the PyQt field input boxes. It also validates to show

```
def checkpassword(self):
    username = self.User_Field.text()
    if username == "":
        QtGui.QMessageBox.information(self, "ERROR", "No username inputted")

    Password = self.Pass_Field.text()

    cmd = 'SELECT TeacherID,User_Level,Password FROM Teachers WHERE username =?'
    cur.execute(cmd, (username,))
    data = cur.fetchall()

    if not data:
        cmd = 'SELECT StudentID, User_Level,Password FROM Students WHERE username =?'
        cur.execute(cmd, (username,))
        data = cur.fetchall()
        data.append("Student")
        if not data:
            self.Output_Label.setText("Incorrect username Try again")
            QtGui.QMessageBox.information(self, "ERROR", "Incorrect username please try again")
            pass
    else:
        data.append("Teacher")

    if __miscfunciton__.verify_hash(Password, data[0][2]):

        self.Output_Label.setText("Logging in.....")
        user_id = data[0][0]
        account_type = data[0][1]
        currentuser.userid = user_id
        currentuser.acctype = account_type
        currentuser.usertype = data[1]
        print(currentuser.userid, currentuser.usertype, currentuser.acctype)
        self.start()
    else:
        self.Output_Label.setText("Incorrect Password Try again")
        QtGui.QMessageBox.information(self, "Password Error", "Password is incorrect, try again")
        pass
```

it then selects the user's data from the database using try and except to prevent crashes when trying to determine user type (student or teacher)

then using my external function, it hashes the entered password in the same format as the stored one and compares them. If they are the same, it logs the user by sending the program to the start() method in if not it rejects the password and throws up a warning message.

See Miscellaneous functions for details on verify_hash()

3.3.4.1.3 START()

This is the final method to run in this class it is essentially the constructor for the Start Screen. It creates the GUI, shows it and hides itself.

```
def start(self):
    if self.StartWindow is None:
        self.StartWindow = StartScreen()

    self.StartWindow.show()

    LoginWindow.hide()
```

it simply creates an instance of the object after checking to see if it already exists before showing it and hiding itself.

3.4 START SCREEN

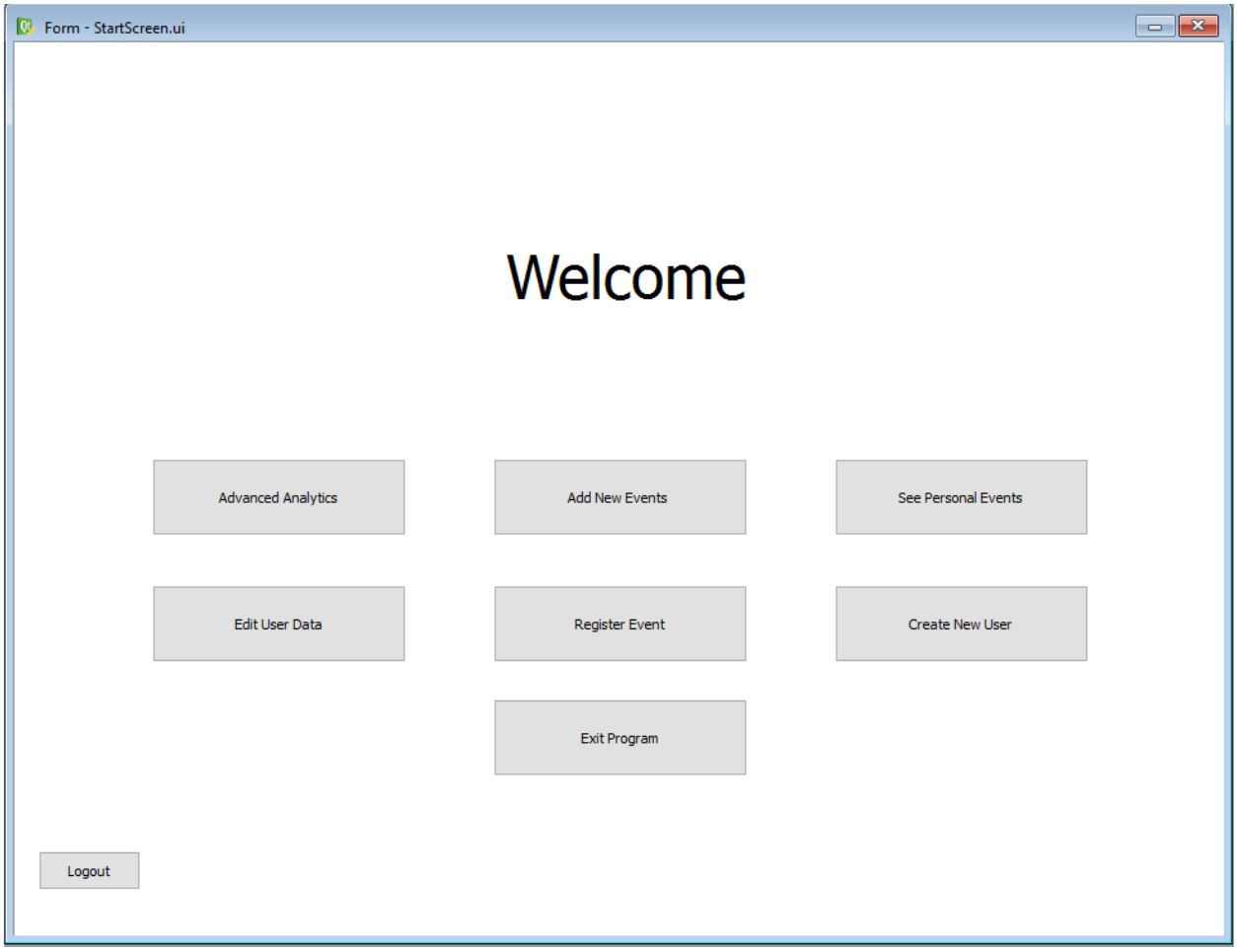
3.4.1 PROJECT DECOMOSITION

3.4.1.1 CLASS REQUIREMENTS

This object will be the launch pad for all the other functions of the program. It will provide clearly labeled buttons directing the user to the different screens. It will need to construct various objects depending on what button is pressed.

3.4.2 DEVELOPMENT

I started development by first designing the GUI in QtDesigner:



I used several QPushButtons to allow the user to navigate to each section of the program. I made the buttons and text large to aid usability. I also used a Qlabel to display the welcome message

The first code I wrote was to setup the UI from the imported .ui file and establish the listeners for the buttons

```
class StartScreen(QtGui.QMainWindow, SWindow):
    def __init__(self, parent = None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setupUi(self)

        self.AEButton.clicked.connect(self.create_event_constructor)
        self.CNUButton.clicked.connect(self.create_new_user_contructor)
        self.EUButton.clicked.connect(self.edit_user_constructor)
        self.ExitButton.clicked.connect(lambda: exit(0))
        self.PEButton.clicked.connect(self.personal_events_contructor)
        self.LogoutButton.clicked.connect(lambda: __miscfuncitons__.logout(self, LoginWindow, currentuser))
        self.AAButton.clicked.connect(self.advanced_analytics_contructor)
        self.REButton.clicked.connect(self.registration_constructor)
```

For the logout and exit buttons I used the lambda function within Python to allow me to run functions and pass parameters.

The exit button runs and exit, code 0, upon press

The logout button uses a _miscfuction_ to clear the variables and return to the login screen, it takes various parameters including the currentuser object

I went on to write the code for all the constructor methods that need to be attached to these buttons.

Going from left to right top row to bottom:

I started by creating the advanced analytics constructor

For each new constructor I created an attribute (initialized as Nonetype)

```
self.AdvancedAnalyticsWindow = None
```

Each constructor has a method which are all in essence the same but creating different objects

```
def advanced_analytics_contructor(self):
    if currentuser.usertype != "Teacher":
        self.Output_Label.setText("Sorry you cannot access this as you are not a Teacher")
    else:
        self.AdvancedAnalyticsWindow = AdvancedAnalyticsScreen()
        self.AdvancedAnalyticsWindow.show()
        self.hide()
```

Some constructors, including this one, check the status of the user to make sure they are a Teacher as some functions are not supposed to be accessible by students

I went on to create a constructor to attach to each remaining button.

```
def registration_constructor(self):
    if currentuser.usertype != 'Teacher':
        self.Output_Label.setText("Sorry Cannot access this option as you are a Student")
        print("You do not have the teacher status required")
    else:
        print("Opening registration screen")
        self.RegistrationWindow = RegistrationScreen()
        self.RegistrationWindow.show()
        self.hide()

def personal_events_contructor(self):
    print("Opening Personal Events Screen")
    self.PersonalEventsWindow = PersonalEventsScreen()
    self.PersonalEventsWindow.show()
    self.hide()

def edit_user_constructor(self):
    print("Opening Edit User Splash Screen")
    self.EditUserWindow = EditSplashScreen()
    self.EditUserWindow.show()
    self.hide()

def create_new_user_contructor(self):
    if currentuser.usertype != 'Teacher':
        print("You do not have the teacher status required")
        self.Output_Label.setText("Sorry Cannot access this option as you are a Student")
    else:
        if self.CreateNewWindow is None:
            self.CreateNewWindow = CreateNewUserScreen()
        print("Opening Create New User Screen")
        self.CreateNewWindow.show()
        self.hide()

def create_event_constructor(self):
    if currentuser.usertype != 'Teacher':
        print("You do not have the teacher status required")
        self.Output_Label.setText("Sorry Cannot access this option as you are a Student")
    else:
        print("Opening the Create New Event Screen")
        self.hide()
        self.AE_Screen = CreateEventScreen()
        self.AE_Screen.show()
```

3.4.3 ITERATIVE TESTING

This class needs to be able to complete the following functions:

1. Open the registration screen, hiding itself
2. Open the analytics screen, hiding itself
3. Open the personal events screen, hiding itself
4. Open the edit user splash screen, hiding itself

5. Open the create new user screen, hiding itself
6. Open the create new event screen, hiding itself
7. Allow the user to log out
8. Allow the user to close the program down

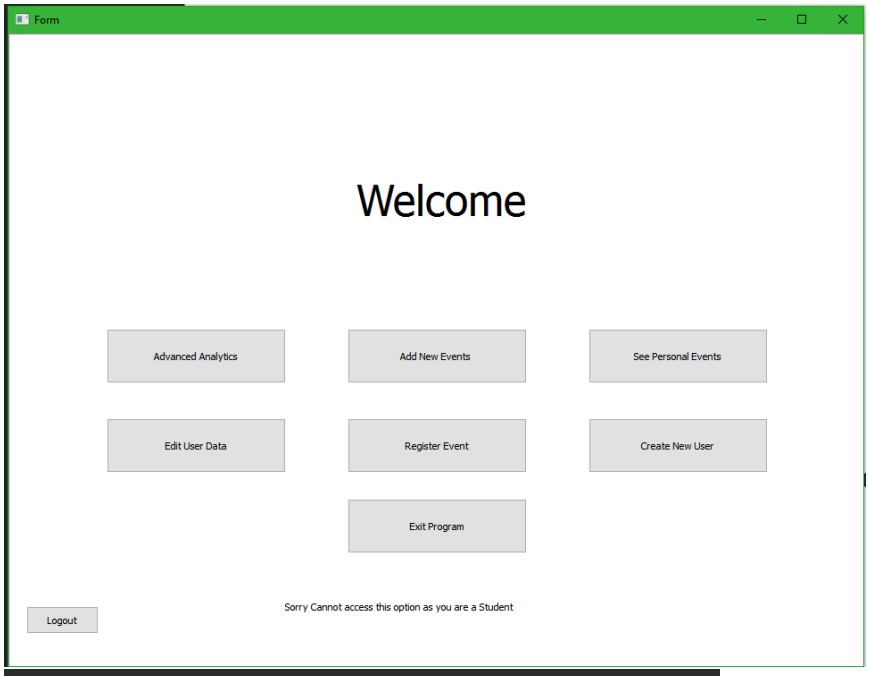
1. :

By pressing the appropriate button this class should present the user with either a error message or an instance of the registration screen.

```
self.REButton.clicked.connect(self.registration_constructor)
```

```
def registration_constructor(self):
    if currentuser.usertype != 'Teacher':
        self.Output_Label.setText("Sorry Cannot access this option as you are a Student")
        print("You do not have the teacher status required")
    else:
        print("Opening registration screen")
        self.RegistrationWindow = RegistrationScreen()
        self.RegistrationWindow.show()
        self.hide()
```

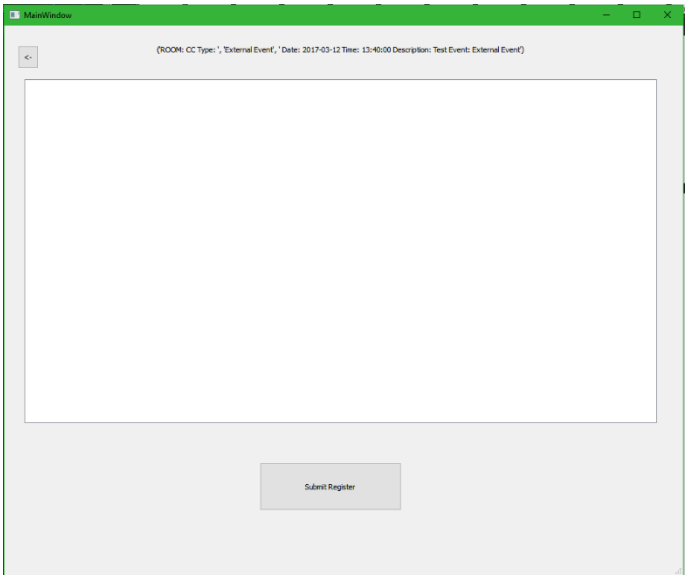
Using the Student Test data the following result is found:




```
You do not have the teacher status required
```

This is the expected result and as such, the program passes this test

Using the Teacher Test data the following result is found:



This is the expected result and as such, the program passes this test

```
Opening registration screen
```

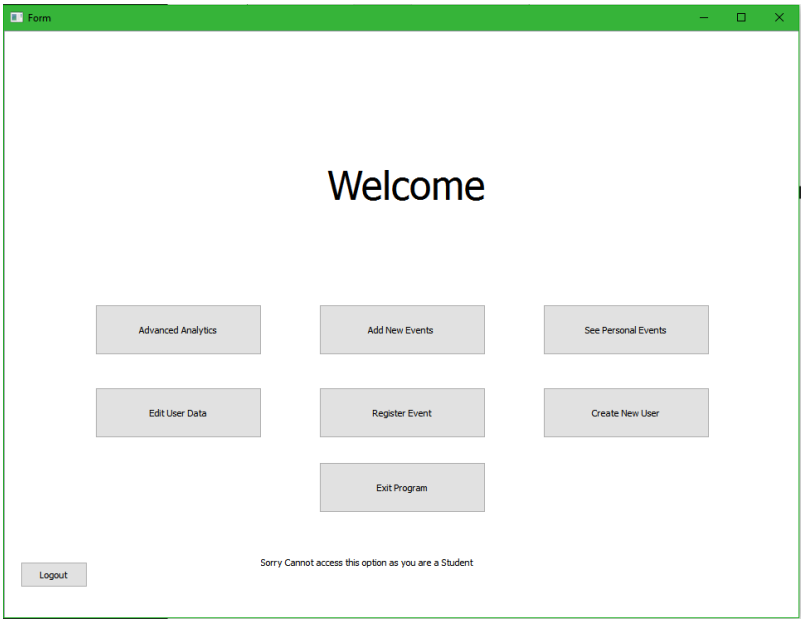
2. :

By pressing the appropriate button this class should present the user with either an error message or an instance of the analytics screen.

```
self.AAButton.clicked.connect(self.advanced_analytics_contructor)
```

```
def advanced_analytics_contructor(self):
    if currentuser.usertype != 'Teacher':
        print("You do not have the teacher status required")
        self.Output_Label.setText("Sorry Cannot access this option as you are a Student")
    else:
        print("Opening analytics Screen")
        self.AdvancedAnalyticsWindow = AdvancedAnalyticsScreen()
        self.AdvancedAnalyticsWindow.show()
        self.hide()
```

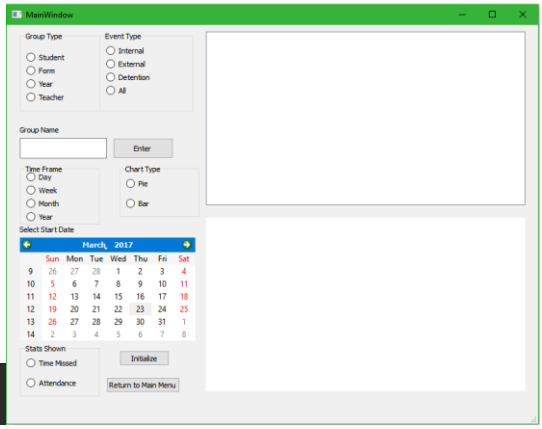
Using the Student Test Data the program produces the following result:




```
You do not have the teacher status required
```

This is the expected result and as such, the program passes this test

Using the Teacher Test Data, the program produces the following result:




```
Opening analytics Screen
```

This is the expected result and as such, the program has passed this test.

3. :

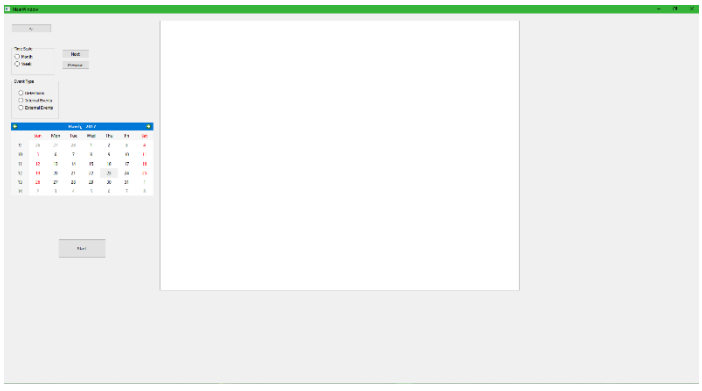
By pressing the appropriate button the program should launch the Personal Events Screen, hiding itself.

This function should work for both student and teachers as the function doesn't differentiate between them. Therefore, I will only use the Teacher Test Data for testing.

```
self.PEButton.clicked.connect(self.personal_events_contructor)
```

```
def personal_events_contructor(self):
    print("Opening Personal Events Screen")
    self.PersonalEventsWindow = PersonalEventsScreen()
    self.PersonalEventsWindow.show()
    self.hide()
```

The result of running this code is:



*Design subject to change

```
Opening Personal Events Screen
```

This is the expected result and as such, the program has passed this test

4. :

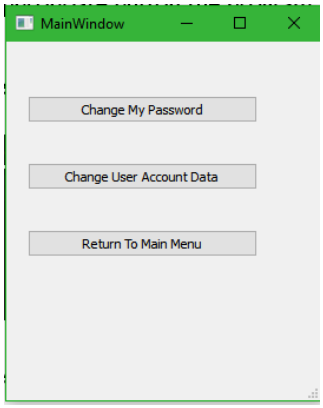
By pressing the appropriate button the program should open the Edit User splash screen, hiding itself.

This function doesn't consider usertype and therefore I will only be testing it for the Teacher Test Data

```
self.EUButton.clicked.connect(self.edit_user_constructor)
```

```
def personal_events_contructor(self):
    print("Opening Personal Events Screen")
    self.PersonalEventsWindow = PersonalEventsScreen()
    self.PersonalEventsWindow.show()
    self.hide()
```

Using the Teacher Test Data the result is:




```
Opening Edit User Splash Screen
```

This is the expected result and as such, the program has passed this test

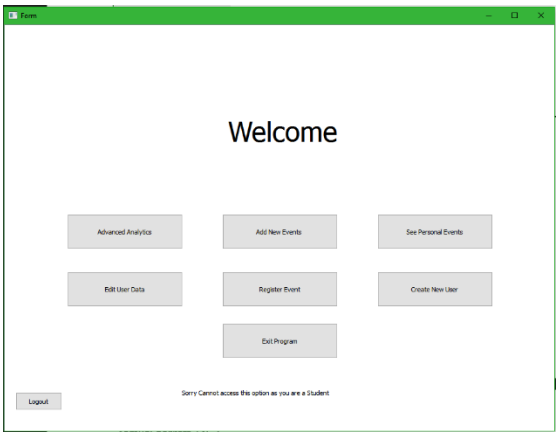
5. :

By pressing the appropriate button the program should either open the Create New User screen, hiding itself or produce an error if the user is not a teacher

```
self.CNUButton.clicked.connect(self.create_new_user_contructor)
```

```
def create_new_user_contructor(self):
    if currentuser.usertype != 'Teacher':
        print("You do not have the teacher status required")
        self.Output_Label.setText("Sorry Cannot access this option as you are a Student")
    else:
        if self.CreateNewWindow is None:
            self.CreateNewWindow = CreateNewUserScreen()
        print("Opening Create New User Screen")
        self.CreateNewWindow.show()
        self.hide()
```

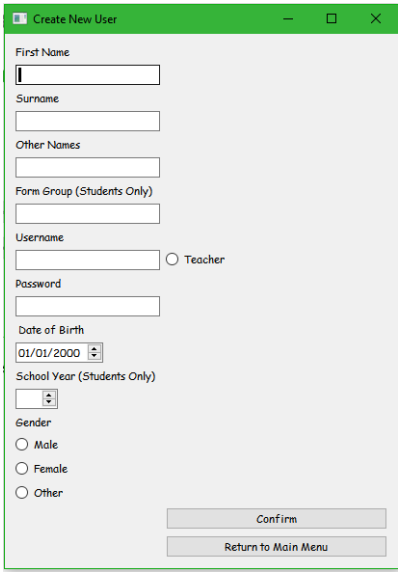
Using the Student Test Data the program produces the following result:




```
You do not have the teacher status required
```

this is the expected result and as such, the program passed this test

using the Teacher Test Data the program produced the following result:




```
Opening Create New User Screen
```

this is the expected result and as such, the program passed this test

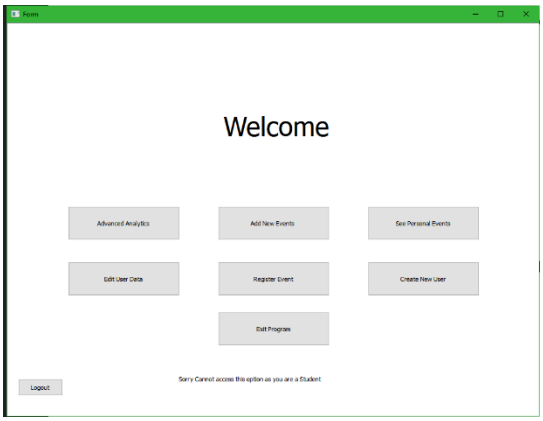
6. :

By pressing the appropriate button the program should either open the Create New Event screen, hiding itself or produce an error if the user is not a teacher

```
self.AEButton.clicked.connect(self.create_event_constructor)
```

```
def create_event_constructor(self):
    if currentuser.usertype != 'Teacher':
        print("You do not habe the teacher status required")
        self.Output_Label.setText("Sorry Cannot access this option as you are a Student")
    else:
        print("Opening the Create New Event Screen")
        self.hide()
        self.AE_Screen = CreateEventScreen()
        self.AE_Screen.show()
```

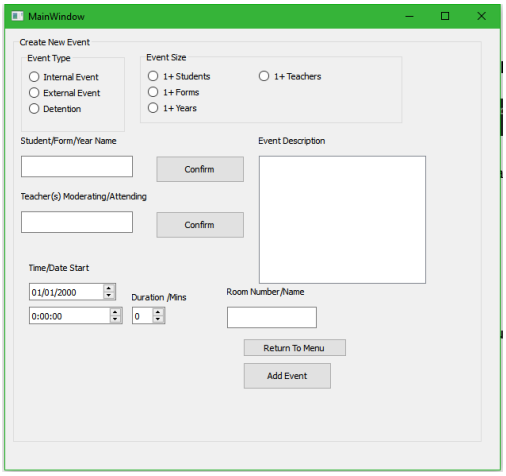
Using the Student Test Data the program produces the following result:




```
You do not have the teacher status required
```

This is the expected result and as such, the program passed this test.

Using the Teacher Test Data the program produces the following result:




```
Opening the Create New Event Screen
```

This is the expected result and as such, the program has passed this test.

7. :

By pressing the appropriate button the program should return the user to the login screen, clearing all details of the previous user

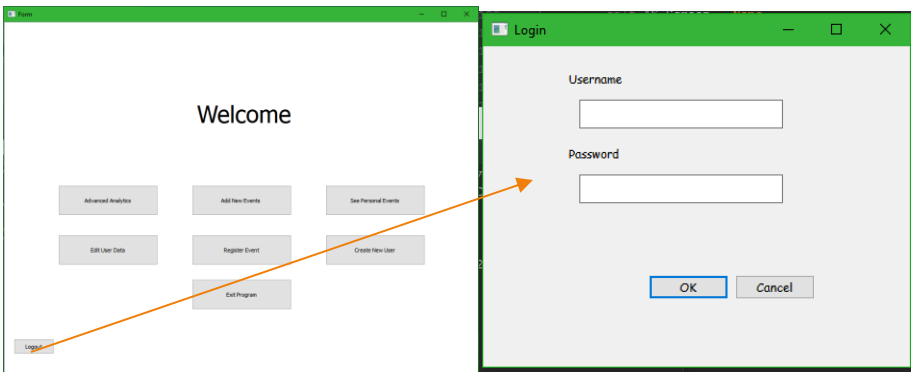
This will work the same for both Student and Teacher users therefore, I am only going to test this function with the Teacher User Data.

```
self.LogoutButton.clicked.connect(lambda: __miscfuncitons__.logout(self, LoginWindow, currentuser))
```

The main function behind this is stored in a separate file and relies on objects being passed as parameters

```
def logout(obj, inwindow, user):
    window = inwindow
    user.userid = None
    user.acctype = None
    user.usertype = None
    window.User_Field.setText("")
    window.Pass_Field.setText("")
    window.Output_Label.setText("")
    window.show()
    obj.close()
```

The Teacher User Data produces the following result:



This is the expected result and as such, the program passed this test

8. :

Upon a button press the program should close. This function works regardless of user type so I will be testing using the Teacher Test Data

```
self.ExitButton.clicked.connect(lambda: exit(0))
```

This is a very simple function and uses only one line of code.

Using the Teacher Test Data the program produces the correct result, this is hard to demonstrate due to the nature of the function leaving no evidence of the program having run.

With all tests passed I am happy that the Start or Splash screen is fully featured and functional and the code is now finalized.

3.4.4 FINAL CODE

This class is made up of an initialization class performing similar tasks to previously outlined __init__() classes. It is also comprised of various constructor methods for other screens which are triggered by button clicks

```
def __init__(self, parent=None):
    QtGui.QMainWindow.__init__(self, parent)
    self.CreateNewWindow = None
    self.setupUi(self)
    self.RegistrationWindow = None
    self.AdvancedAnalyticsWindow = None
    self.PersonalEventsWindow = None
    self.EditUserWindow = None
    self.AE_Screen = None
    self.data = []
    self.AEButton.clicked.connect(self.create_event_constructor)
    self.CNUButton.clicked.connect(self.create_new_user_contructor)
    self.EUButton.clicked.connect(self.edit_user_constructor)
    self.ExitButton.clicked.connect(lambda: exit(0))
    self.PEButton.clicked.connect(self.personal_events_contructor)
    self.LogoutButton.clicked.connect(lambda: __miscfuncitons__.logout(self, LoginWindow, currentuser))
    self.AAButton.clicked.connect(self.advanced_analytics_contructor)
    self.REButton.clicked.connect(self.registration_constructor)
```

3.4.4.1 __INIT__()

this class initialises all the variables and attributes within the class to either Nonetype or to empty lists as well as setting up the GUI using a build in metho d

it also sets up listeners for each button to check if it has been pressed and directs the program to the approprate method upon a corresponding click.

3.4.4.2 REGISTRATION_CONSTRUCTOR()

```
def registration_constructor(self):
    self.RegistrationWindow = RegistrationScreen()
    self.RegistrationWindow.show()
    self.hide()
```

this is one of many constructor methods in this class it simply creates an object using the 'RegistrationScreen' class, hides itself and shows the new object to the user.

3.4.4.3 ADVANCED_ANALYTICS_CONSTRUCTOR()

```
def advanced_analytics_contructor(self):
    self.AdvancedAnalyticsWindow = AdvancedAnalyticsScreen()
    self.AdvancedAnalyticsWindow.show()
    self.hide()
```

this constructor creates the 'AdvancedAnalyticsScreen' object, hides itself and shows the new object

3.4.4.4 PERSONAL_EVENTS_CONSTRUCTOR()

```
def personal_events_contructor(self):
    self.PersonalEventsWindow = PersonalEventsScreen()
    self.PersonalEventsWindow.show()
    self.hide()
```

this constructor creates the 'PersonalEventsScreen' class, shows it and hides itself

3.4.4.5 EDIT_USER_CONSTRUCTOR()

```
def edit_user_constructor(self):
    self.EditUserWindow = EditSplashScreen()
    self.EditUserWindow.show()
    self.hide()
```

this constructor creates the 'EditSplashScreen' object, shows it and hides itself

3.4.4.6 CREATE_NEW_USER_CONSTRUCTOR()

```
def create_new_user_contructor(self):
    if self.CreateNewWindow is None:
        self.CreateNewWindow = CreateNewUserScreen()
    self.CreateNewWindow.show()
    self.hide()
```

this constructor creates the 'CreateNewUserScreen', shows it and hides itself

3.4.4.7 CREATE_EVENT_CONSTRUCTOR()

```
def create_event_constructor(self):
    if currentuser.usertype != 'Teacher':
        self.Output_Label.setText("Sorry Cannot access this option as you are a Student")
    else:
        self.hide()
        self.AE_Screen = CreateEventScreen()
        self.AE_Screen.show()
```

this constructor creates the 'CreateEventScreen' but checks the user type of the currently logged in user, stored in the User object, if they are a teacher it will launch the window but will

reject them otherwise

3.5 CREATE NEW USER SCREEN

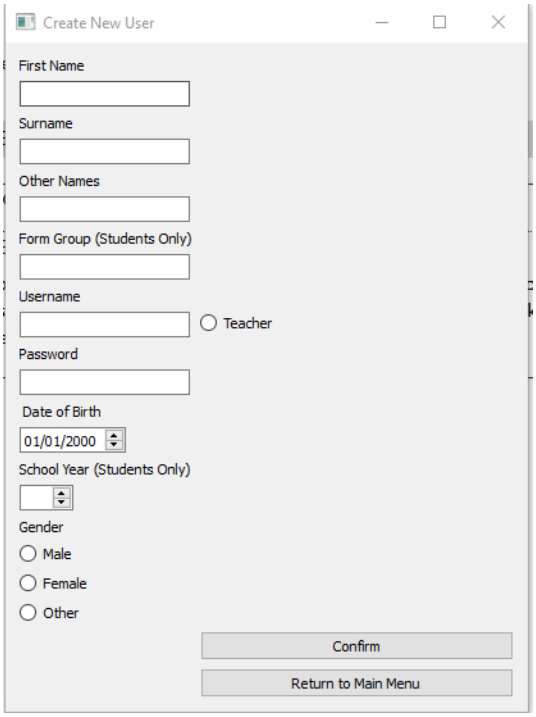
3.5.1 PROJECT DECOMOSITION

3.5.1.1 CLASS REQUIREMENTS

This class will need to be able to efficiently take in all required data for the creation of a new user. So that no fields are left blank in the DB. It will also need to run presence checks on each field to stop null values from being stored in the database causing errors later.

3.5.2 DEVELOPMENT

I started by designing the UI in QtDesigner



- I used Qlineedits for entering the bulk of the data
- I also used a Datetime widget for entering date of birth
- I used a spin box for year input
- I used a group of 3 radio buttons for gender selection

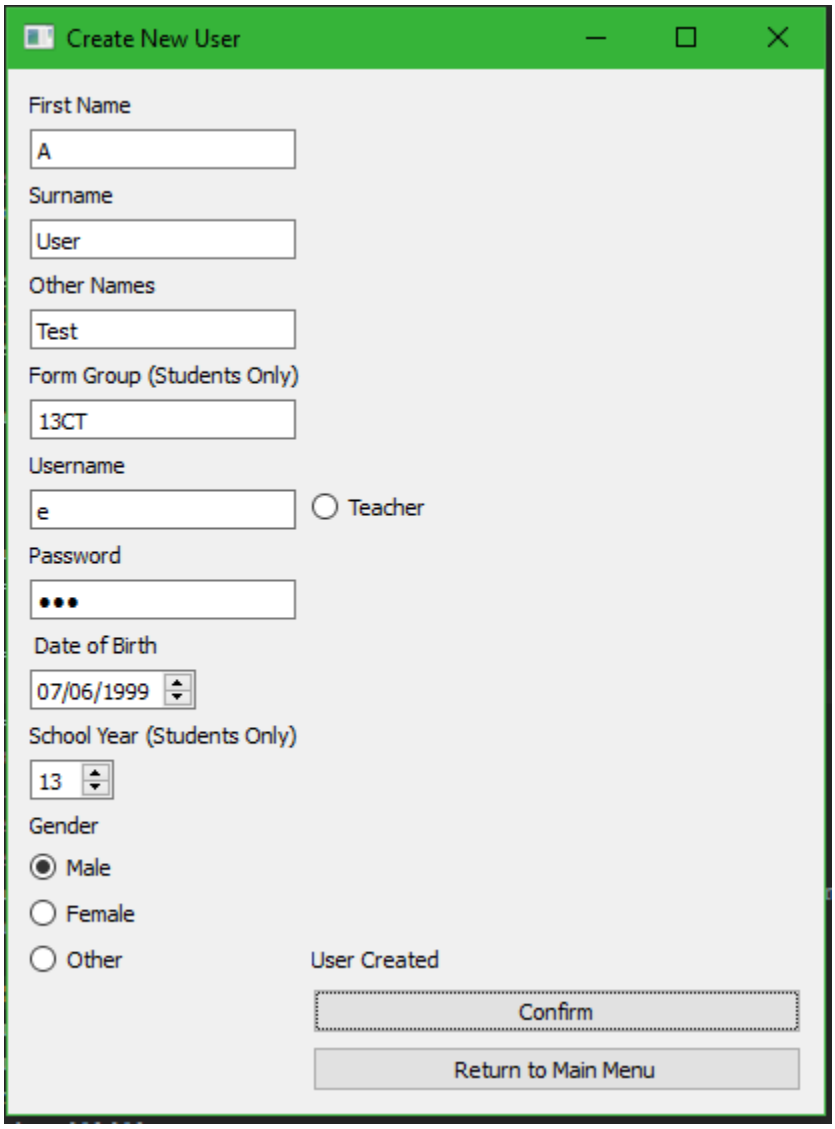
3.5.3 ITERATIVE TESTING

The code will need to take in all values from the PyQt GUI objects. To test this I will enter the predefined Test Data

```
FName = self.FNameInput.text()
SName = self.SNameInput.text()
UName = self.UNameInput.text()
Password = self.PasswordInput.text()
Hashed_Password = __miscfuncitons__.hashpass(Password)
if (FName or SName or UName or Password) == "":
    self.label_7.setText("Incorrect Values Try again")
    sys.exit(0)
if self.GenderMaleButton.isChecked():
    GenderID = 1
elif self.GenderFemaleButton.isChecked():
    GenderID = 2
else:
    GenderID = 3
if self.IfTeacherButton.isChecked():
    cmdtype = "Teacher"
else:
    cmdtype = "Student"

if cmdtype == "Student":
    OName = self.ONameInput.text()
    DoB = self.DoBInput.text()
    Year = self.YearInput.text()
    Form = self.FormInput.text()
    print(FName,SName,OName,UName,Password,Hashed_Password,DoB,Year,Form)
```

With the Student Test Data the program produces this result:



As you can see this is the expected result and all data has been ingested and is stored in a usable format

Therefore, the program has passsed this test

```
A User Test e 123 0a6c55d58f1794836da98695f0558a40e6a98c6dd1ed79a3cb0dfc1cf5cad5c8
 :5f0e3ce63489496c9745e1e0d1f8de8f 07/06/1999 13 13CT
13CT [(1,)]
```

The teacher user enters less data as some fields are not required for creation of a Teacher User in the database. Therefore if the code works for a student the code will work just as well for a teacher, for this reason I will only test the data ingest using the Student Test Data

The program must go on to process the data and extrapolate it to find foreign keys necessary for user creation

```
Year = int(Year)
YearID = Year - 6
cmd = 'SELECT FormID FROM Forms WHERE YearID=? AND Form_Name = ?'
cur.execute(cmd, (YearID, Form[2:],))
data = cur.fetchall()
print(Form, data)
FormID = data[0][0]
# GET NEXT STUDENTID
if FName or OName or SName or UName or Password or DoB or Year or Form == "" or None:
    self.label_7.setText("Incorrect Values Try again")
cmd = 'SELECT StudentID FROM Students'
cur.execute(cmd)
data = cur.fetchall()
StudentID = data[-1][0] + 1
cmd = 'INSERT INTO Students (StudentID, Forename, Other_Names, Surname, YearID, FormID, Date_of_Birth, Username, Password , User_Level,GenderID) ' \
      'VALUES (?,?,?,?,?,?,?,?,?,?,?)'
cur.execute(cmd,
            (StudentID, FName, OName, SName, YearID, FormID, DoB, UName, Hashed_Password, 0, GenderID,))
con.commit()

self.label_7.setText("User Created")
```

Using the student Test data the program produces the following output:

2	7	A	Test	User	7	1	07/06/1999	e	0a6c55d58f1794836da98695f0558a40e6a98c6dd1ed79a3cb0dfc1cf5cad5…	0	1

This shows the created user in the Database. All fields are filled out with the correct keys and information. This is the expected value and therefore, the program has passed this test

These are the two main functions the program needed to perform and as it has passed these I feel the code is complete

3.5.4 FINAL CODE

This class is made up of an initialization class performing similar tasks to previously outlined __init__() classes. It also has its main function stored in a second method which takes input of all the data in the form, verifies it and sends it to the correct place in the DB.

3.5.4.1 __INIT__()

```
def __init__(self, parent=None):
    QtGui.QWidget.__init__(self, parent)
    self.setupUi(self)
    self.ConfirmButton.clicked.connect(self.append)
    self.ReturnButton.clicked.connect(lambda: returntostart(self))
```

This method simply sets up the listeners for the buttons and sets up the GUI using PyQt methods. All data in this class is stored in local variables for security to prevent sensitive user information from being public to the entire program

3.5.4.2 APPEND()

```
def append(self):
    FName = self.FNameInput.text()
    SName = self.SNameInput.text()
    UName = self.UNameInput.text()
    Password = self.PasswordInput.text()
    Hashed_Password = __miscfuncitons__.hashpass(Password)

    if (FName or SName or UName or Password) == "":
        self.label_7.setText("Incorrect Values Try again")
        sys.exit(0)
    if self.GenderMaleButton.isChecked():
        GenderID = 1
    elif self.GenderFemaleButton.isChecked():
        GenderID = 2
    else:
        GenderID = 3
    if self.IfTeacherButton.isChecked():
        cmdtype = "Teacher"
    else:
        cmdtype = "Student"
    if cmdtype == "Student":
        OName = self.ONameInput.text()
        DoB = self.DoBInput.text()
        Year = self.YearInput.text()
        Form = self.FormInput.text()
        Year = int(Year)
        YearID = Year - 6
        cmd = 'SELECT FormID FROM Forms WHERE YearID=? AND Form_Name = ?'
        cur.execute(cmd, (YearID, Form[2:],))
        data = cur.fetchall()
        print(Form, data)
        FormID = data[0][0]
        # GET NEXT STUDENTID
        if FName or OName or SName or UName or Password or DoB or Year or Form == "" or None:
            self.label_7.setText("Incorrect Values Try again")
        cmd = 'SELECT StudentID FROM Students'
        cur.execute(cmd)
        data = cur.fetchall()
        StudentID = (len(data) + 1)
```

The method starts by taking input from all the various fields on the input screen. It also hashes the entered password in the correct format for storage using the 'hashpass' function

Once it has done this is fetches any needed data from the DB; these include the foreign keys to be stored in the student or teacher table, as the user will have entered a seconary more user friendly form identifier. It then performs a presence check on all the data and returns an error to the user I

Once it has all the required data to perform the database injection it executes an insertion query filling in the blanks with the collected data. It performs different insertions depending on whether a student or a teacher user is being created. Also for a teacher to be created the creating user must be of teacher status or higher. It checks for this and rejects the user if they do not meet the criteria.

```
cmd = '''INSERT INTO Students
        (StudentID, Forename, Other_Names, Surname, YearID, FormID, Date_of_Birth, Username,
         Password , User_Level,GenderID) VALUES (?,?,?,?,?,?,?,?,?,?)'''
cur.execute(cmd,
            (StudentID, FName, OName, SName, YearID, FormID, DoB, UName, Hashed_Password, 0, GenderID,))
con.commit()

self.label_7.setText("User Created")

cmdtype == "Teacher":
if currentuser.acctype >= 1:
    cmd = 'SELECT TeacherID FROM Teachers'
    cur.execute(cmd)
    data = cur.fetchall()
    TeacherID = len(data) + 1
    cmd = 'INSERT INTO Teachers (TeacherID, Forename,Surname,Username,Password,User_Level) VALUES (?,?,?,?,?)'
    cur.execute(cmd, (TeacherID, FName, SName, UName, Hashed_Password, 1,))
    con.commit()

else:
    pass  ######user below teac
```

3.6 CREATE NEW EVENT SCREEN

3.6.1 PROJECT DECOMOSITION

3.6.1.1 CLASS REQUIREMENTS

This class will be required to take in all data related to the creation of a new event through various PyQt objects. It will then need to process the input to be in the correct form for DB storage as well as calculating other pieces of data to streamline the user experience

3.6.2 ITERATIVE TESTING

The first test will be to make sure the code ingests all entered data correctly and extrapolates it to find any other required information

The ingest of data in this screen is three fold. Teacher data and student data are gathered on a user-to-user basis by a button press and the rest of the details are taken in upon event creation

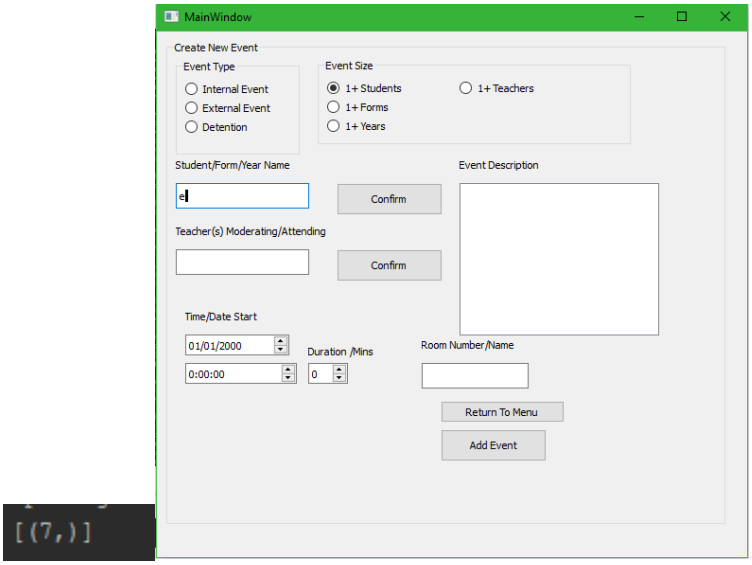
To test the Student input:

```
self.ListPart = []
```

```
self.SFY_Confirm_Button.clicked.connect(
    lambda: self.ListPart.append(self.getlistpart(self.SFY_Input.text())))
```

```
def getlistpart(self, inp):
    self.Etype = ""
    if self.S_Rad.isChecked():
        self.Etype = "Student"
    if self.F_Rad.isChecked():
        self.Etype = "Form"
    if self.Y_Rad.isChecked():
        self.Etype = "Year"
    self.PartData = []
    if self.Etype == "Student":
        inp = inp.split()
        if inp[0] == "":
            self.OutputLabel.setText("Invalid Name")
            QtGui.QMessageBox.information(self, "Error", "Invalid Name")
        self.SFY_Input.setText("")
        try:
            cmd = 'SELECT StudentID FROM Students WHERE Forename =? AND Surname = ?'
            cur.execute(cmd, (inp[0], inp[1],))
        except:
            cmd = 'SELECT StudentID FROM Students WHERE Username = ?'
            cur.execute(cmd, (inp[0],))
        checkdata = cur.fetchall()
        if not checkdata:
            print("No user with that name")
            QtGui.QMessageBox.information(self, "Error", "No such user")
        else:
            if self.PartData.__contains__(inp):
                QtGui.QMessageBox.information(self, "Error", "Participant already present")
                print("Person already added")
            else:
                self.studcount += 1
                return checkdata
    if self.Etype == "Form":
        if inp == "":
            QtGui.QMessageBox.information(self, "Error", "No such form")
        self.SFY_Input.setText("")
        cmd = 'SELECT FormID FROM Forms WHERE Form_Name LIKE ?'
        cur.execute(cmd, (inp,))
        checkdata = cur.fetchall()  ##for EType being form or above need iterative function to generate database
        if not checkdata:  ##entries where it goes through every student who is a member of those groups
            print("NO SUCH FORM")
            QtGui.QMessageBox.information(self, "Error", "No such form")
        else:
            cmd = 'SELECT StudentID FROM Students WHERE FormID = ?'
            print("FORMCHECKDATA", checkdata)
            cur.execute(cmd, (checkdata[0][0],))
            studdata = cur.fetchall()
            self.studcount += len(studdata[0])
            return studdata
```

Using the Student Test Data the program produces this result:



This is the expected result, it has used the input, determined whether it is a username or first and last name and has returned the students' studentID

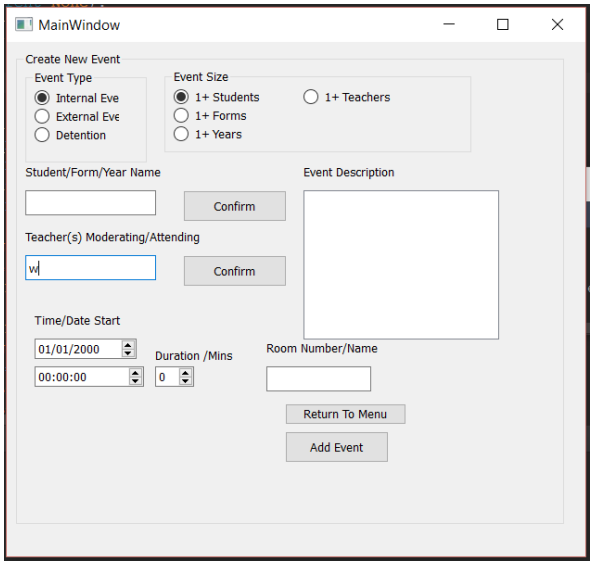
Therefore, the program has passed this test

To test the Teacher Input:

```
self.T_Confirm_Button.clicked.connect(
    lambda: self.ListTeach.append(self.getlistteach(self.T_Input.text())))
```

```
self.ListTeach = []
```

Using the Teacher Test data the program produces the following output




```
[(5,)]
```

this is the expected output and as such, the program has passed this test

To test Form input:

I will use the form name 13CT

I must then also test that upon event creation all remaining data is taken input of correctly. To do this I will use the following test data:

Time Start	Date	Duration	Room Name	Description
12/03/2017	13:40:00	60	CC	Test Event: Internal Event

3.6.3 FINAL CODE

This class like all of my classes starts with an __init__() method which initializes variables and listens for button presses

3.6.3.1 __INIT__()

```
def __init__(self, parent=None):
    QtGui.QMainWindow.__init__(self, parent)
    self.setupUi(self)
    self.ListTeach = []
    self.ListPart = []
    self.studcount = 0
    self.ReturnButton.clicked.connect(lambda: returntostart(self))
    self.SFY_Confirm_Button.clicked.connect(
        lambda: self.ListPart.append(self.getlistpart(self.SFY_Input.text())))
    self.T_Confirm_Button.clicked.connect(
        lambda: self.ListTeach.append(self.getlistteach(self.T_Input.text())))
    self.AddButton.clicked.connect(self.addevent)
    self.PartData = None
    self.Etype = None
    self.TeachData = None
    self.date = None
    self.timestart = None
    self.duration = None
    self.description = None
    self.EtypeID = None
    self.eventid = None
    self.tlb = None
    self.STUDID = None
    self.tub = None
    self.TEACHID = None
```

here the classes attributed lists and variables are initialized as empty and Nonetype respectively

it also creates listeners for button presses and runs the appropriate method when a button is pressed

3.6.3.2 GETLISTPART()

This method is run when the button next to the student (user)name input field is pressed. It runs SQL query searching for the information related to the entered student name. if it cannot find a student matching it will alert the user otherwise it appends a list of student participants for use later

```
def getlistpart(self, inp):
    self.Etype = ""
    if self.S_Rad.isChecked():
        self.Etype = "Student"
    if self.F_Rad.isChecked():
        self.Etype = "Form"
    if self.Y_Rad.isChecked():
        self.Etype = "Year"
    self.PartData = []
    if self.Etype == "Student":
        inp = inp.split()
        if inp[0] == "":
            self.OutputLabel.setText("Invalid Name")
            QtGui.QMessageBox.information(self, "Error", "Invalid Name")
        self.SFY_Input.setText("")
        try:
            cmd = 'SELECT StudentID FROM Students WHERE Forename =? AND Surname = ?'
            cur.execute(cmd, (inp[0], inp[1],))
        except:
            cmd = 'SELECT StudentID FROM Students WHERE Username = ?'
            cur.execute(cmd, (inp[0],))
        checkdata = cur.fetchall()
        if not checkdata:
            print("No user with that name")
            QtGui.QMessageBox.information(self, "Error", "No such user")
        else:
            if self.PartData.__contains__(inp):
                QtGui.QMessageBox.information(self, "Error", "Participant already present")
                print("Person already added")
            else:
                self.studcount += 1
                return checkdata
    if self.Etype == "Form":
        if inp == "":
            QtGui.QMessageBox.information(self, "Error", "No such form")
        self.SFY_Input.setText("")
        cmd = 'SELECT FormID FROM Forms WHERE Form_Name LIKE ?'
        cur.execute(cmd, (inp,))
        checkdata = cur.fetchall()  ##for EType being form or above need iterative function to generate database
        if not checkdata:  ##entries where it goes through every student who is a member of those groups
            print("NO SUCH FORM")
            QtGui.QMessageBox.information(self, "Error", "No such form")
        else:
            cmd = 'SELECT StudentID FROM Students WHERE FormID = ?'
            print("FORMCHECKDATA", checkdata)
            cur.execute(cmd, (checkdata[0][0],))
            studdata = cur.fetchall()
            self.studcount += len(studdata[0])

            return studdata
```

```
if self.Etype == "Year":
    if inp == "":
        QtGui.QMessageBox.information(self, "Invalid Year Entry", "Invalid Year Entry")
    self.SFY_Input.setText("")
    cmd = 'SELECT YearID FROM Years WHERE "Year" =?'
    cur.execute(cmd, (inp,))
    ydata = cur.fetchall()
    if not ydata:
        QtGui.QMessageBox.information(self, "Error", "No Such Year Group")
        print("No Such Year Group")
    cmd = 'SELECT StudentID FROM Students WHERE YearID =?'
    cur.execute(cmd, (ydata[0]))
    studdata = cur.fetchall()
    self.studcount += len(studdata[0])
    return studdata
```

Due to the layout of the input form the program must first work out what kind of data has been provided to it. Using the radio buttons in the form to determine if it has been provided with the name of a student, form or year group it then performs a corresponding SQL query.

It then checks for presence of data and alerts the user if no data corresponding to the group specified can be found.

3.6.3.3 GETLISTTEACH()

```
def getlistteach(self, inp):
    if self.T_Rad.isChecked():
        self.Etype = "Teacher"
    self.T_Input.setText("")
    self.TeachData = []
    inp = inp.split()
    print(inp)
    if inp[0] == "":
        QtGui.QMessageBox.information(self, "Invalid Name", "Invalid Name")
    if len(inp) == 1:
        cmd = 'SELECT TeacherID FROM Teachers WHERE Username = ?'
        cur.execute(cmd, (inp[0],))
    else:
        cmd = 'SELECT TeacherID FROM Teachers WHERE Forename =  ? AND Surname = ?'
        cur.execute(cmd, (inp[0], inp[1],))
    checkdata = cur.fetchall()
    print(checkdata)
    if not checkdata:
        print("No such member of staff")
        QtGui.QMessageBox.information(self, "No such member of staff", "No such Member of Staff"
    else:
        self.TeachData.append(inp)
        return checkdata
```

This method is much shorter than the equivalent for student input as it does not need to determine group type or have differing SQL queries dependent on the group type.

This simply searches the database for the entered teacher username and returns the teacherID or an error if the member of staff cannot be found in the DB.

3.6.3.4 ADDEVENT()

```
def addevent(self):  ###triggered when addevent button is clicked
    print(self.ListPart, self.ListTeach)
    # iteratively goes through each in the list of students andor teachers and adds an associated commitment linked to their account
    cmd = 'SELECT RoomID FROM Rooms WHERE RoomNameNumber = ?'
    cur.execute(cmd, (self.RoomEntry.text(),))
    RoomID = cur.fetchall()
    print("ROOMID", RoomID)
    if self.IE_Rad.isChecked():
        self.EtypeID = 1
    if self.EE_Rad.isChecked():
        self.EtypeID = 2
    if self.D_Rad.isChecked():
        self.EtypeID = 3
    self.date = str((self.DateIn.date()).toPyDate())
    self.timestart = str((self.TimeStart.time()).toPyTime())
    self.duration = self.DurationIn.text()
    self.description = str(self.Desc_Box.toPlainText())

    cmd = 'SELECT EventID FROM Events'
    cur.execute(cmd)
    etempdata = cur.fetchall()
    print(etempdata)
    etempdata = etempdata[-1:][0]
    print(etempdata)
    self.eventid = int(etempdata[0]) + 1
    if not RoomID:
        RoomID = 5
    cmd = 'INSERT INTO Events (EventID, RoomID, TypeID, "Date", "Time", Description, Duration) VALUES (?,?,?,?,?,?,?)'
    cur.execute(cmd,
                (self.eventid, RoomID[0][0], self.EtypeID, self.date, self.timestart, self.description,
                 self.duration,))
    con.commit()
    print(self.ListPart)
    if self.Etype == "Teacher":
        self.add_teachers()
    if self.Etype != "Teacher" and self.studcount != 0:
        self.add_students()
        self.add_teachers()
    if self.Etype != "Teacher" and self.studcount == 0:
        QtGui.QMessageBox.information(self, "Error", "No students to add")
        print("No students to add")
```

this method adds the main event data into the Events table in the DB. It also takes input of the rest of the data from the form. Such as RoomID and event type

once it has determined the next eventID it enters all the collected data into the database

then it runs functions to add the necessary data into the CommitmentStudent and CommitmentTeacher DB tables

3.6.3.5 ADD_STUDENTS()

```
def add_students(self):
    print(self.ListPart)
    for Student in self.ListPart[0]:  ##CONTWORK Make this work for forms, years etc.
        if Student:
            print(Student, "THIS IS THE STUDENT DATA ")
            self.STUDID = Student
            cmd = 'SELECT ComID FROM CommitmentsStudent'
            cur.execute(cmd)
            tempdata = cur.fetchall()
            x = tempdata[-1:][0][0]
            x = int(x) + 1
            cmd = 'SELECT * FROM Events JOIN CommitmentsStudent WHERE Events.EventID = ' \
                  '(SELECT CommitmentsStudent.EventID WHERE CommitmentsStudent.StudentID = ?)' \
                  ' AND Events.Date = ? AND Events.Time BETWEEN ? AND ?'
            self.tlb = self.timestart
            datetime_object = datetime.strptime(self.timestart, "%H:%M:%S")
            self.tub = datetime_object + timedelta(minutes=int(self.duration))
            self.tub = self.tub.time()
            self.tub = str(self.tub)
            cur.execute(cmd, (Student[0], self.date, self.tlb, self.tub,))
            print(Student[0], self.date, self.tlb, self.tub)
            checkdata = cur.fetchall()
            print(checkdata, "Fin is a Finlei")
            if checkdata:
                if self.EtypeID == 3:
                    if checkdata[2] == 3:
                        error_string = "User % has a double detention"
                        error_string = error_string.replace("%", str(Student[0]))
                        QtGui.QMessageBox.information(self, "Error", error_string)
                        print("DOUBLE DETENTION")
                        pass
                    else:
                        cmd = 'INSERT INTO CommitmentsStudent (ComID, StudentID, EventID) VALUES (?,?,?)'
                        print(((len(tempdata)), Student, self.eventid))
                        cur.execute(cmd, (x, Student[0],
                                          self.eventid), )  # PossFeature Add check to see if they want to override with detention
                        cmd = 'DELETE FROM CommitmentsStudent WHERE EventID = ?'
                        cur.execute(cmd, (checkdata[0],))
                        con.commit()
                else:
                    error_string = "User % has a double event"
                    error_string = error_string.replace("%", str(Student[0]))
                    QtGui.QMessageBox.information(self, "Error", error_string)
                    print("DOUBLE EVENT")
                    pass
            else:
                cmd = 'INSERT INTO CommitmentsStudent (ComID, StudentID, EventID) VALUES (?,?,?)'
                print(((len(tempdata)), Student, self.eventid))
                cur.execute(cmd, (x, Student[0], self.eventid), )
                con.commit()
        else:
            pass
```

this method iteratively goes through the previously compiled list of StudentIDs and individually adds an entry into CommitmentsStudent specific to them. It starts by determining the next available CommitmentID or ComID in the DB. it then checks to see if the current user already has an event booked at the time of the new event. If they do it looks to see the nature of the event. If a detention is being added and the clashing event is not a detention it will overwrite it; otherwise it will alert the user to the fact that there has been a clash and the event has not been added for that user. if there are no clashing events it simply inserts the necessary details into the DB

3.6.3.6 ADD_TEACHERS()

```
def add_teachers(self):
    for Teacher in self.ListTeach:
        if Teacher:
            self.TEACHID = Teacher
            cmd = 'SELECT ComID FROM CommitmentsTeacher'
            cur.execute(cmd)
            tempdata = cur.fetchall()
            x = tempdata[-1:][0][0]
            x = int(x) + 1
            cmd = 'SELECT * FROM Events INNER JOIN CommitmentsTeacher WHERE CommitmentsTeacher.TeacherID = ?' \
                  ' AND Events.Date = ? AND Events.Time BETWEEN ? AND ?'
            cur.execute(cmd, (Teacher[0][0], self.date, self.tlb, self.tub,))
            checkdata = cur.fetchall()
            if checkdata:
                print("Double event")
                error_string = "Teacher % has a double detention"
                error_string = error_string.replace("%", str(Teacher[0][0]))
                QtGui.QMessageBox.information(self, "Error", error_string)
                pass
            else:
                cmd = 'INSERT INTO CommitmentsTeacher (ComID, TeacherID, EventID) VALUES (?,?,?)'
                cur.execute(cmd, (x, Teacher[0][0], self.eventid,))
                con.commit()
        else:
            pass
```

this method performs the same task as the last but for the teachers added to an event. It determines the next ComID to be used and checks to see if the teacher in question is already booked in for an event. The type of event doesn't not matter in this circumstance. If there is a clash the new event will not be added for the teacher and the user will be notified. Otherwise the event will be added into CommitmentsTeacher with the appropriate information

3.6.3.7 NOTES

```
class CreateNewUserScreen(QtGui.QWidget, Ui_CreateWindow):
    def __init__(self, parent=ChangePasswordScreen):
        self.parent = parent
        QtGui.QWidget.__init__(self)
        self.setupUi(self)
        self.ConfirmButton.clicked.connect(self.append)
        self.ReturnButton.clicked.connect(lambda: returntostart(self))

    def append(self):
        FName = self.FNameInput.text()
        SName = self.SNameInput.text()
        UName = self.UNameInput.text()
        Password = self.PasswordInput.text()
        pcheck, ucheck = __miscfuncitons__.checkdata(UName, Password, cur)
        if not pcheck["Overall"]:
            error_string = "Password is not valid %s"
            error_string = error_string.replace("%s", str(pcheck))
            print(error_string)
            QtGui.QMessageBox.information(self, "ERROR", error_string)
        elif ucheck:
            QtGui.QMessageBox.information(self, "ERROR",
                                          "Username is not valid, less than 8 characters or already in use")
        else:
            Hashed_Password = self.parent.hashpass(self,Password)
            if (FName or SName or UName or Password) == "":
```

Figure 1

In this object I utilised the OOP construct of parent classes, whereby child classes can access the attributes and methods defined within their parents. Here I inherited the

'ChangePasswordScreen' to access the hashpass method which takes an unhashed variable and retuns a hashed version with a salt. The code where this is used can be seen in **Figure 1**

3.7 SEE EVENTS SCREEN

3.7.1 PROJECT DECOMPOSITION

This screen needs to be able to show the currently logged in user their events in a given time frame. It must also allow the user to filter the events by type and to pan through time periods both future, past and present

3.7.2 DEVELOPMENT AND ITERATIVE TESTING

3.7.2.1 CLASS REQUIREMENTS

3.7.3 FINAL CODE

3.7.3.1 __INIT__()

```
def __init__(self, parent=None):
    QtGui.QMainWindow.__init__(self, parent)
    self.setupUi(self)
    self.MMButton.clicked.connect(lambda: returntostart(self))
    startdate = datetime.today().date()
    self.currentendbound = None
    self.currentstartbound = None
    self.data = []
    self.DataModel = None
    ts = 0
    if self.MRad.isChecked():
        ts = 1
    if self.WRad.isChecked():
        ts = 0
    self.PrevButton.clicked.connect(
        lambda: self.main(self.getlastbound(ts, self.currentstartbound), self.currentstartbound))
    self.NextButton.clicked.connect(
        lambda: self.main(self.currentendbound, self.getnextbound(ts, self.currentendbound)))
    self.InitialiseButton.clicked.connect(
        lambda: self.main(startdate - timedelta(days=startdate.weekday()), self.getnextbound(ts, startdate)))
```

this class initializes all variables and arrays within the object and sets up listeners for button presses. These then run functions

the button presses run functions using the lambda function within python, this allows me to pass parameters into these methods which are calculated elsewhere.

3.7.3.2 GETLASTBOUND()

```
def getlastbound(self, ts, sd):
    if self.MRad.isChecked():
        ts = 1
    if self.WRad.isChecked():
        ts = 0
    print(ts, "TSTS")
    if ts == 0:  # if time show = Week
        lmonday = sd - timedelta(days=sd.weekday(), weeks=1)
        try:
            lmonday = lmonday.date()
        except:
            lmonday = lmonday
        print("LBIP", sd)
        print("LBOP", lmonday)
        return lmonday

    if ts == 1:  # if time show = Month

        lmonth = sd + relativedelta.relativedelta(months=-1)
        try:
            lmonth = lmonth.date()
        except:
            pass
        print(lmonth)
        return lmonth

        pass
```

this method is used to find the lower bound of the time range for selecting events from the database. It uses the timedelta module to subtract months or weeks, depending on what radio button is selected, from the selected 'start date' from the PyQt calendar widget

it is used when the 'Previous' button is pressed and returns the data directly into the self.main() method.

3.7.3.3 GETNEXTBOUND()

```
def getnextbound(self, ts, sd):
    if self.MRad.isChecked():
        ts = 1
    if self.WRad.isChecked():
        ts = 0
    if ts == 0:  # if time show = Week
        startdate = str(sd)
        weekday = 0
        d = datetime.strptime(startdate, '%Y-%m-%d')
        t = timedelta((7 + weekday - d.weekday()) % 7)
        nweek = (d + t).strftime('%Y-%m-%d')
        print(nweek)  #
        nmonday = nweek

        return nmonday

    if ts == 1:  # if time show = Month

        return sd + relativedelta.relativedelta(months=1)
```

this function performs the inverse of the previous with it selecting the upper bound of the date range.

It adds a week or a month to the 'start date' depending on the radio button that is selected

This function is also used in a lambda function to directly return data into the self.main() method

3.7.3.4 MAIN()

```
def main(self, sd, ed):
    self.currentstartbound = sd
    self.currentendbound = ed
    grouptype = None
    if self.DRad.isChecked():
        grouptype = 3
    elif self.IERad.isChecked():
        grouptype = 1
    elif self.EERad.isChecked():
        grouptype = 2
    if currentuser.usertype == 'Student':  ## if studentEvents
        if grouptype is not None:
            cmd = '''SELECT (SELECT Events_Type.Description FROM Events_Type WHERE Events_Type.TypeID= Events.TypeID),
                                Events.Description,(SELECT Rooms.RoomNameNumber FROM Rooms WHERE Rooms.RoomID = Events.RoomID)
                               ,Events.Date,Events.Time,Events.Duration FROM Events
                                INNER JOIN CommitmentsStudent
                                WHERE Events.EventID = (SELECT CommitmentsStudent.EventID WHERE CommitmentsStudent.StudentID = ?)
                                AND Events.Date BETWEEN ? AND ? AND Events.TypeID = ? '''
            cur.execute(cmd, (currentuser.userid, str(sd), str(ed), grouptype,))
        else:
            cmd = '''SELECT (SELECT Events_Type.Description FROM Events_Type WHERE Events_Type.TypeID= Events.TypeID),Events.Description,
              (SELECT Rooms.RoomNameNumber FROM Rooms WHERE Rooms.RoomID = Events.RoomID)
             ,Events.Date,Events.Time,Events.Duration FROM Events
              INNER JOIN CommitmentsStudent
              WHERE Events.EventID = (SELECT CommitmentsStudent.EventID WHERE CommitmentsStudent.StudentID = ?) AND Events.Date BETWEEN ? AND ? '''
            cur.execute(cmd, (currentuser.userid, str(sd), str(ed),))
    else:  # if teacher
        print(grouptype)
        if grouptype is not None:
            cmd = '''SELECT (SELECT Events_Type.Description FROM Events_Type WHERE Events_Type.TypeID= Events.TypeID),Events.Description,
                      (SELECT Rooms.RoomNameNumber FROM Rooms WHERE Rooms.RoomID = Events.RoomID)
                    ,Events.Date,Events.Time,Events.Duration FROM Events INNER JOIN CommitmentsTeacher
                    WHERE Events.EventID = (SELECT CommitmentsTeacher.EventID WHERE CommitmentsTeacher.TeacherID = ?) AND (Events.Date BETWEEN ? AND ?)
                    AND (Events.TypeID = ?)'''
            cur.execute(cmd, (currentuser.userid, str(sd), str(ed), grouptype,))

        else:
            cmd = '''SELECT (SELECT Events_Type.Description FROM Events_Type WHERE Events_Type.TypeID= Events.TypeID),Events.Description,
                      (SELECT Rooms.RoomNameNumber FROM Rooms WHERE Rooms.RoomID = Events.RoomID)
                     ,Events.Date,Events.Time,Events.Duration FROM Events INNER JOIN CommitmentsTeacher
                      WHERE Events.EventID = (SELECT CommitmentsTeacher.EventID WHERE CommitmentsTeacher.TeacherID = ?) AND Events.Date BETWEEN ? AND ?'''
            cur.execute(cmd, (currentuser.userid, str(sd), str(ed),))

    self.data = cur.fetchall()
    print(self.data)
    self.DataModel = QtGui.QStandardItemModel(self)
    self.tableView.setModel(self.DataModel)
    # self.load_data()
    __miscfuncitons__.load_data(self, self.data, "Event Type,Description,Location,Date,Time Start,Duration")
```

This method is comprised of several large SQL statements which are run depending on level of refinement the user has selected.

It starts by checking this by looking at a group of radio buttons which specify if a certain type of event is requested. Based on this it then executes one of 4 SQL statements, two for each user type (student or teacher). The parameters of this method are the results of the previous two methods and these specify the upper and lower bounds of the Events.Date column.

Once it has executed one of these statements with the appropriate fields completed the resulting data is sent to the tableView using the load_data _miscfunction_.

Each of these SQL statements includes inner joins to several tables as we need to translate the foreign keys into more usable data for display to the user. The use of inner joins allows us to more easily fetch related data from different tables. Without them we would have to run many more, smaller queries which would not be efficient

3.8 DETENTION FORM SCREEN

3.8.1 PROJECT DECOMPOSITION

3.8.1.1 CLASS REQUIREMENTS

This class is required to take information from a parent or super class and format it into a document that can be printed out and handed to a student or parent.

3.8.2 DEVELOPMENT AND ITERATIVE TESTING

3.8.3 FINAL CODE

3.8.3.1 __INIT__()

```
def __init__(self, _info_obj):
    QtGui.QMainWindow.__init__(self, _info_obj)
    self.setupUi(self)
    print("STARTED")
    print(_info_obj)
    self.STUDID = _info_obj["studentid"]
    self.duration = _info_obj["duration"]
    self.timestart = _info_obj["timestart"]
    self.date = _info_obj["date"]
    print(self.STUDID, self.date, self.duration, self.timestart)
    self.Print_Button.clicked.connect(self.gotoprint)
```

Here I setup the attributes for the class, by setting them equal to values stored in a dictionary passed as a parameter upon object creation. I also setup a listener for the button

3.8.3.2 GOTOPRINT()

```
def gotoprint(self):  # prints the form filling it out with data from the createEvent Screen

    ## dd/mm/yyyy format
    cmd = 'SELECT Forename FROM Students WHERE StudentID = ? '
    print(self.STUDID)
    cur.execute(cmd, (self.STUDID,))
    StudentData = cur.fetchall()
    StudentName = StudentData[1]
    self.Stud_Label.setText(StudentName)
    self.Date_Label.setText(self.date)
    cmd = '''SELECT (SELECT Years.Year FROM Years
            INNER JOIN Students
            WHERE Years.YearID == (SELECT Students.YearID WHERE StudentID = ?)
            ),(SELECT Form_Name FROM Forms
            INNER JOIN Students
            WHERE Forms.FormID == (SELECT Students.FormID WHERE StudentID = ?))
            '''
    cur.execute(cmd, (self.STUDID, self.STUDID))
    formname = cur.fetchall()
    formname = formname[0][0]
    yearname = formname[1][0]
    formgroup = yearname, formname
    self.Form_Label.setText(formgroup)
    self.Dur_Label.setText(self.duration)
    cmd = 'SELECT Username FROM Teachers WHERE TeacherID =?'
    cur.execute(cmd, (currentuser.userid,))
    tdata = cur.fetchall()
    tdata = tdata[0]

    self.Date_Label_2.setText(self.date)
    self.Start_Label.setText(self.timestart)
    self.Issue_Label.setText(tdata)
    self.Reason_Label.setText(self.description)
    printer = QtGui.QPrinter()
    dialog = QtGui.QPrintDialog(printer, self)
    if dialog.exec() != QtGui.QDialog.Accepted:
        return
    p = QtGui.QPixmap.grabWidget(self.ToPrint)
    printlabel = QtGui.QLabel()
    printlabel.setPixmap(p)
    painter = QtGui.QPainter(printer)
    printlabel.render(painter)
    painter.end()
```

Here I replace all the placeholder strings in the form with the correct data entered via the event creation screen. Once this is complete I use the PyQt printer function to send the form to the inbuilt windows printer.

3.9 ADVANCED ANALYTICS SCREEN

3.9.1 PROJECT DECOMPOSITION

3.9.1.1 CLASS REQUIREMENTS

This class is required to perform statistical analysis on the data collected through the use of the program

3.9.2 DEVELOPMENT AND ITERATIVE TESTING

3.9.3 FINAL CODE

3.9.3.1 __INIT__()

```
def __init__(self, parent=None):
    QtGui.QMainWindow.__init__(self, parent)
    self.setupUi(self)
    self.MMButton.clicked.connect(lambda: returntostart(self))
    self.BButton.clicked.connect(self.mainfunc)
    self.GEnter.clicked.connect(self.getgname)
    self.GName = ""
    self.GType = None
    self.EType = None
    self.TFrame = None
    self.CType = None
    self.StartDateQT = None
    self.StartDate = None
    self.EndDate = None
    self.StatsShown = None
    self.userdata = None
    self.comdata = None
    self.durdata = None
    self.alldata = None
    self.Model = None
    self.data = None
```

Here I setup the attributes as Nonetype and establish listeners for the various buttons on the form.

3.9.3.2 MAINFUNC()

```
def mainfunc(self):
    # grouptype
    if self.SRad.isChecked():
        self.GType = 0
    if self.FRad.isChecked():
        self.GType = 1
    if self.YGRad.isChecked():
        self.GType = 2
    if self.TRad.isChecked():
        self.GType = 3
    # EventType
    if self.IERad.isChecked():
        self.EType = 1
    if self.EERad.isChecked():
        self.EType = 2
    if self.DERad.isChecked():
        self.EType = 3
    if self.ARad.isChecked():
        self.EType = 4
    # TimeFrame
    if self.DRad.isChecked():
        self.TFrame = 0
    if self.WRad.isChecked():
        self.TFrame = 1
    if self.MRad.isChecked():
        self.TFrame = 2
    if self.YRad.isChecked():
        self.TFrame = 3
    # ChartType
    if self.PRad.isChecked():
        self.CType = 0
    if self.BRad.isChecked():
        self.CType = 1
    # StartDate
    self.StartDate = self.calendarWidget.selectedDate()
    self.StartDateQT = self.StartDate
    self.StartDate = self.StartDate.toPyDate()
    print(self.StartDate)
    # EndDate
    if self.TFrame == 0:
        self.EndDate = QtCore.QDate.addDays(self.StartDateQT, 1)
    if self.TFrame == 1:
        self.EndDate = QtCore.QDate.addDays(self.StartDateQT, 7)
    if self.TFrame == 2:
        self.EndDate = QtCore.QDate.addMonths(self.StartDateQT, 1)
    if self.TFrame == 3:
        self.EndDate = QtCore.QDate.addYears(self.StartDateQT, 1)
    self.EndDate = self.EndDate.toPyDate()
```

```
print(self.EndDate)
# StatsShown
if self.TMRad.isChecked():
    self.StatsShown = 0
else:
    self.StatsShown = 1
self.retdata()
```

Here I ingest all data inputted via the input boxes and radio buttons. The radio button input requires a lot of 'if' statements. This then leads on to the retdata() method

3.9.3.3 RETDATA()

```
def retdata(self):
    self.userdata = []
    if self.GType == 0:  ##if student
        namesplit = self.GName.split(" ")
        print(len(namesplit), namesplit)
        if len(namesplit) == 1:
            cmd = 'SELECT * FROM Students WHERE Username = ?'
            cur.execute(cmd, (self.GName,))
        elif len(namesplit) == 2:
            cmd = 'SELECT * FROM Students WHERE Forename = ? AND Surname = ?'
            cur.execute(cmd, (namesplit[0], namesplit[1],))
        self.userdata = cur.fetchall()
    if self.GType == 3:  # if teacher
        namesplit = self.GName.split(" ")
        if len(namesplit) == 1:
            cmd = 'SELECT * FROM Teachers WHERE Username = ?'
            cur.execute(cmd, (self.GName,))
        elif len(namesplit) == 2:
            cmd = 'SELECT * FROM Teachers WHERE Forename = ? AND Surname = ?'
            cur.execute(cmd, (namesplit[0], namesplit[1],))
        self.userdata = cur.fetchall()

    if self.GType == 1:  # if form
        s = self.GName
        match = regex.compile("[^\d]").search(s)
        namesplit = [s[:match.start()], s[match.start():]]
        cmd = 'SELECT * FROM Students WHERE Students.FormID =(SELECT Forms.FormID FROM Forms WHERE Forms.Form_Name = ? AND Forms.YearID = (SELECT Years.YearID FROM Years WHERE "Year" = ?))'
        cur.execute(cmd, (namesplit[1], namesplit[0],))
        self.userdata = cur.fetchall()

    if self.GType == 3:  # if year
        cmd = 'SELECT * FROM Students WHERE Students.YearID = (SELECT YearID FROM Years WHERE "Year" = ?)'
        cur.execute(cmd, (self.GName,))
        self.userdata = cur.fetchall()
    self.comdata = []
    tempdata = []
    for each in self.userdata:  # selects all commitments related to user
        if self.GType != 3:
            if self.EType != 4:
                cmd = "SELECT * FROM CommitmentsStudent WHERE StudentID = ? AND (SELECT TypeID FROM Events WHERE CommitmentsStudent.EventID = Events.EventID ) = ? AND (SELECT Events.Date " \
                      "FROM Events WHERE CommitmentsStudent.EventID = Events.EventID) BETWEEN ? AND ?"
                cur.execute(cmd, (each[0], self.EType, str(self.StartDate), str(self.EndDate),))
            else:
                cmd = 'SELECT * FROM CommitmentsStudent WHERE StudentID = ? AND (SELECT Events.Date FROM Events WHERE CommitmentsStudent.EventID = Events.EventID) BETWEEN ? AND ?'
                cur.execute(cmd, (each[0], str(self.StartDate), str(self.EndDate),))
            tempdata = cur.fetchall()
```

```
self.comdata.append(tempdata)
print(self.comdata)
self.comdata = self.comdata[0]
self.durdata = []
self.alldata = []
for each in self.comdata:
    cmd = 'SELECT Duration,Time FROM Events WHERE EventID = ?'
    cur.execute(cmd, (each[2],))
    tempfetch = cur.fetchall()
    self.alldata.append([tempfetch, each])
if self.StatsShown == 0:  # if stats on lesson time missed
    tottime = 0
    for each in self.alldata:
        tottime += int(each[0][0][0])
    print(tottime)
    totevents = len(self.alldata)
    d = 0
    ie = 0
    ee = 0
    print(self.alldata)
    for i in self.alldata:
        print("I", i)
        cmd = 'SELECT TypeID FROM Events WHERE EventID = ?'
        cur.execute(cmd, (i[1][2],))
        temp = cur.fetchall()
        print("TEMPSS", temp)
        if temp[0][0] == 1:
            ie += 1
        if temp[0][0] == 2:
            ee += 1
        if temp[0][0] == 3:
            d += 1

    modeldata = [self.GName, tottime, totevents, d, ie, ee]
    self.data = [modeldata]
    print(self.data)
    __miscfuncitons__.load_data(self, self.data,
                                "Group Name,Total Time,Total Events,Detenions,Internal Events,External Events")
```

```
if self.StatsShown == 1:  # Event Attendance Data
    attended = 0
    missed = 0
    print("ALLDATA::", self.alldata)
    for each in self.alldata:
        if each[1][3] != "None":
            if each[1][3] == 0:
                missed += 1
            elif each[1][3] == 1:
                attended += 1
    total = attended+missed
    self.data = [total,attended,missed]
    print(attended, missed)
    __miscfuncitons__.load_data(self, self.data,"Total Events,Attended,Missed")
    self.webView.settings().setAttribute(QWebSettings.LocalContentCanAccessRemoteUrls, True)
    html = """
    <html>
      <head>
        <script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
        <script type="text/javascript">
          google.charts.load('current', {'packages':['corechart']});
          google.charts.setOnLoadCallback(drawChart);

          function drawChart() {

            var data = google.visualization.arrayToDataTable([
              ['%q', '%w'],
              ['Attended',     4],
              ['Missed',      5],
            ]);

            var options = {
              title: 'Attendance Data'
            };

            var chart = new google.visualization.PieChart(document.getElementById('piechart'));

            chart.draw(data, options);
          }
        </script>
      </head>
      <body>
        <div id="piechart" style="width: 900px; height: 500px;"></div>
      </body>
    </html>
    """

    html = html.replace("%q", "Events")
```

```
    html = html.replace("%w", "Attened/Not Attended")
    html = html.replace("%a", str(attended))
    html = html.replace("%m", str(missed))
    self.webView.setHtml(html)

# self.alldata =  [   [   [(duration,timestart)]  ,(commitmentid,studentid,eventid,attended?)]  ,...]  <----template layout

print(self.alldata, "alldta")
```

In this method I use various SQL queries to fetch data which I then operate on to find statistics. These calculated statistics are then displayed in a table and pie chart format. The pie chart runs inside of a PyWebView and is constructed using an HTML script.

3.10 MODIFY USERS

3.10.1 PROJECT DECOMPOSITION

3.10.1.1 CLASS REQUIREMENTS

This class will need to allow administrative users to change all details of a user within the program. It should also allow non-administrative users to change their own passwords. It should keep all confidential data secret and rehash passwords if they are changed

As such, this function will have its own splash screen which allows the user to navigate to screens which will allow them to either change their own password or edit other users' data

3.10.2 SPLASH SCREEN

This screen will direct the user to one of the next two screens it will have three buttons and must not allow non-administrative users to access the 'Edit Users Screen'

3.10.3 DEVELOPMENT AND ITERATIVE TESTING

3.10.3.1 FINAL CODE

3.10.3.1.1 __INIT__()

```
def __init__(self, parent=None):
    QtGui.QMainWindow.__init__(self, parent)
    self.setupUi(self)
    self.MMButton.clicked.connect(lambda: returntostart(self))
    self.CMPButton.clicked.connect(self.cpfunc)
    self.CUADButton.clicked.connect(self.cuadfunc)
    self.ChangePasswordWindow = None
    self.ChangeUserDataWindow = None
```

this class sets up the attributes of the class as Nonetype and sets up listeners for the buttons

3.10.3.1.2 CPFUNC()

```
def cpfunc(self):
    self.ChangePasswordWindow = ChangePasswordScreen()
    self.ChangePasswordWindow.show()
    self.close()
```

simply a constructor method for the 'ChangePasswordScreen' object, is shows the new object and closes itself

3.10.3.1.3 CUADFUNC()

```
def cuadfunc(self):
    self.ChangeUserDataWindow = EditUsersScreen()
    self.ChangeUserDataWindow.show()
    self.close()
```

another constructor method. This creates an instance of the 'EditUsersScreen' class, shows it and closes itself

3.10.4 CHANGE PASSWORD SCREEN

3.10.5 DEVELOPMENT AND ITERATIVE TESTING

3.10.5.1 FINAL CODE

3.10.5.1.1 __INIT__()

```
def __init__(self, parent=None):
    QtGui.QMainWindow.__init__(self, parent)
    self.setupUi(self)
    self.MMButton.clicked.connect(lambda: returntostart(self))
    self.CPButton.clicked.connect(self.cpmethod)
```

this method sets up the ui of the window and establishes listeners for button presses, no attributes are used to keep the security of the new password intact.

3.10.5.1.2 CPMETHOD()

```
def cpmethod(self):
    currentpassword = self.CurPassField.text()
    confirmcurrentpassword = self.ConCurPassField.text()
    if currentpassword != confirmcurrentpassword:
        self.ErrorLabel.setText("Current Passwords Do not Match Try again")
        pass
    newpassword = self.NewPassField.text()

    confirmnewapassword = self.ConNewPassField.text()
    if newpassword != confirmnewapassword:
        self.ErrorLabel.setText("New Passwords do not match Try again")
    else:
        hashedpass = __miscfuncitons__.hashpass(newpassword)
        if currentuser.usertype == "Student":
            cmd = 'UPDATE Students SET Password = ? WHERE StudentID = ?'

        else:
            cmd = 'UPDATE Teachers SET Password = ? WHERE TeacherID = ?'
        cur.execute(cmd, (str(hashedpass), currentuser.userid,))
        con.commit()
        returntostart(self)
```

this method takes in the user's entry of their old password and a confirmation entry. If they match it returns an error screen

However, if they do match it takes input of the new password and it's confirmation entry, again checking if they match.

If they do it hashes the password using the hashpass _miscfunction_ and runs a SQL query to amend the stored password depending on if the user is a teacher or student

3.10.6 EDIT USERS SCREEN

This screen needs to be only accessible by admin users. It must keep user data confidential as far as possible and hash any changes to passwords.

3.10.7 DEVELOPMENT AND ITERATIVE TESTING

3.10.7.1 FINAL CODE

3.10.7.1.1 __INIT__()

```
def __init__(self, parent=None):
    QtGui.QMainWindow.__init__(self, parent)
    self.setupUi(self)
    self.MMButton.clicked.connect(lambda: returntostart(self))
    self.SearchButton.clicked.connect(self.searchforuser)
    self.CommitButton.clicked.connect(self.upd_record)
    self.data = []
    self.tabletype = None
    self.tdata = []
    self.DataModel = None
    self.colcount = None
    self.updata = None
```

This method sets up the attributes of the class as Nonetype or empty arrays and sets up listeners for the buttons

3.10.7.1.2 SEARCHFORUSER()

```
def searchforuser(self):
    username = self.UserInput.text()

    if username == "" or None or len(username.split(" ")) > 2:
        QtGui.QMessageBox.information(self, "Invalid Username entered", "Invalid Username Entered")

    try:
        searchdata = username.split(' ")
        cmd = 'SELECT * FROM Students WHERE Forename = ? AND Surname = ? '
        cur.execute(cmd, (searchdata[0], searchdata[1],))
        self.data = cur.fetchall()
        self.tabletype = 0
        if not self.data:
            cmd = 'SELECT * FROM Teachers WHERE Forename =? AND Surname = ?'
            cur.execute(cmd, (searchdata[0], searchdata[1],))
            self.data = cur.fetchall()
            self.tabletype = 1
    except:
        cmd = 'SELECT * FROM Students WHERE Username = ?'
        cur.execute(cmd, (username,))
        self.data = cur.fetchall()
        self.tabletype = 0
        if not self.data:
            cmd = 'SELECT * FROM Teachers WHERE Username = ?'
            cur.execute(cmd, (username,))
            self.data = cur.fetchall()
            self.tabletype = 1
    print(self.data)
    if self.tabletype == 1:
        self.tdata = list(self.data[0])
        self.tdata[4] = "******"
    if self.tabletype == 0:
        self.tdata = list(self.data[0])
        self.tdata[9] = "******"
    self.data = []
    self.data.append(self.tdata)

    self.DataModel = QtGui.QStandardItemModel(self)
    self.tableView.setModel(self.DataModel)
    _miscfuncitons_.load_data(self, self.data, "")
```

this method is run when the ‘Search’ button is pressed.

It takes input from the search field and searches the database for a user matching those details. It checks for both students and teachers and returns an error if nothing is found.

It does this by using a SQL query with parameters shown by ‘?’ . I then replace these with correct parameters in the correct format (string, integer etc.)

Once it has all the data on the user it prepares it for display. It does this by changing the password item to a string of asterisks, the location in the array depends on if they are a student or teacher so a attributes called self.tabletype is set at data collection and is used in this process.

Once the data is prepped the data is sent to the tabelView using the load_data() _miscfunction_

3.10.7.1.3 UPD_RECORD()

```
def upd_record(self):
    self.updata = []
    for i in range(self.colcount):
        temp = self.DataModel.data(self.DataModel.index(0, i))
        self.updata.append(temp)
    if self.updata == self.data:
        QtGui.QMessageBox.information(self, "ERROR", "No Changes Made")

    newhashpass = __miscfuncitons__.hashpass(self.updata[-2])
    print(newhashpass)
    self.updata[-2] = newhashpass
    print(self.updata)

    if self.tabletype == 0:
        cmd = '''
                UPDATE Students
                SET Forename = ?,
                    Other_Names = ?,
                    Surname = ?,
                    YearID = ?,
                    FormID = ?,
                    Gender = ?,
                    Date_of_Birth = ?,
                    Username = ?,
                    Password = ?,
                    User_Level= ?
                WHERE StudentID = ?
                        '''
        cur.execute(cmd, (
            self.updata[1], self.updata[2], self.updata[3], self.updata[4], self.updata[5], self.updata[6],
            self.updata[7], self.updata[8], self.updata[9], self.updata[10], self.updata[0],))
        con.commit()

    if self.tabletype == 1:
        cmd = '''
                UPDATE Teachers
                SET Forename = ?,
                    Surname = ?,
                    Username = ?,
                    Password = ?,
                    User_Level = ?
                WHERE TeacherID = ?
        '''
        cur.execute(cmd, (
            self.updata[1], self.updata[2], self.updata[3], self.updata[4], self.updata[5], self.updata[0],))
        con.commit()
```

once the user has found the user they have required they can use the tableView to change data. If they have made all the changes they require they hit the 'Confirm' button and this script will run.

It starts by converting the data in the table back into a list that python can understand. It also checks and if there are no changes it notifies the user through an error box.

It will rehash the password stored in the list before updating the record in the DB with the new information.

There are two different SQL update queries to accomadate for changing both student and teacher data

Once it has committed these changes the script stops

3.11 EVENT REGISTRATION

3.11.1 PROJECT DECOMPOSITION

3.11.1.1 CLASS REQUIREMENTS

3.11.2 DEVELOPMENT AND ITERATIVE TESTING

3.11.3 FINAL CODE

3.11.3.1 __INIT__()

```
def __init__(self, parent=None):
    QtGui.QMainWindow.__init__(self, parent)
    self.setupUi(self)
    self.MMButton.clicked.connect(lambda: returntostart(self))
    self.data = []
    self.updata = []
    self.DataModel = None
    self.flag = True
    self.eid = None

    self.selectevent()

    self.SubButton.clicked.connect(self.registration)
```

Here I set up the attributes of the class as Nonetype, setup the GUI for the screen and set up the button listeners

3.11.3.2 SELECTEVENT()

```
def selectevent(self):
    currentdate = datetime.datetime.today().date()
    currenttime = datetime.datetime.now().time()
    cmd = '''SELECT * FROM Events e INNER JOIN CommitmentsTeacher t ON e.EventID = t.EventID WHERE t.TeacherID = ? AND e.Date = ?'''
    ## TEST Returned event times
    cur.execute(cmd, (currentuser.userid, currentdate,))
    cureventdata = cur.fetchall()
    print(cureventdata,"AVANT")
    for event in cureventdata:
        tlb = datetime.datetime.strptime(event[4], '%H:%M:%S')
        tub = tlb + timedelta(minutes=int(event[6]))
        tlb = tlb.time()
        tub = tub.time()
        if not __miscfuncitons__.time_in_range(tlb, tub, currenttime):
            cureventdata.remove(event)
        else:
            pass
    cmd = '''SELECT s.StudentID, s.Forename, s.Surname, y.Year, f.Form_Name, c."Attended?" FROM CommitmentsStudent c INNER JOIN Students s ON c.StudentID = s.StudentID
             INNER JOIN Years y ON s.YearID = y.YearID INNER JOIN Forms f ON s.FormID = f.FormID WHERE c.EventID = ? '''
    print(cureventdata,"APRES")
    cur.execute(cmd, (cureventdata[0][9],))
    studentdata = cur.fetchall()
    self.eid = cureventdata[0][9]
    print(studentdata, "STUDENTDATA")
    for student in studentdata:
        listappend= []
        for item in student:
            listappend.append(item)
        self.data.append(listappend)

    for i in range(len(self.data)):
        if self.data[i][5] == 1:
            self.data[i][5] = "/"
        elif self.data[i][5] == 0:
            self.data[i][5] = "\\"
        else:
            pass

    __miscfuncitons__.load_data(self, self.data,
                                "ID,Forename,Surname,Year,Form,Attended? ('/'Yes / '\\' No )")
```

This runs automatically after the initialization method. It searches the database for events the current teacher is registered to attend. It then sorts them until it finds the one (or none) that are currently occurring. From there it selects all student participants for that event and displays them along with their registration status in the table. It converts the Boolean values from the database to a ‘/’ or ‘\’

3.11.3.3 REGISTRATION()

```
def registration(self):
    self.updata = []
    for z in range(len(self.data)):
        temprow = []
        for i in range(len(self.data[0])):
            temp = self.DataModel.data(self.DataModel.index(z, i))
            temprow.append(temp)
        self.updata.append(temprow)
    for events in self.data:
        if self.updata == events:
            QtGui.QMessageBox.information(self, "ERROR", "No Changes Made")
    else:
        print(self.updata, "UPDATA")
        for each in self.updata:
            print(each[5], "EACH5")
            if each[5] == "/" or each[5] == "\\":
                print("TRUE")
                if each[5] == "/":
                    each[5] = 1
                else:
                    each[5] = 0
                print(each[5],"E5")
                cmd = '''UPDATE CommitmentsStudent
                  SET "Attended?" =? WHERE StudentID = ? and EventID = ?
                '''
                print(each[5],each[1],self.eid)
                cur.execute(cmd, (each[5], each[0], self.eid, ))
                con.commit()
            else:
                error_string = "ERROR With Value for user %s please enter '/' for present or '\\' for absent"
                error_string = error_string.replace("%s", str(each[1:3]))
                QtGui.QMessageBox.information(self, "ERROR", error_string)
                break
```

This method it triggered when the teacher submits their register by pressing a button. It re-converts the '/' and '\' to Boolean values and then adds it to the correct database records before committing the changes. If a student hasn't been registered it notifies the teacher with an error box.

3.12 MISCELLENIOUS FUNCTIONS

3.12.1 VERIFYHASH()

```
def verify_hash(self, userpass, storedpass):
    from PyQt4 import QtGui
    import hashlib
    password = None
    salt = None
    try:
        password, salt = storedpass.split(':')
    except:
        QtGui.QMessageBox.information(self, "Password Error", "Error with Password")
    try:
        data = [password, hashlib.sha256(salt.encode() + userpass.encode()).hexdigest()]
        return data[0] == data[1]
    except:
        pass
```

this procedure is used for logging in and when changing your password. It takes two main parameters. The stored password (hashed) and the entered password (unhashed). It hashes the entered password into the same format as the stored password and returns a Boolean value depending on whether they match. Here you can see that QtGui and hashlib have both been locally imported as needed

3.12.2 HASHPASS()

```
def hashpass(password):
    import uuid
    import hashlib
    salt = uuid.uuid4().hex
    return hashlib.sha256(salt.encode() + password.encode()).hexdigest() + ":" + salt
```

this function is used when creating a new user or forcing a password change. It takes a unhashed string and returns it hashed in the correct format to be stored. Here you can see that both uuid and hashlib have been imported locally as needed

3.12.3 LOGOUT()

```
def logout(obj, inwindow, user):
    window = inwindow
    user.userid = None
    user.acctype = None
    user.usertype = None
    window.User_Field.setText("")
    window.Pass_Field.setText("")
    window.Output_Label.setText("")
    window.show()
    obj.close()
```

This is a procedure which is used when the user wants to log out. It must clear all data related to the previous session including any details stored in the user object. It also readies the login screen for use and closes the splash screen where this function is invoked

3.12.4 LOAD_DATA()

```
def load_data(obj, data, labels):

    """Loads data into a table given the containing object,
    list of data to load (one level deep) and a string of
    labels separated by a comma"""

    from PyQt4 import QtGui
    if len(data) > 0:
        for i in range(len(obj.data) - 1, -1):
            data.pop(i)
    obj.DataModel = QtGui.QStandardItemModel(obj)
    obj.tableView.setModel(obj.DataModel)
    for row in data:
        items = [
            QtGui.QStandardItem(str(field))
            for field in row
            ]
        obj.DataModel.appendRow(items)
    obj.DataModel.setHorizontalHeaderLabels(labels.split(","))
    # manual setting of column headings
```

This function is used whenever data is needed to be loaded into a PyQt tableView widget. It takes three parameters:

- Obj
  - The object that houses the currently used tableView
- Data
  - The python list of data that needs to be loaded into the table
- Labels
  - A sting of column headings each separated by a comma

3.12.5 CHECKDATA()

```
def checkdata(u, p, cur):
    import re as regex

    def checkpassword():
        len_error = len(p) >= 8
        cont_digit = regex.search(r"\d", p) is None
        cont_caps = regex.search(r"[A-Z]", p) is None
        cont_lcase = regex.search(r"[a-z]", p) is None
        cont_symbol = regex.search(r"\W", p) is None
        overall = not (len_error, cont_digit, cont_caps, cont_lcase, cont_symbol)
        return {
            "Overall": overall,
            "Length Less than 8": len_error,
            "Contains Digits": cont_digit,
            "Contains Caps": cont_caps,
            "Contains Lowercase": cont_lcase,
            "Contains Symbols (!,?,#, etc.)": cont_symbol
        }

    passwordresults = checkpassword()

    def checkusername():
        if len(u) > 5:
            cmd = 'SELECT StudentID FROM Students WHERE Username = ?'
            cur.execute(cmd, (u,))
            ucheckdata = cur.fetchall()
            return ucheckdata
        else:
            return False

    usernameresults = checkusername()
    return passwordresults, usernameresults
```

This is an example of one of my uses of functions as it returns a value rather than outputting to the user. It imports regex locally as it is necessary. It uses regex to check the password for certain characters including digits and symbols, to ascertain if the password is strong enough for use. It returns true or false depending on if the username and password meet the criteria.

4 EVALUATION

4.1 TESTING SUCCESS CRITERIA

4.1.1 TEST PLAN

Upon the completion of each class (screen), I will test that it meets the success criteria related to that screen (see Success Criteria). For the most part I will simply run the program and provide it with acceptable data, to test the functionality of the data, before performing destructive testing with illegal data entry. If the program fails either of the tests I will then have to go back to the code to find the source of the error and fix it. Any fixes I perform will be documented with the issue and fix I used to solve it.

4.1.1.1.1 PROGRAM INITIALISATION

Success Criteria Number(s)	Test	Destructive Test	Required Result
1	Run Program	Move/delete a UI file	Program starts correctly or notifies the user that a UI file cannot be found
2	Run Program	Move/delete the Database file	Program starts correctly or notifies the user that a DB file cannot be found
3	Run Program	Removes needed Library File	Program starts correctly or notifies the user that an external library is missing
4	n/a	Entry of illegal data in every field	Program doesn't crash and tells user that data isn't of the correct format

5	Completed code should feature classes and methods	n/a	GUI forms should be generated using class based objects
6	All screens accessed via related buttons	Buttons pressed in quick succession	Program should open all forms when required. Only one form should be open at one time. Program should not crash upon multiple button presses

4.1.1.1.2 LOGIN SCREEN

7	Test data entered into the form	Illegal data entered into the form	Program should allow entry of both (at this stage)
8	Test data Password entered into the password field	n/a	Password field should not display actual characters, rather dots or asterisks
9/10	Have program print passwords that are being compared for login	n/a	Printed values should not be in plaintext format, should be in a hashed format
11	Log in using student and teacher test data	n/a	Program should login and print usertype to console for verification
12	Log in using student and teacher testdata	n/a	Print out User object attributes after attempted creation
13	n/a	Attempt to login using incorrect Username	Program should return an error message telling the user that the password is username
14	n/a	Attempt to login using incorrect password	Program should return an error message telling the

			user that the password is incorrect
15	Attempt to login using student and teacher test data	n/a	Program should tell the user that their details are correct and open the splash screen
16	Login using student and teacher test data	n/a	Print out sql queries to be executed and results. Proof of this working can be seen from the successful login tested above
17			Proof of this working can be seen from the successful login tested above
18	Press the cancel button on the form	n/a	Program should exit to desktop

4.1.1.1.3 SLASH SCREEN

19	Login to show the form	n/a	Program should display the form as designed
20,21,22,23	Whist logged in using the test student account, attempt to access screen	n/a	Program should tell the user that they are not able to access this feature
24	Whilst logged in, press the logout button	n/a	Program should close the splash screen and return the user to the login screen, with all fields blank
25	After logging out, print the user attributes	n/a	They should all hold None values
26	Whilst logged in, press the exit button	n/a	Program should exit to the desktop

27,28,29,30	Whilst logged in using the teacher test user, attempt to access the screen	n/a	Program should open the screens and close the splash screen
31,32	Whilst logged in, attempt to access the screen	n/a	Program should open the screen and close the splash screen

4.1.1.1.4 CREATE NEW USER SCREEN

33,34	Fill out all fields with valid data, press the confirm button	Leave some fields blank	the program should run as normal if all fields hold values. If any are empty the program should return an error message to the user
35	Fill out all data for student user creation and press the confirm button	n/a	The program should enter all data into the database in a fully normalized format
36	Fill out all data for teacher user creation and press the confirm button		The program should enter all data into the database in a fully normalized format
37	Fill out all fields with relevant data	Fill fields with illegal data	Program should run if all data entered meets the requirements, otherwise it should return an error to the user.
38	By looking at the form you should be able to tell which fields much be filled out for different user creation	n/a	A group of test users should easily be able to tell, respond to feedback if necessary
39,40,41	Create test users	Create test user with incorrect names for forms etc.	If all data can be found in the database a new user with the correct foreign keys

			should be created. Else the user should be notified that a name doesn't check out

4.1.1.1.5 CREATE NEW EVENT SCREEN

42,43	Create event with all fields filled out	Create event with some empty fields	If all fields are filled the program should continue. Otherwise it should alert the user to any empty fields
44,45,46	Attempt to create an event of the three different types	N/A	Program should take in all input and create a new fully normalized DB entry
47,48	Using valid event data Print values to entered into DB	Using illegal event data. Print all values to be entered into the DB	All values should be primary or foreign keys. Otherwise it should return an error that a something doesn't exist in the DB
49	Press the return button	n/a	Program should close current window and display the splash screen
50	Create event with valid time frame	Create event with invalid time frame	Program should register whether the event being created runs between accepted time frames (break, lunch, after school) depending on event type. Returning an error if the event is outside of these times
51	Create event with free room	Create event with known booked room	Program should return an error if the room is already booked. Otherwise it

			should continue with the process
52	Create event with free users	Create an event with known double booked users	If there is a double booked user: Program should assess level of importance of prior engagement and overwrite if possible. Otherwise it should alert the user to the error Otherwise it should add the user to the event

4.1.1.1.6 SEE PERSONAL EVENTS SCREEN

53	Use the calendar widget to enter a date. Program prints it	n/a	Program should print the date in a valid format
54	Use the radio buttons to select a time frame	n/a	Print the name of which radio button is selected
55	Press the next or previous buttons once events have been initialized	n/a	Program should display different events depending on the current time range selected
56	Use radio button to select event type	n/a	Program should only allow one button to be selected at once. Program should then only display events of the selected type
57	Click the return button	n/a	Program closes current window and shows the splash screen
58	Initialize data	n/a	Program should not show any foreign keys, instead

			displaying their related secondary keys
59	Initialize data	n/a	All events shown should be relevant to the user. Can be checked against the commitmentstudent DB table

4.1.1.1.7 ADVANCED ANALYTICS SCREEN

60	Analyse data specifying different group types	n/a	Program should output data relevant to each group
61,62,63	Enter group name	Enter invalid group name	Program should show an error message if invalid group name is inputted. Otherwise it should search the corresponding database table for the group, returning the resulting data to the program
64,65,66	Instruct the program to show attendance data	n/a	Program should go through all fetched data and find the attendance ratio of the data set. It should then return these results with the data set to the table and pie chart

4.1.1.1.8 MODIFY USER SCREEN

67	Attempt to change password as a student user	n/a	Program should check confirmation passwords to match and current password to be correct before

			entering it into the DB
68	Attempt to change password das a teacher user	n/a	Program should check confirmation passwords to match and current password to be correct before entering it into the DB
69,70	Search for user	Search for non existent data	program should search for user and display all data, with all foreign keys converted, in the table. If the user cannot be found it should alert the user
71	Change values and hit confirm button	No changes made	Program should return an error if no changes are made. Otherwise it should change the database entry with new *fully normalized* values

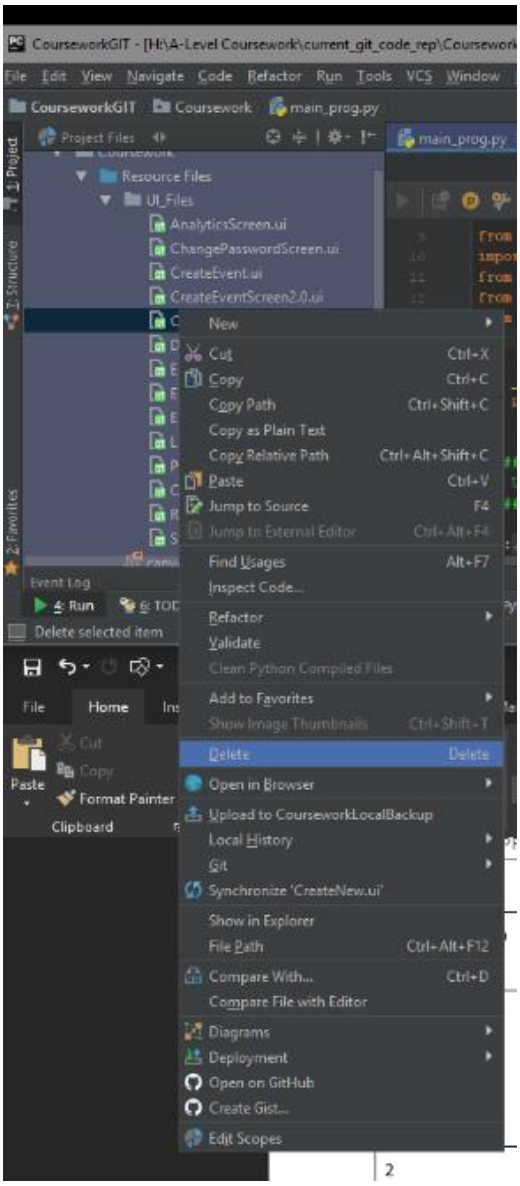
4.1.1.1.9 REGISTRATION SCREEN

72,73,74	Open the window	n/a	Program should show the details of all enrolled users. Should only display attendees of current event
75	Press confirm button after completing register	n/a	Program should check to see if the register is complete before entering it into the DB. Return an error if students haven't been registered

4.1.2 TESTING

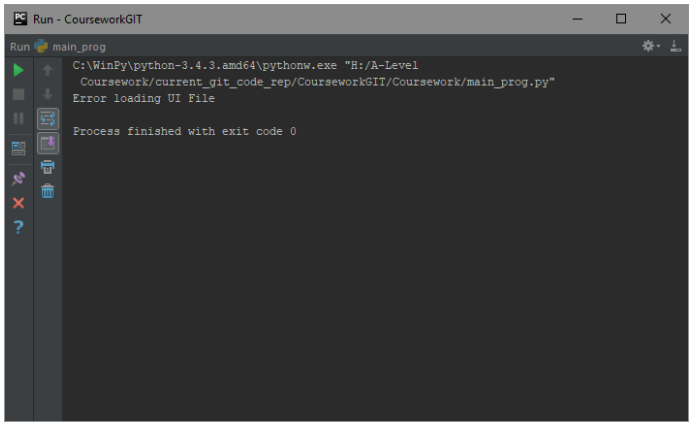
4.1.2.1 PROGRAM INITIALISATION/ GENERAL

4.1.2.1.1 CRITERION 1



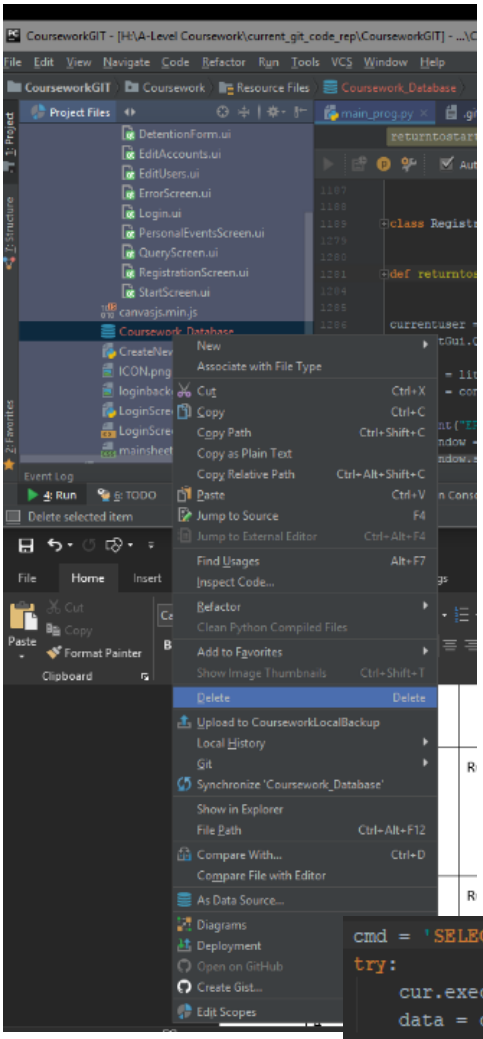
by deleing a ui file the program should alert the user to an error instead of just crashing

<u>result:</u>



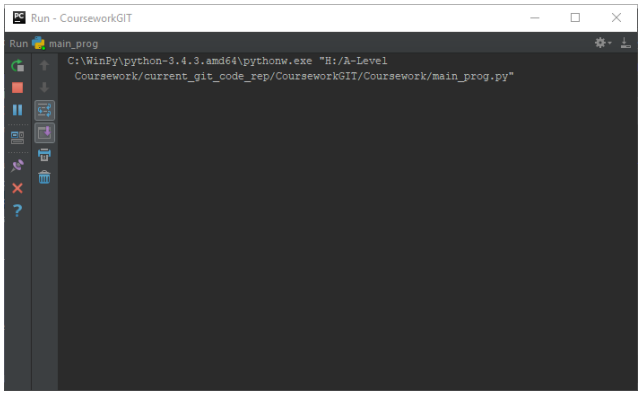
This is the required result the program has fulfilled this criterion

4.1.2.1.2 CRITERION 2



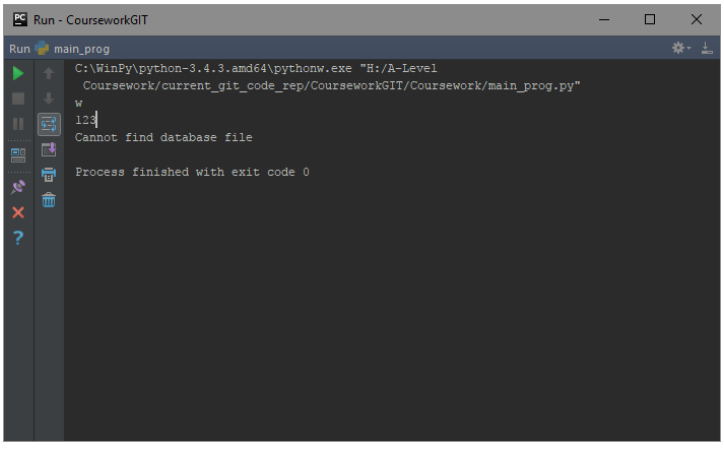
by deleting the database file, upon running the program I should receive an error and the program should exit rather than it crashing with an error code

result



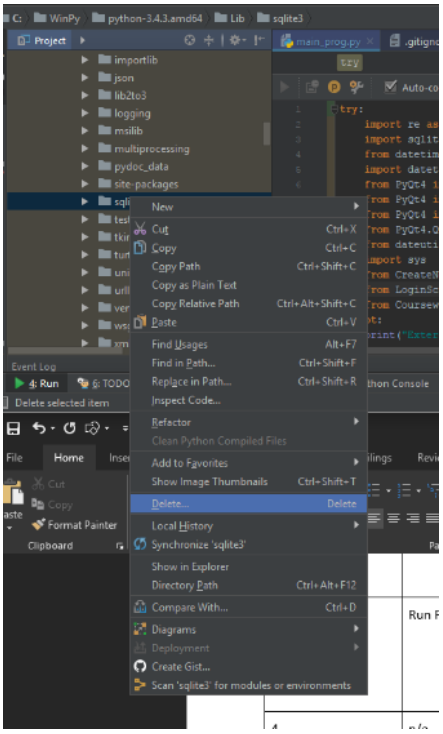
This is not the required result and it is due to python generating a blank database file when it cannot find the one it is looking for. To fix this I added a try/except around the first query. The result of this change is:

```
cmd = 'SELECT TeacherID,User_Level,Password FROM Teachers WHERE username =?'
try:
    cur.execute(cmd, (username,))
    data = cur.fetchall()
except:
    print("Cannot find database file")
    exit(0)
```



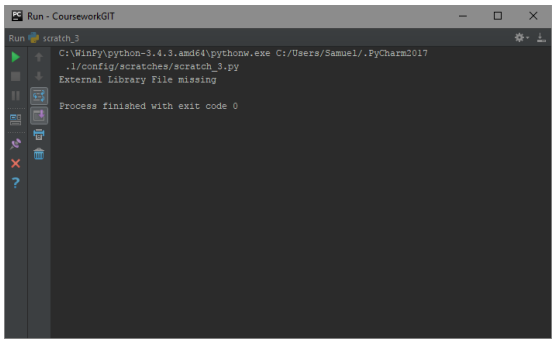
The code has now passed this test after initially failing it.

4.1.2.1.3 CRITERION 3



by deleting the sqlite3 external library I will test criterion 3.

The result of this is:



The program has fulfilled criterion 3

4.1.2.1.4 CRITERION 4

Will be tested within each sub program

4.1.2.1.5 CRITERION 5

As you can see from the Final Code, here, the program features classes and methods, denoted by 'class' and 'def' respectively

4.1.2.1.6 CRITERION 6

Will be tested within each sub program

4.1.2.2 LOGIN SCREEN

4.1.2.2.1 CRITERIA 7,8,9,10,11,12

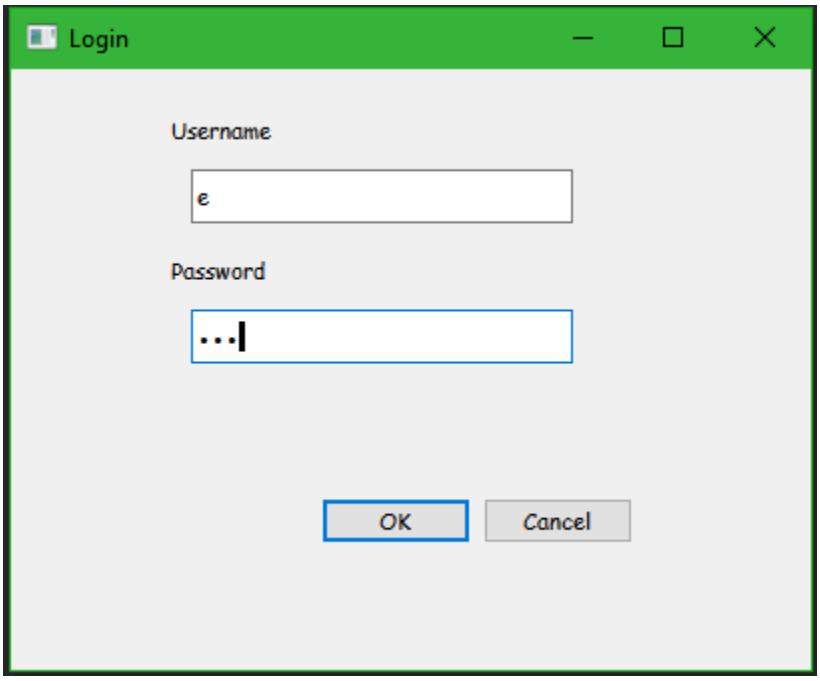


Figure 1

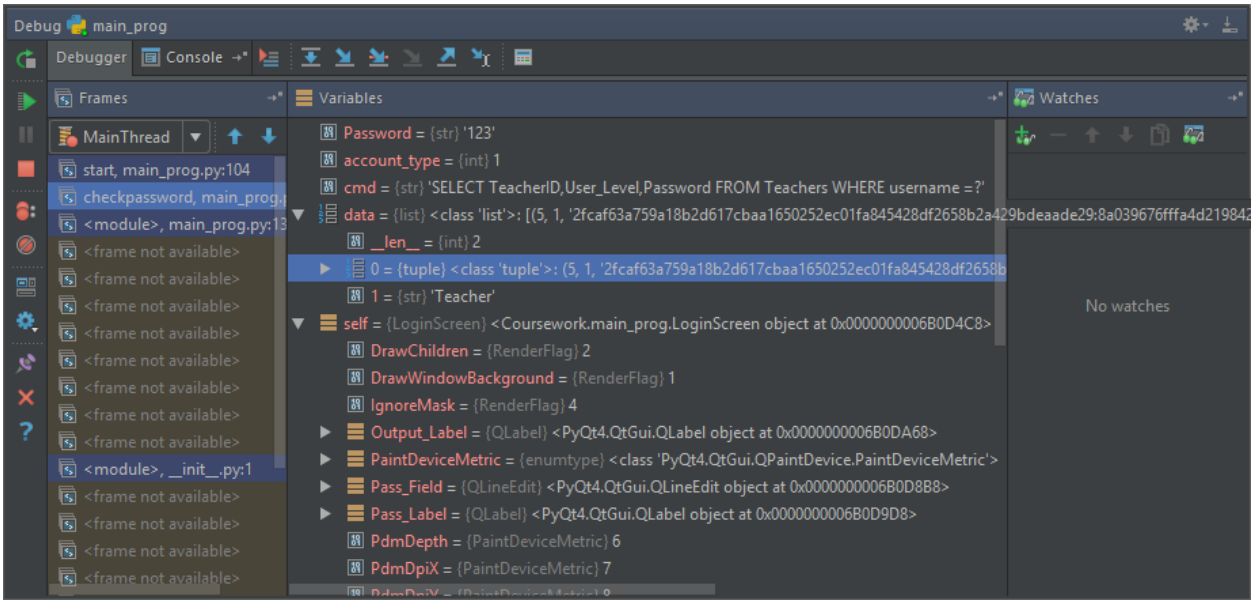
```
related teacher data: []
related student data [(7, 0, 'cfbaa60a71942fcdc9d08ffdd3a809dc8daf6f89427049c372260c7df955ab35:9997c6200a3e4ca4bba10c263fcdef89')]
cfbaa60a71942fcdc9d08ffdd3a809dc8daf6f89427049c372260c7df955ab35 dd 9997c6200a3e4ca4bba10c263fcdef89 ""dddddd
PASSWORD CORRECT
7 Student 0
```

Figure 2

The input seen in **Figure 1** produces the output shown in **Figure 2.** this data is the Student Test data, defined in the design stage.

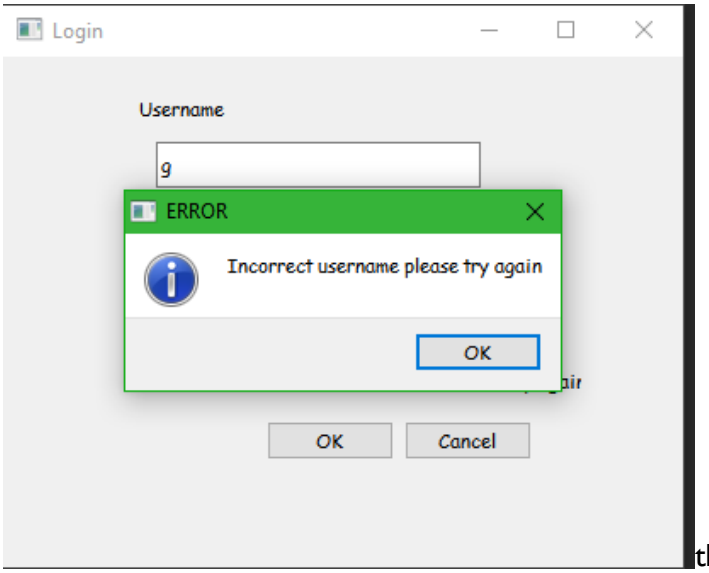
This has passed all the tests for these criteria

Using the build in debugger in my IDE I am able to set a breakpoint as just before this form closes and run the program. Upon entering the correct details the program ends and I am able to see the current values of all the variables currently loaded in memory

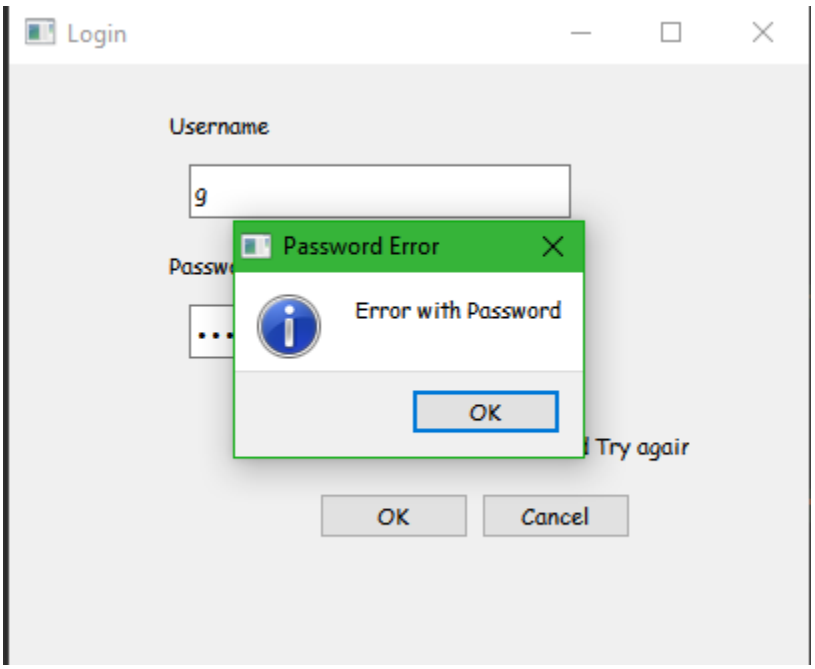


Using this view I can see the data fetched from the DB as well as data entered and hashed by the program. This data is saved to a file for use later if necessary

4.1.2.2.2 CRITERIA 13,14



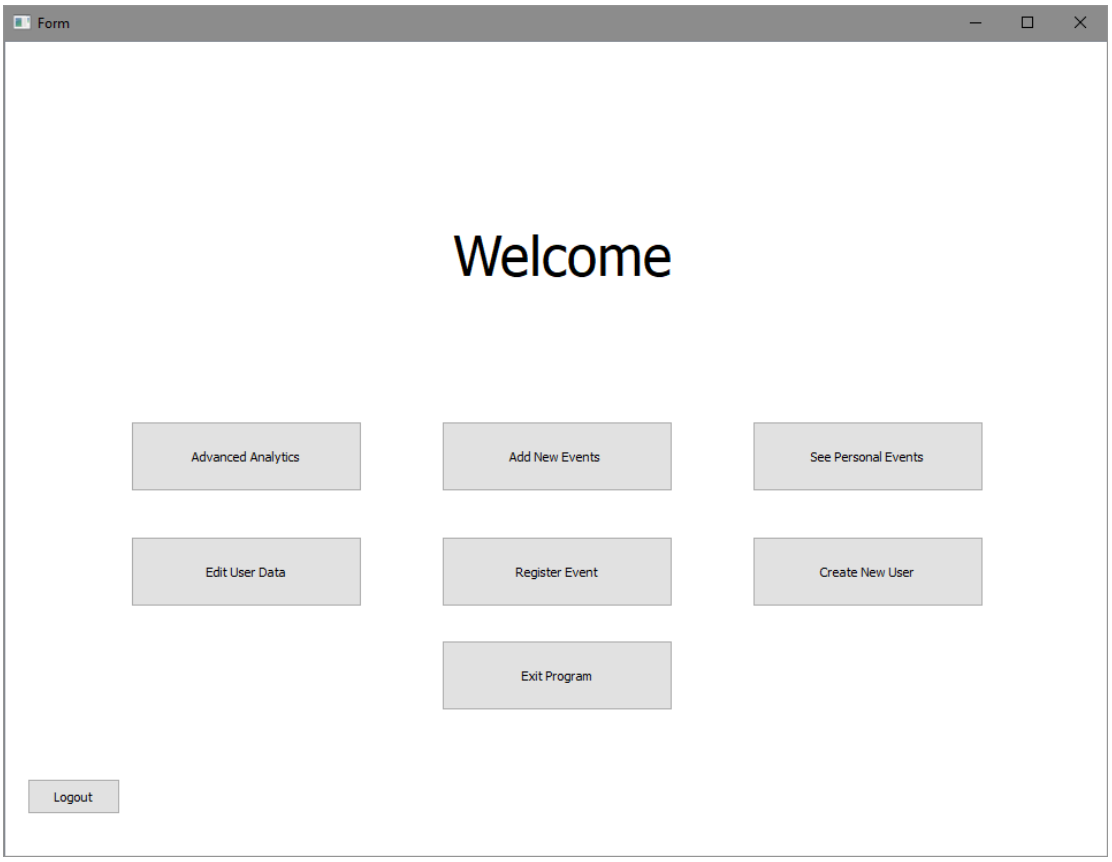
the code has passed criterion 13 as it detects incorrect usernames and alerts the user

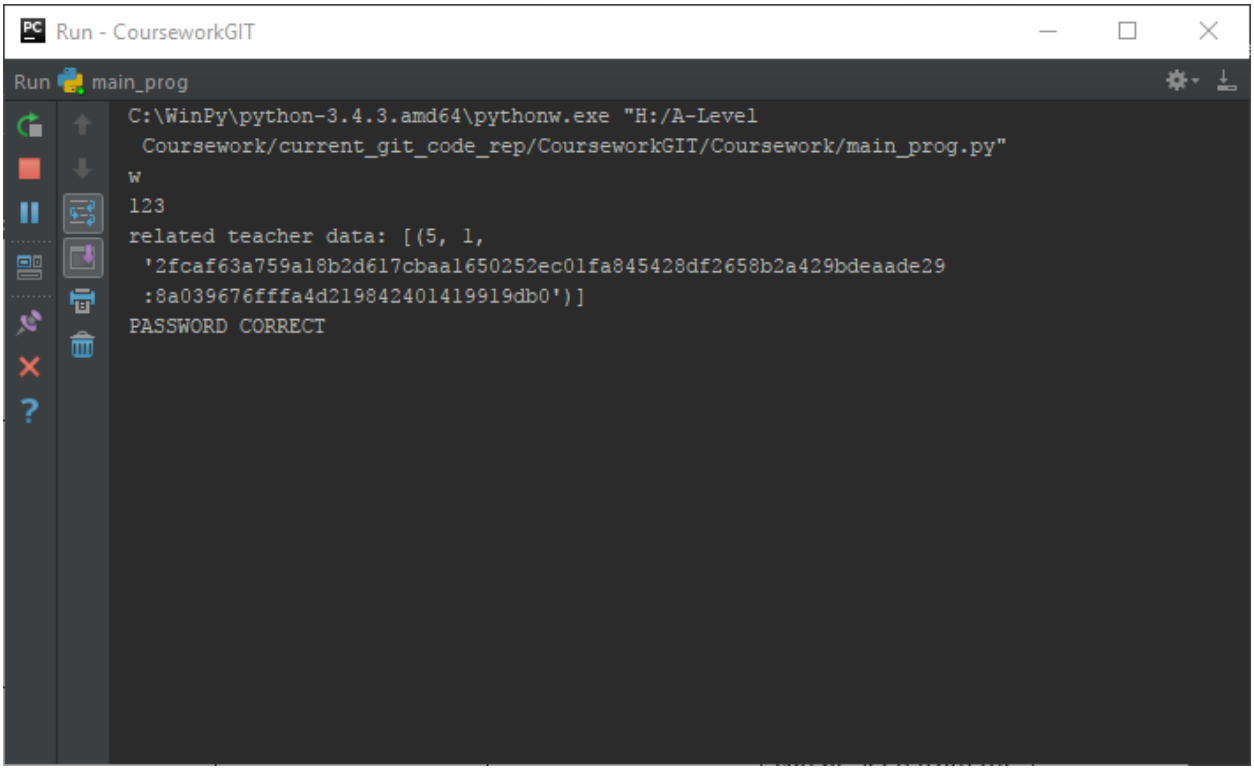


The code has passed criterion 14

4.1.2.2.3 CRITERIA 15,16,17,19

Using the student test data to test both cases as the code to open the splash screen is the same for both, the following result is produced





It has passed these criteria

I can also use this code to log into a teacher (admin) account which can go on to access functions that the student user cannot as you will see when testing criteria 20,21,22,23,27,28,29,30. You can see examples of logging into a teacher account in the iterative testing section on this screen found Here

4.1.2.2.4 CRITERION 18

By pressing the exit button the program should close leaving no remaining processes. It does this successfully by running exit(0) upon the button press.

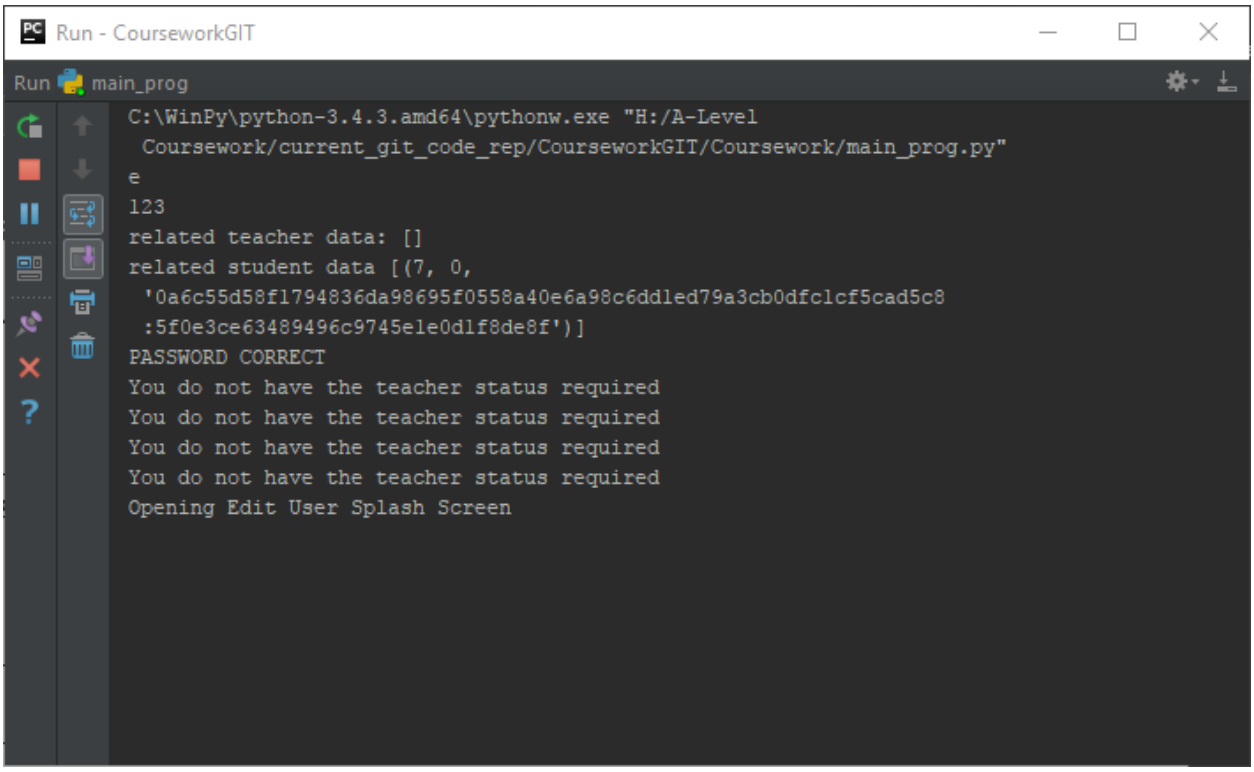
It has fulfilled this criterion

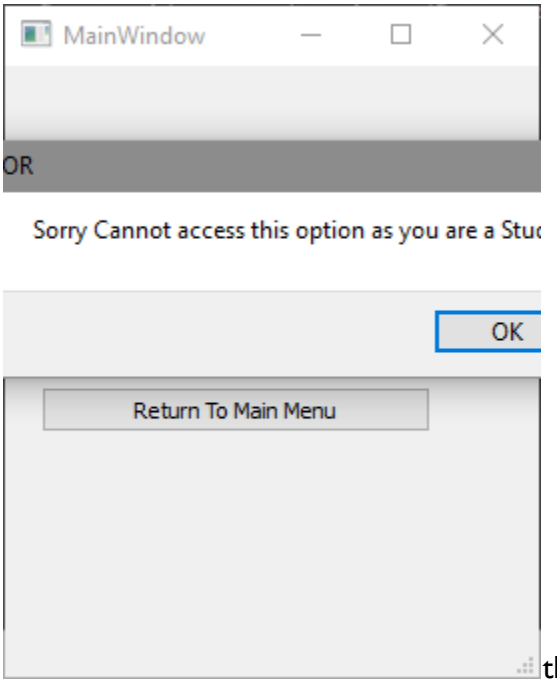
The login screen has fulfilled all stated criteria

4.1.2.3 SPLASH SCREEN

4.1.2.3.1 CRITERIA 20,21,22,23

Logging into the student test account and then pressing the buttons leading to the admin screens produced this result:



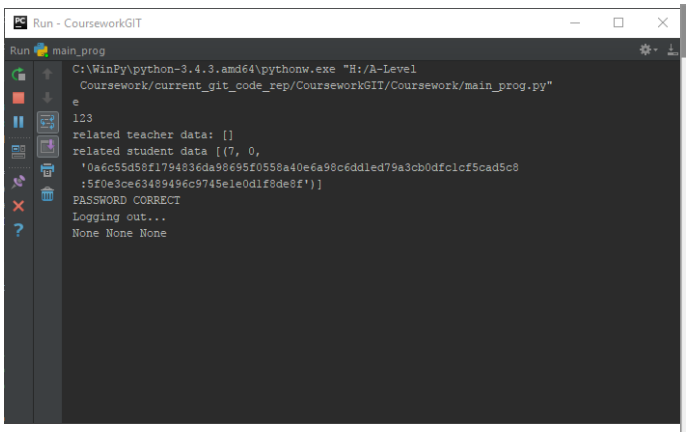


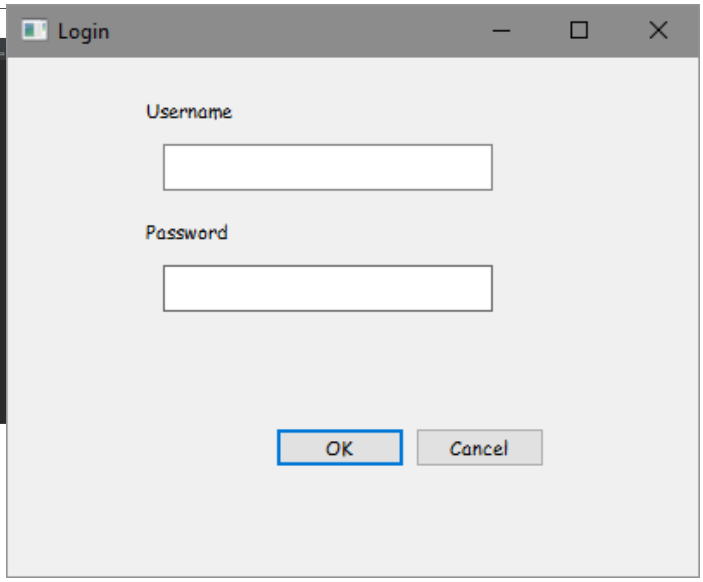
this is the required result and therefore the program fulfills these criteria

4.1.2.3.2 CRITERIA 24,25

Upon pressing the logout button the login screen should display with empty inputs and the user object attributes should be cleared

The result of logging out is:





The program has fulfilled these criteria

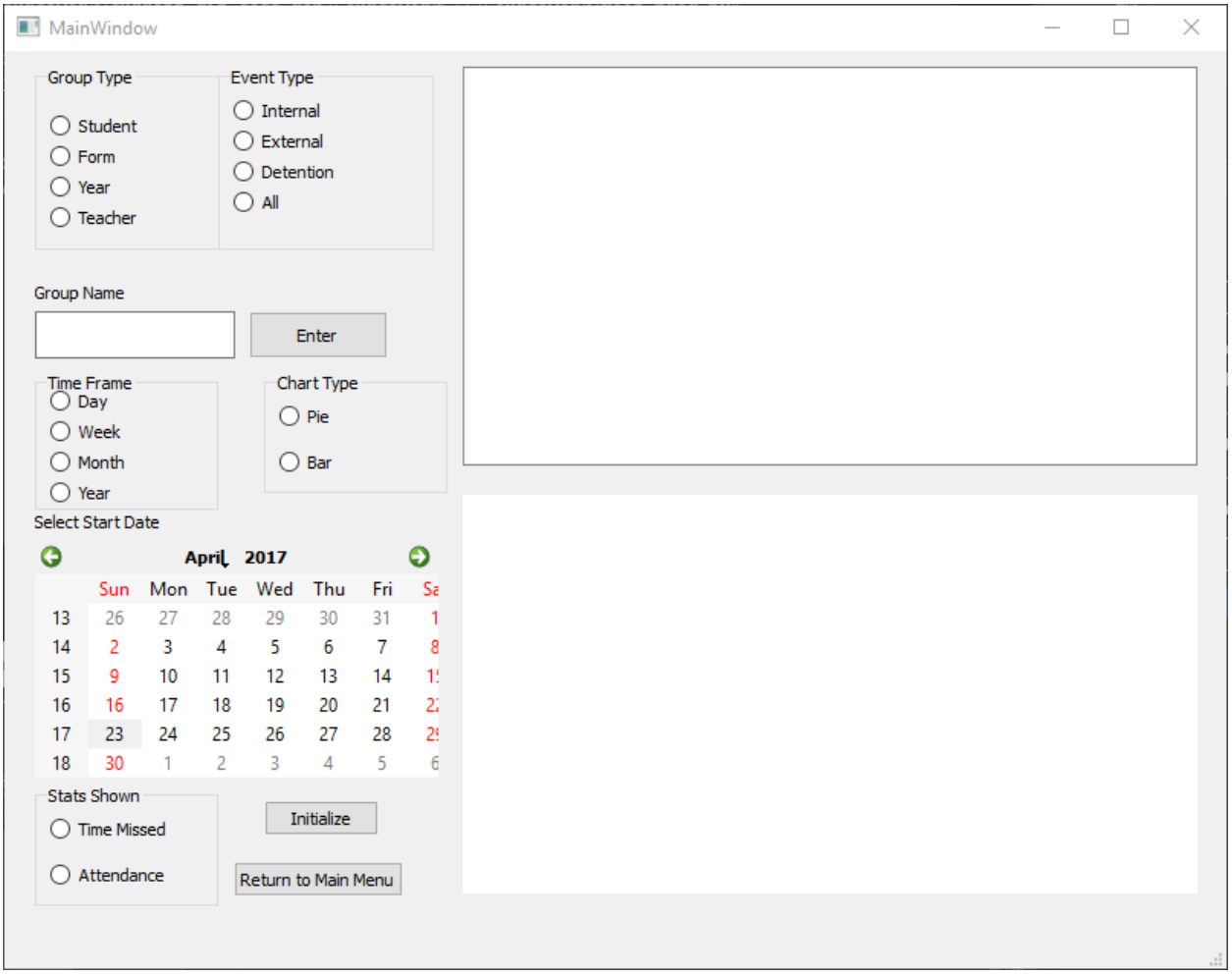
4.1.2.3.3 CRITERION 26

Pressing the exit button runs an exit(0) script which ends the current python process, this has fulfilled the criterion

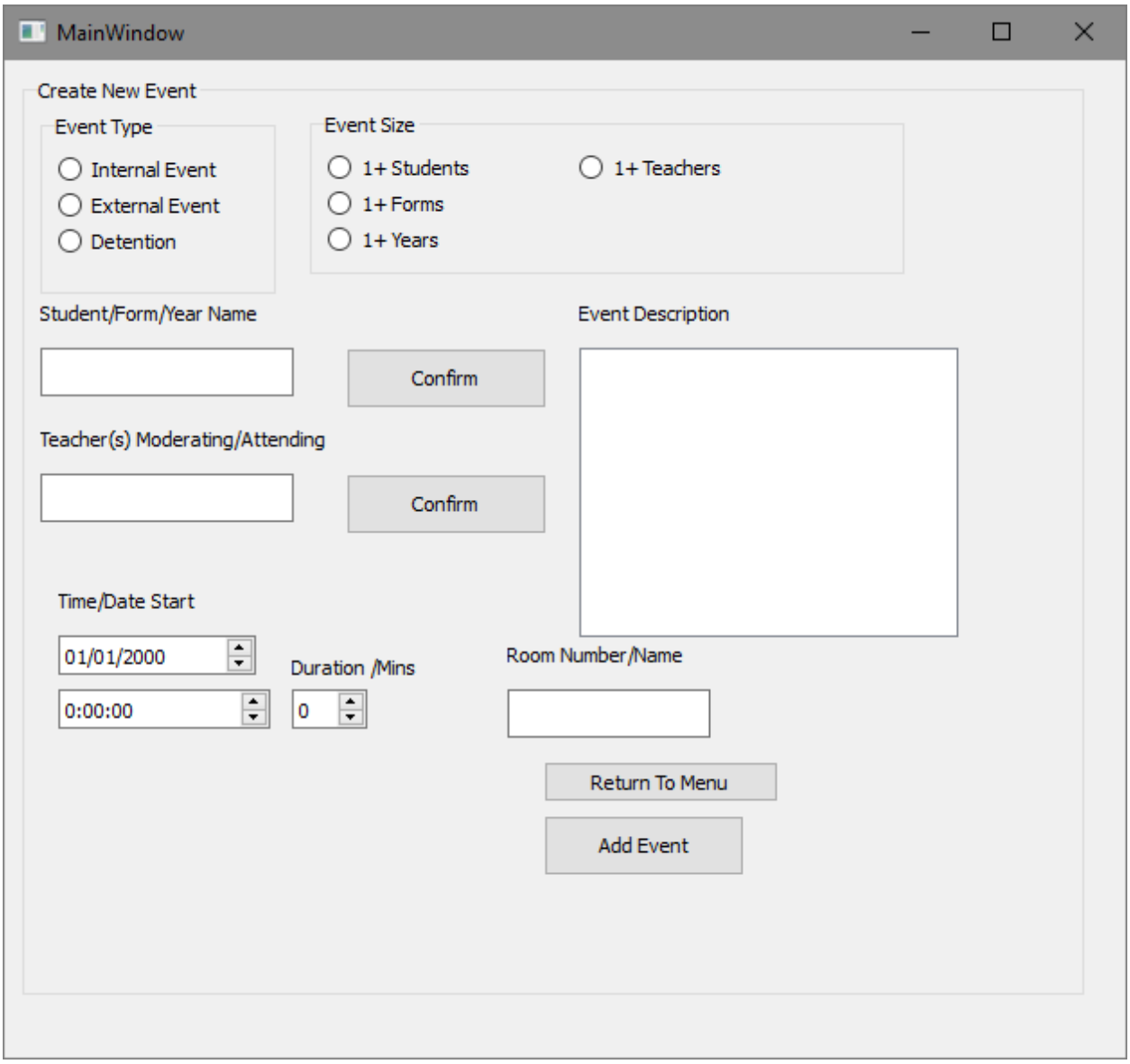
4.1.2.3.4 CRITERIA 27,28,29,30,31,32

Pressing the following buttons produces the following results:

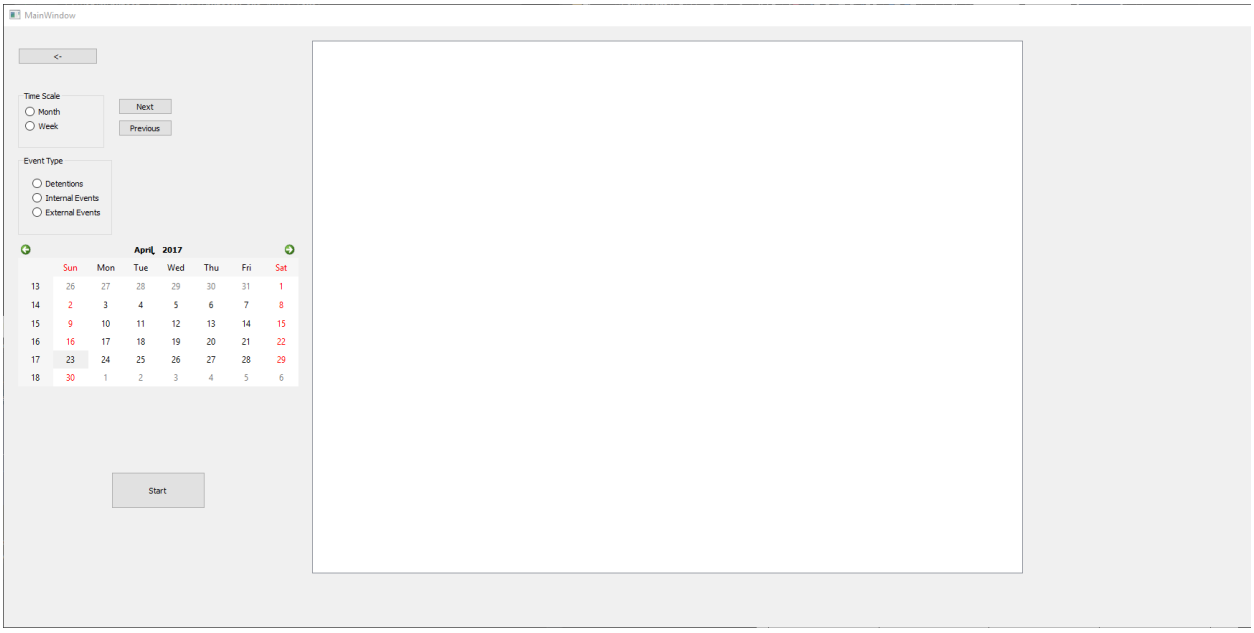
Advanced analytics:



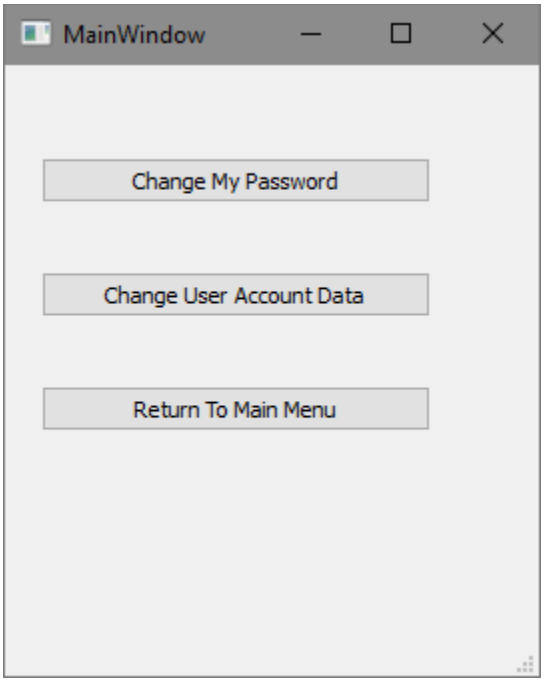
Create new event:



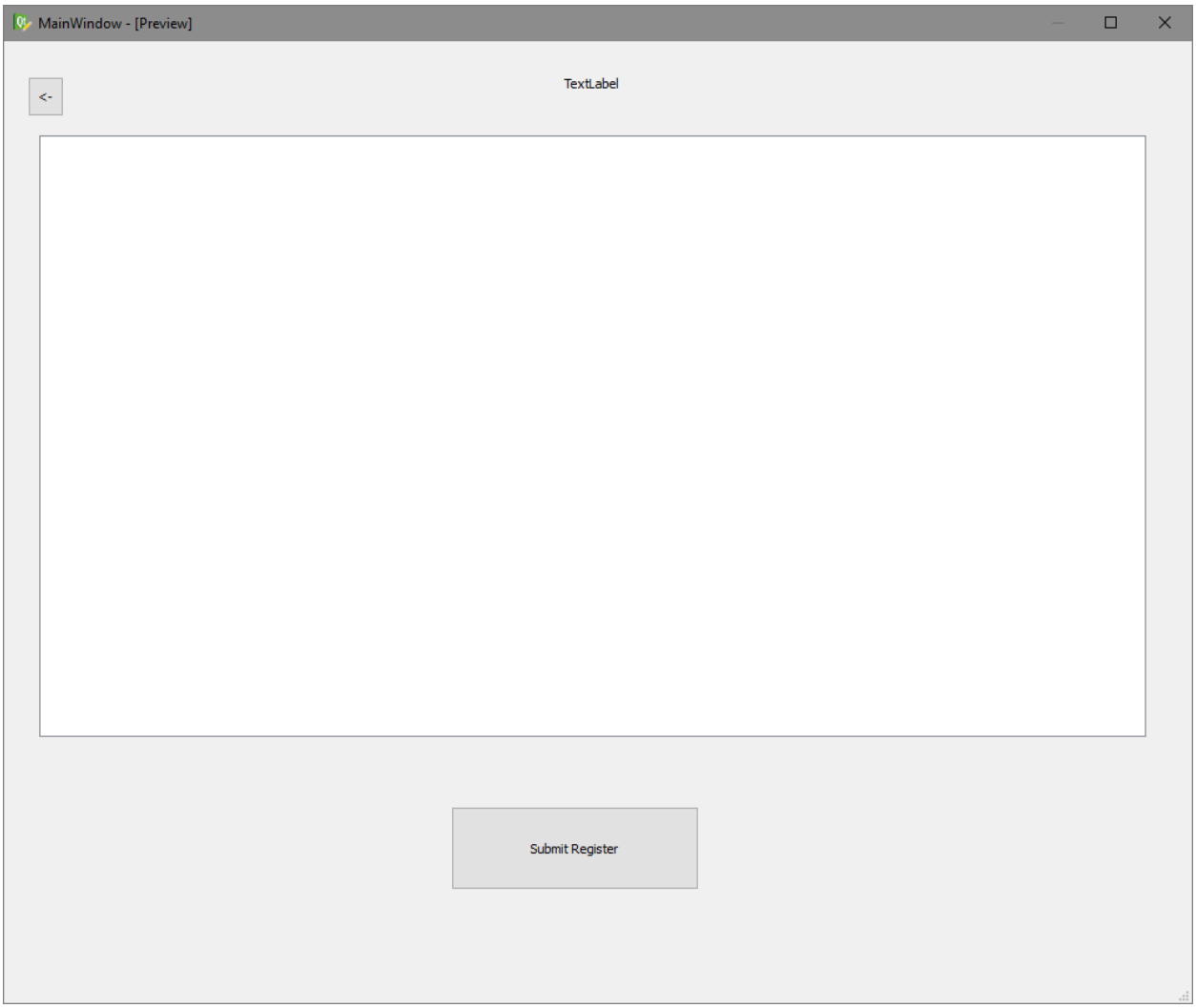
See personal events:



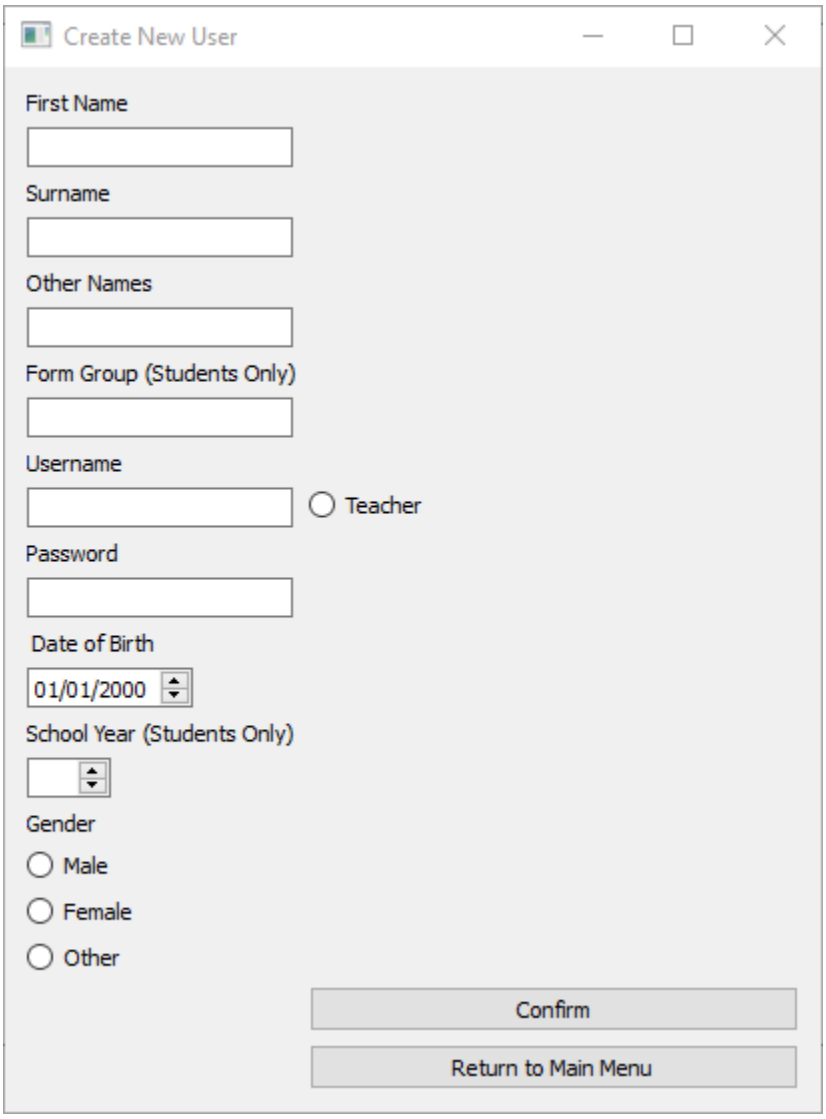
Edit user data:



Register event:



Create new user:



This is the expected result for the tests therefore the program has fulfilled these criteria

4.1.2.4 CREATE NEW USER SCREEN

4.1.2.4.1 CRITERIA 33,34,35

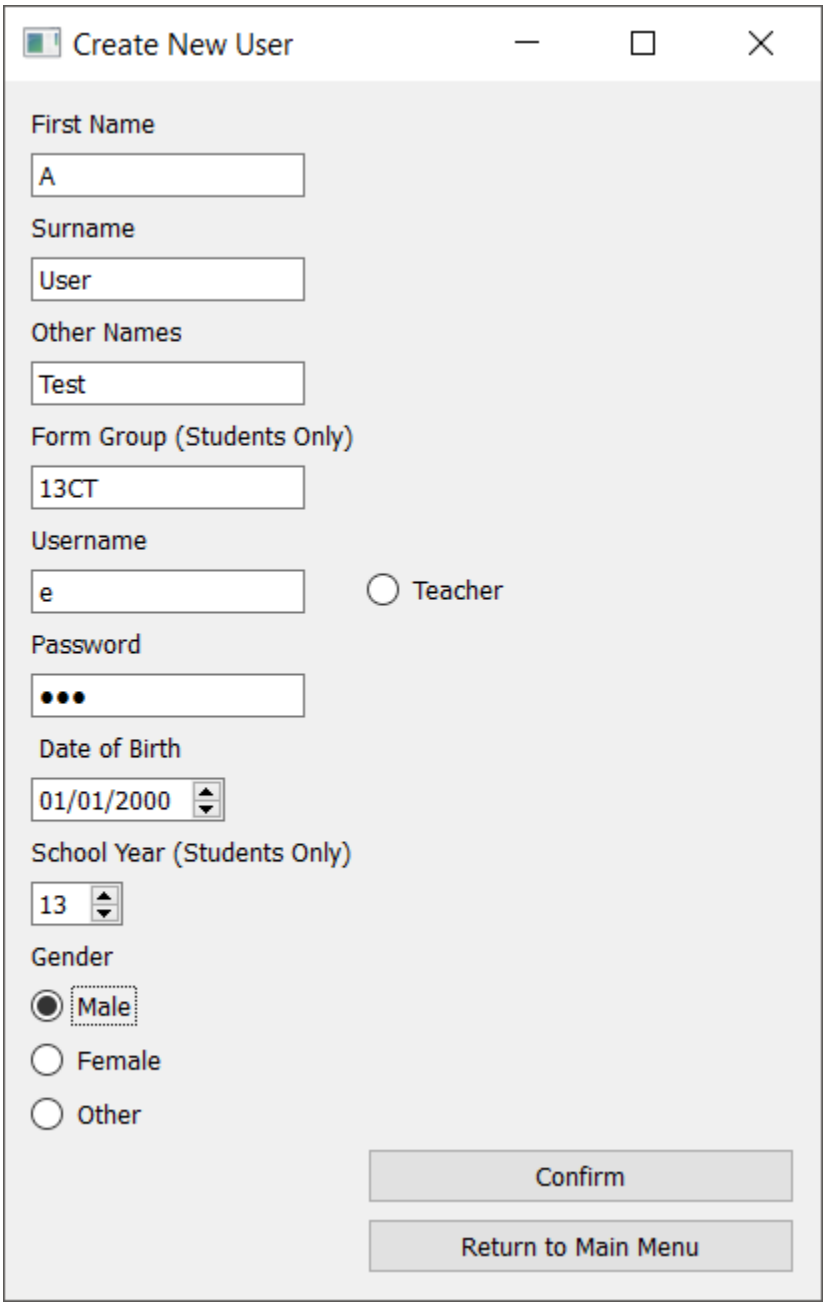


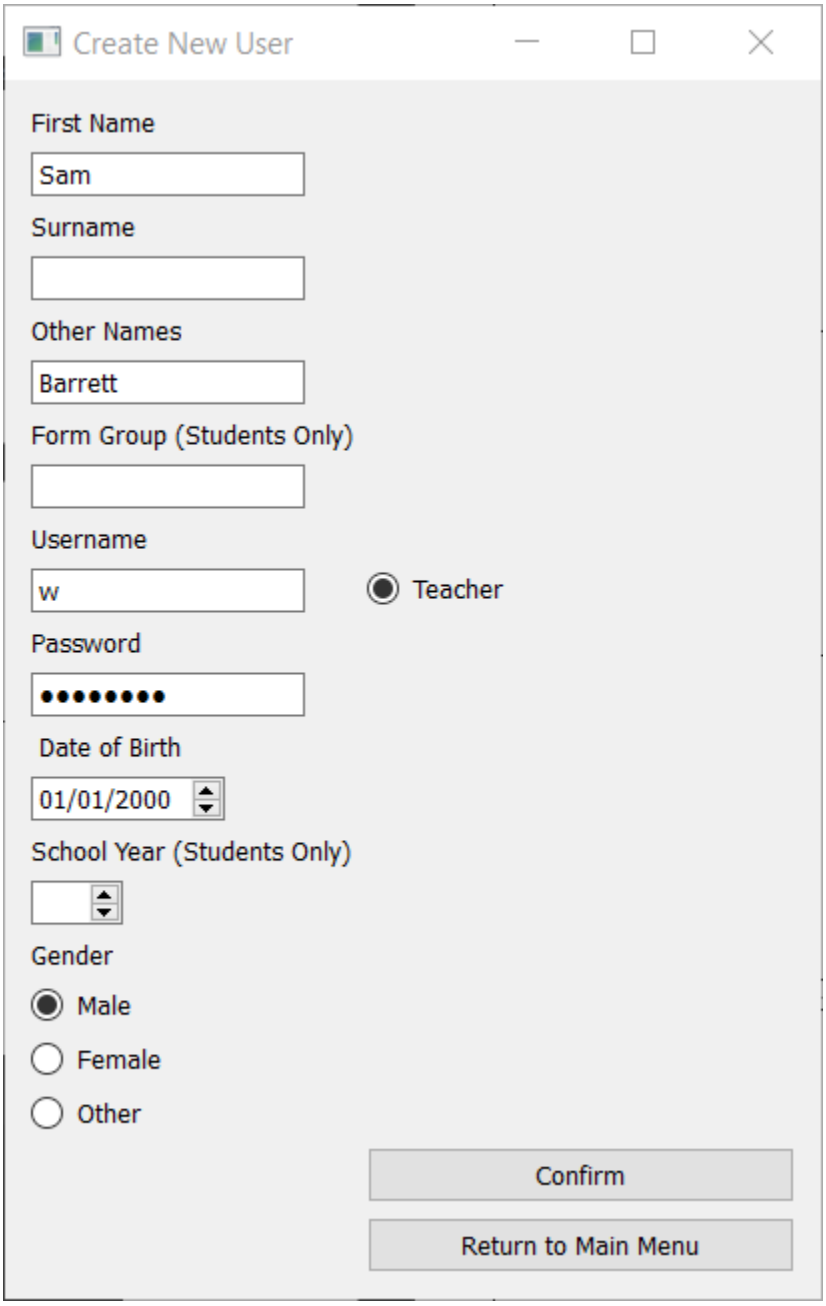
Figure 2

The input shown in **Figure 1** gives the following result:

2	7	A	Test	User	7	1	07/06/1999	e	0a6c55d58f1794836da98695f0558a4…	0	1

Therefore the program has fulfilled these criteria

4.1.2.4.2 CRITERION 36



5	Sam	Barrett	w	2fcaf63a759a18b2d617cbaa1650252ec01fa845428df2658b2a429bdeaade29:8a039676fffa4d219…	1

4.1.3 USER TESTING

I provided my original interviewees with the final prototype code and asked them to use the progrmam for a minimum of 20 minutes and to provide me with feedback including any missing features or suggested changes

4.1.3.1 FEEDBACK 1

Key points:

- User interfaces are functional but difficult to use initially.
- Difficult to run, requiring the installation of python and an IDE
- Very stable, no crashes
- Some fonts are too small or buttons are fairly labelled
    - Therefore failure of **Essential features/ Core functionality/ 2**

Timed Tests:

- Time to login = 5seconds , acceptable
- Time to access personal events from splash screen = 3 seconds, acceptable

- Mansoor Wooding

4.1.3.2 FEEDBACK 2

Key points:

- Solid use of database integration, full normalisation present
- Could be improved through the use of a more fully-featured version of SQL (MySQL etc.)
- User interface could prove challenging for non-tech oritented students and staff
- Very few visual queues for user eg. Red colouring on any required fields that have been left blank

Timed tests:

- Time to login = 3 seconds, acceptable
- Time to create event = 80 seconds, too long
    - To optimise, make sure user interface flows and hitting tab takes you to the next field

- Daniel Thomas

4.1.3.3 FEEDBACK 3

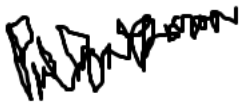
- In its current form not useful to the school as a whole due to its poor networking integration
- Client side functionality is good but user interfaces could do with improvement before final release
    - Perhaps integrating google material design
- Needs more statistical analysis functions for admin and senior faculty members
    - To allow for more in depth monitoring of students inline with our school ethos

- System for editing user details is very streamlined and secure, much improved over the current SIMS based solution

Timed Tests:

- Time to logon = 3 seconds, acceptable
- Time to find average attendance of form group = 1m30s, too long
    - User interface needs to be streamlined perhaps with frequently used inputs being saved as profiles for later use
- Time to change user password = 20 seconds, good time much faster than current system

- Paul Berkman



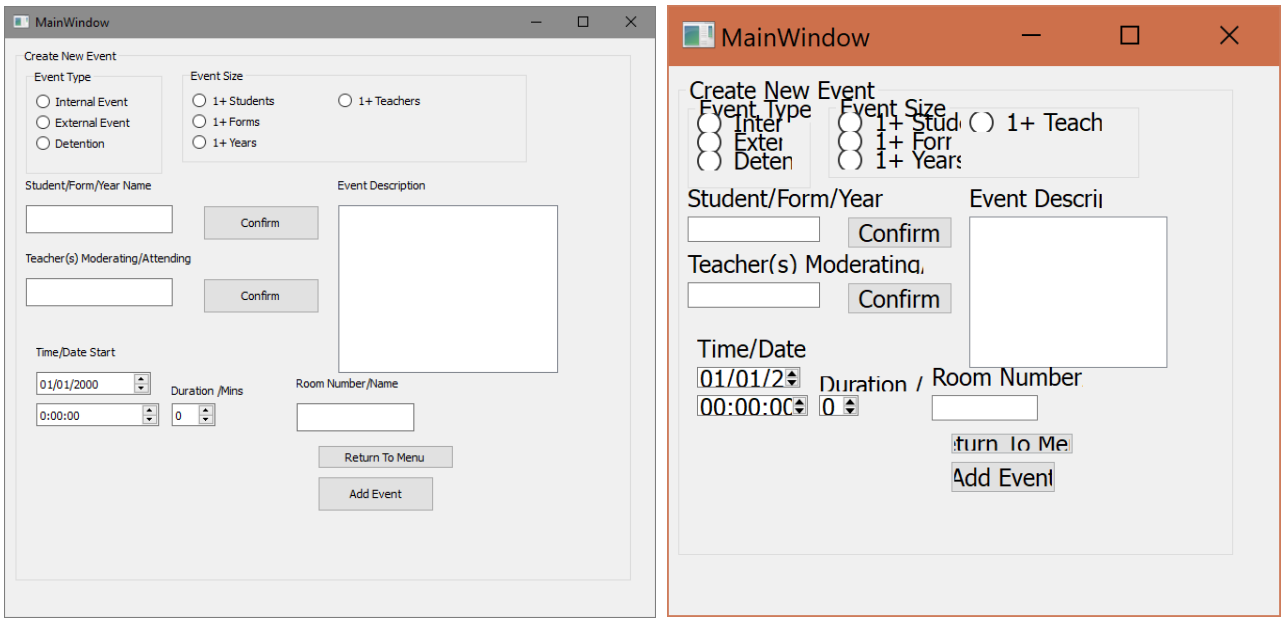
4.1.4 MISCELLANEOUS TESTS

4.1.4.1 DISPLAY SCALING

The program should be able to run on multiple systems of varying display resolutions and aspect ratios.

To test this I have run the program on my personal system which runs at 1080p on a 23" monitor. All GUI screenshots up to this point have been taken from this monitor. To test this objective I will run the program again on a second system which uses a 15" 4k resolution screen (3840x2160).

Example of program on 1080p desktop system against 4k system:



Here you can see that the scaling has not worked and the GUI on the 4k display (**Figure 2**) is almost unusable. This is therefore a failed test. To address this issue I need to look into PyQt4 scaling and perhaps port to a different library such as PyQt5 or Tkinter.

4.2 PARTIAL SUCCESSES

- A few of the features in the program, in my opinion are only partially successfully integrated :
    - The program does not allow for networking
        - This could be addressed in a second version of the program
        - As this is only a prototype the second version may be the final before release to consumers
        - The second version of the program could be written in a different language that lends itself more to networking
    - The program does not allow the program to be run on multiple systems
        - i.e. due to the restrictions of SQLite3 we cannot host the database and access it remotely. This prevents us from running the system on more than one system
        - to fix this we could port over to using a more fully-featured MySQL or alternative
        - the program is also struggles to run on non-windows, Linux or MacOS systems due to our choice of programming language

- to fix this issue the second version of the code could be written in a more mobile friendly language such as Java or Swift

4.3 OVERALL CONCLUSION BASED ON SUCCESS CRITERIA TESTING

Overall I feel that my program has satisfied most of the predefined success criteria. Based on the number and importance of those left out I feel confident in saying that my solution does what it is supposed to do efficiently, however leaving room for improvement in later iterations.

4.4 FINAL DESIGN

As you can see through my testing in the evaluation, found here, and my iterative testing in the development section, found here, you can see that my final GUI designs are very similar to my original mock-ups. They are in most cases functional with a few exceptions where improvements could be made. The design language leaves something to be desired as the gray is fairly dull and the buttons and other objects are fairly boring also.

In order to improve the design of the program in later iterations I may need to enlist the help of a creative professional who would be much more proficient at this sort of work when compared to me.

One example of where the GUI isn't as functional as I would have hoped is in the analytics screen which requires the entry of a lot of data via different fields which makes the whole system seem arduous and sloppy. To improve this I may remove some data entry and instead opt for pop-up windows or custom selection boxes. This would also reduce the amount of validation required.

4.5 MAINTAINABILITY

I feel that the final code is fairly easy to maintain for a developer with strong knowledge of python. I have commented various coding schemas that I have used throughout the program. I have also provided templates for many of the arrays used within the program to allow the programmer to understand the contents of the array during the running of the program.

4.5.1 COMPLEXITY

The complexity of my program is difficult to define due to several different functions being performed. Examples include:

- Searching database for students
  - Time and space complexity defined by $O(n)$
- Logging in
  - Time complexity of $O(n)$ for SQL select query
  - Time complexity of $O(1)$ for checking hash as it doesn't take longer due to a single comparison being made

4.6 GOING FORWARD…

4.6.1 PORT PROGRAM TO WEB OR MOBILE

One of the issues highlighted whilst testing and evaluating the success of my program was the lack of cross-platform support. The fact that this cannot run concurrently on many devices and that it can only run easily on Windows (7 or above), Linux and MacOSX limits its usefulness in a school environment.

Many schools including the Royal Grammar School utilize iPads in the classroom for most lessons. Therefore the ability for this program to run on these devices is paramount to the final release. To add this functionality the program would need to be ported to a supported language such as Swift. To do this I would need to completely redesign certain aspects of the program, mainly the external library usage as many of them will not have a direct swift counterpart. I could also attempt to use some of the available Python to Swift transpilers.

To run this program as a web client would require it to utilize different languages for the front and some of the backend. Using Django with python you can create web services although it may be easier to have PHP scripts doing the server side processing instead of python due to its wider usage.

The move to a different language with a different GUI framework might also remove the issue of scaling on different display resolutions and apect ratios.

4.6.2 FURTHER FEATURES

Having finished initial development I feel there are still some functions that would be useful to add for the next iteration:

- More statistical analysis options available to admin users
- Separate admin and student clients to increase security and perhaps reduce the amount of validation necessary

Due to the failed test namely in and around having a intuitive user interface; to rectify this I feel a different UI framework or the employ of a artistically minded individual may help to improve this for later iterations as it would allow for the creation of prettier GUIs which are easier for the customer to use and for people to be trained on.

Also responding to user feedback (see **Feedback 1**) I have added a feature whereby fields left blank by the user that require an input will be marked with red as shown below in **Figure 1**

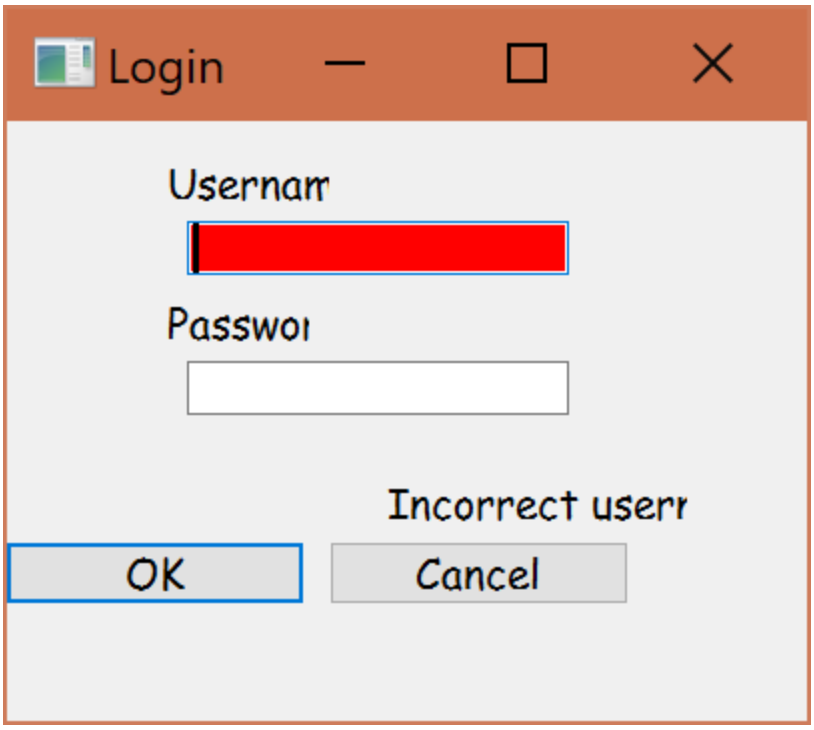


Figure 1

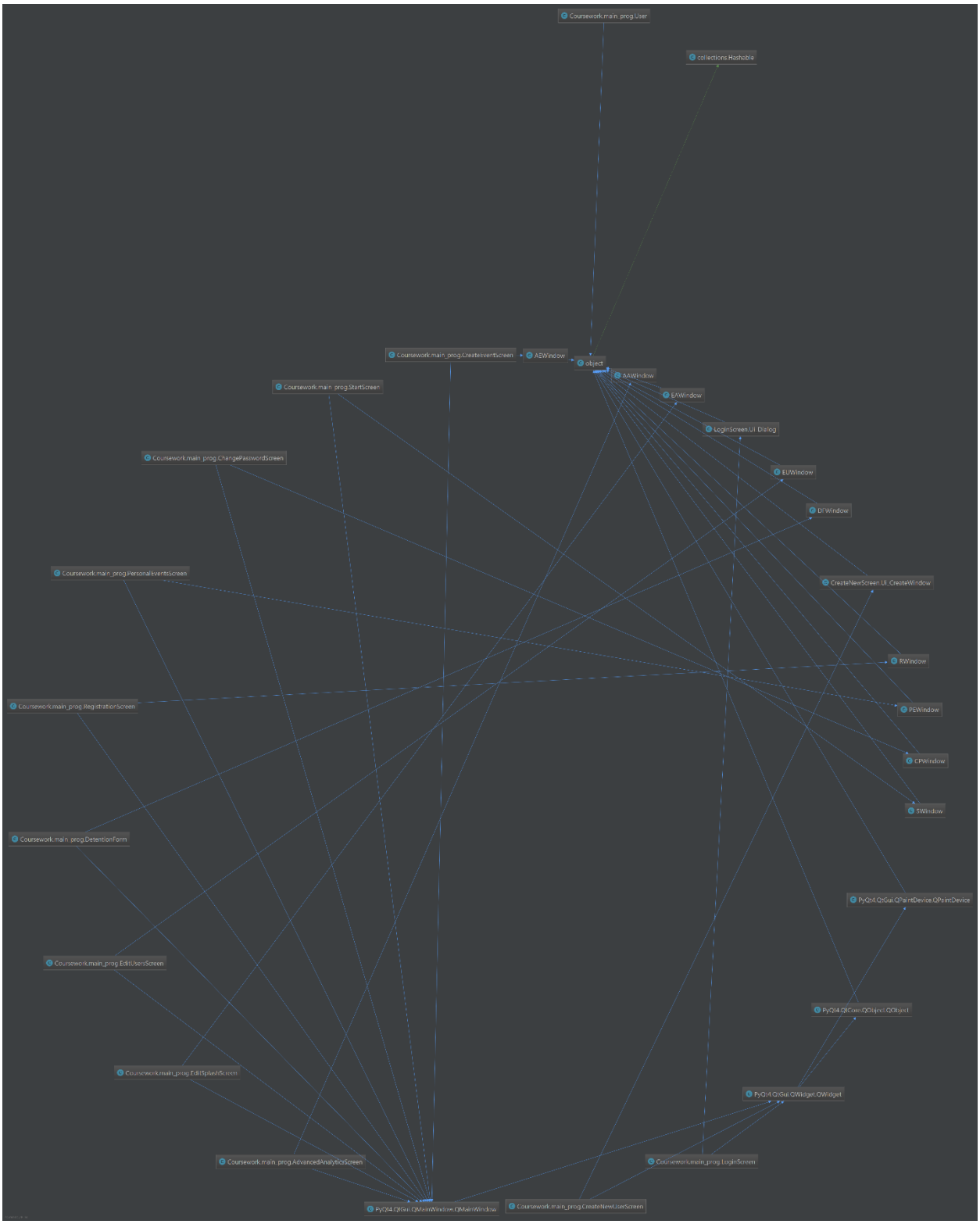
This, I feel, will make it much clearer to the user where data is missing when filling out some of the larger forms within the program. Based on showing this to Mansoor Wooding, the user who requested this usability feature, I feel this change is successful with him saying that it made data entry in the program "much clearer". This test was also run on the 4k system futher showing the scaling issues the PyQt Framework has

4.7 NOTE

The structure of this document was designed in accordance with chapters 19 through 21 of the OCR Computer Science for A Level textbook by George Rouse, Jason Pitt and Sean O'Byrne.

These chapters outlined the necessary content for each of the sections (1-4) showing what to include and what to leave out along with exemplars of work.

5 FINAL CODE




```
try:
    import re as regex
```

```
    import sqlite3 as lite
    from datetime import timedelta, datetime
    import datetime
    from PyQt4 import QtCore
    from PyQt4 import QtGui
    from PyQt4 import uic
    from PyQt4.QtWebKit import QWebSettings
    from dateutil import relativedelta
    import sys
    from CreateNewScreen import Ui_CreateWindow
    from LoginScreen import Ui_Dialog
    from Coursework import __miscfuncitons__
except:
    print("External Library File missing")


def __main__():
    pass


################################################
### USE QtGui.QMessageBox.information(self,"","")###
################################################

try:
    AEWindow = uic.loadUiType('./Resource Files/UI_Files\CreateEventScreen2.0.ui')[0]
    EAWindow = uic.loadUiType("./Resource Files/UI_Files\EditAccounts.ui")[0]
    SWindow = uic.loadUiType("./Resource Files/UI_Files\StartScreen.ui")[0]
    CPWindow = uic.loadUiType("./Resource Files/UI_Files\ChangePasswordScreen.ui")[0]
    DFWindow = uic.loadUiType("./Resource Files/UI_Files\DetentionForm.ui")[0]
    EUWindow = uic.loadUiType("./Resource Files/UI_Files\EditUsers.ui")[0]
    PEWindow = uic.loadUiType("./Resource Files/UI_Files\PersonalEventsScreen.ui")[0]
    AAWindow = uic.loadUiType("./Resource Files/UI_Files\AnalyticsScreen.ui")[0]
    RWindow = uic.loadUiType("./Resource Files/UI_Files\RegistrationScreen.ui")[0]
except:
    print("Error loading UI File")
    exit(0)


class User:
    def __init__(self):
        self.userid = None
        self.usertype = None
        self.acctype = None


class LoginScreen(QtGui.QWidget, Ui_Dialog):
    def __init__(self, parent=None):

        QtGui.QWidget.__init__(self, parent)
        self.setupUi(self)
        self.setObjectName("testtest")
        self.buttonBox.rejected.connect(lambda: exit(0))
        self.buttonBox.accepted.connect(self.checkpassword)
        self.StartWindow = None


    def checkpassword(self):
        username = self.User_Field.text()
        if username == "":
            QtGui.QMessageBox.information(self, "ERROR", "No username inputted")
        print(username)
        Password = self.Pass_Field.text()
```

```
        print(Password)

        cmd = 'SELECT TeacherID,User_Level,Password FROM Teachers WHERE username =?'
        try:
            cur.execute(cmd, (username,))
            data = cur.fetchall()
        except:
            print("Cannot find database file")
            exit(0)
        print("related teacher data:", data)

        if not data:
            cmd = 'SELECT StudentID, User_Level,Password FROM Students WHERE username
=?'
            cur.execute(cmd, (username,))
            data = cur.fetchall()
            print("related student data", data)
            if not data:
                self.Output_Label.setText("Incorrect username Try again")
                QtGui.QMessageBox.information(self, "ERROR", "Incorrect username
please try again")
            else:
                data.append("Student")

        else:
            data.append("Teacher")
        if __miscfuncitons__.verify_hash(self, Password, data[0][2]):
            print("PASSWORD CORRECT")
            self.Output_Label.setText("Logging in........")
            user_id = data[0][0]
            account_type = data[0][1]
            currentuser.userid = user_id
            currentuser.acctype = account_type
            currentuser.usertype = data[1]
            self.start()
        else:
            self.Output_Label.setText("Incorrect Password Try again")
            QtGui.QMessageBox.information(self, "Password Error", "Password is
incorrect, try again")
            pass

    def start(self):
        if self.StartWindow is None:
            self.StartWindow = StartScreen()

        self.StartWindow.show()

        LoginWindow.hide()


class StartScreen(QtGui.QMainWindow, SWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)

        self.CreateNewWindow = None
        self.RegistrationWindow = None
        self.AdvancedAnalyticsWindow = None
        self.PersonalEventsWindow = None
        self.EditUserWindow = None
        self.AE_Screen = None
        self.AEButton.clicked.connect(self.create_event_constructor)
        self.CNUButton.clicked.connect(self.create_new_user_contructor)
```

```
        self.EUButton.clicked.connect(self.edit_user_constructor)
        self.ExitButton.clicked.connect(lambda: exit(0))
        self.PEButton.clicked.connect(self.personal_events_contructor)
        self.LogoutButton.clicked.connect(lambda: __miscfuncitons__.logout(self,
LoginWindow, currentuser))
        self.AAButton.clicked.connect(self.advanced_analytics_contructor)
        self.REButton.clicked.connect(self.registration_constructor)

    def advanced_analytics_contructor(self):
        if currentuser.usertype != 'Teacher':
            print("You do not have the teacher status required")
            self.Output_Label.setText("Sorry Cannot access this option as you are a
Student")
        else:
            print("Opening analytics Screen")
            self.AdvancedAnalyticsWindow = AdvancedAnalyticsScreen()
            self.AdvancedAnalyticsWindow.show()
            self.hide()

    def registration_constructor(self):
        if currentuser.usertype != 'Teacher':
            self.Output_Label.setText("Sorry Cannot access this option as you are a
Student")
            print("You do not have the teacher status required")
        else:
            print("Opening registration screen")
            self.RegistrationWindow = RegistrationScreen()
            self.RegistrationWindow.show()
            self.hide()

    def personal_events_contructor(self):
        print("Opening Personal Events Screen")
        self.PersonalEventsWindow = PersonalEventsScreen()
        self.PersonalEventsWindow.show()
        self.hide()

    def edit_user_constructor(self):
        print("Opening Edit User Splash Screen")
        self.EditUserWindow = EditSplashScreen()
        self.EditUserWindow.show()
        self.hide()

    def create_new_user_contructor(self):
        if currentuser.usertype != 'Teacher':
            print("You do not have the teacher status required")
            self.Output_Label.setText("Sorry Cannot access this option as you are a
Student")
        else:
            if self.CreateNewWindow is None:
                self.CreateNewWindow = CreateNewUserScreen()
            print("Opening Create New User Screen")
            self.CreateNewWindow.show()
            self.hide()

    def create_event_constructor(self):
        if currentuser.usertype != 'Teacher':
            print("You do not have the teacher status required")
            self.Output_Label.setText("Sorry Cannot access this option as you are a
Student")
        else:
            print("Opening the Create New Event Screen")
            self.hide()
            self.AE_Screen = CreateEventScreen()
```

```
            self.AE_Screen.show()


class EditUsersScreen(QtGui.QMainWindow, EUWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.MMButton.clicked.connect(lambda: returntostart(self))
        self.SearchButton.clicked.connect(self.searchforuser)
        self.CommitButton.clicked.connect(self.upd_record)
        self.data = []
        self.tabletype = None
        self.tdata = []
        self.DataModel = None
        self.colcount = None
        self.updata = None

    def hashpass(self,password):
        import uuid
        import hashlib
        salt = uuid.uuid4().hex
        return hashlib.sha256(salt.encode() + password.encode()).hexdigest() + ":" +
salt

    def searchforuser(self):
        username = self.UserInput.text()

        if username == "" or None or len(username.split(" ")) > 2:
            QtGui.QMessageBox.information(self, "Invalid Username entered", "Invalid
Username Entered")

        try:
            searchdata = username.split(" ")
            cmd = 'SELECT s.StudentID,s.Forename,s.Other_Names,s.Surname,' \
                  '(SELECT Gender.Gender FROM Gender JOIN Students WHERE
Gender.GenderID = Students.GenderID ),' \
                  '(SELECT Years.Year FROM Years JOIN Students WHERE Students.YearID =
Years.YearID) ,' \
                  '(SELECT Forms.Form_Name FROM Forms JOIN Students WHERE Forms.FormID
= Students.FormID )' \
                  ',s.Date_of_Birth,s.Username,s.Password,s.User_Level' \
                  ' FROM Students s WHERE s.Forename = ? AND s.Surname = ?'
            cur.execute(cmd, (searchdata[0], searchdata[1],))
            self.data = cur.fetchall()
            self.tabletype = 0
            if not self.data:
                cmd = 'SELECT * FROM Teachers WHERE Forename =? AND Surname = ?'
                cur.execute(cmd, (searchdata[0], searchdata[1],))
                self.data = cur.fetchall()
                self.tabletype = 1
        except:
            cmd = 'SELECT s.StudentID,s.Forename,s.Other_Names,s.Surname,' \
                  '(SELECT Gender.Gender FROM Gender JOIN Students WHERE
Gender.GenderID = Students.GenderID ),' \
                  '(SELECT Years.Year FROM Years JOIN Students WHERE Students.YearID =
Years.YearID) ,' \
                  '(SELECT Forms.Form_Name FROM Forms JOIN Students WHERE Forms.FormID
= Students.FormID )' \
                  ',s.Date_of_Birth,s.Username,s.Password,s.User_Level' \
                  ' FROM Students s WHERE s.Username = ?'
            cur.execute(cmd, (username,))
            self.data = cur.fetchall()
            self.tabletype = 0
```

```
            if not self.data:
                cmd = 'SELECT * FROM Teachers WHERE Username = ?'
                cur.execute(cmd, (username,))
                self.data = cur.fetchall()
                self.tabletype = 1
        if self.tabletype == 1:
            self.tdata = list(self.data[0])
            self.tdata[4] = "******"
        if self.tabletype == 0:
            self.tdata = list(self.data[0])
            self.tdata[9] = "******"
        self.data = []
        self.data.append(self.tdata)

        self.DataModel = QtGui.QStandardItemModel(self)
        self.tableView.setModel(self.DataModel)
        if self.tabletype == 1:
            self.colcount = 6
            labellist = "UserID,Forename,Surname,Username,Password,User Level"
        if self.tabletype == 0:
            self.colcount = 11
            labellist = "UserID,Forename,Other Names,Surname,Gender,Year,Form,Date of
Birth,Username,Passowrd,User Level"
        __miscfuncitons__.load_data(self, self.data, labellist)

    def upd_record(self):
        self.updata = []
        for i in range(self.colcount):
            temp = self.DataModel.data(self.DataModel.index(0, i))
            self.updata.append(temp)
        if self.updata == self.data:
            QtGui.QMessageBox.information(self, "ERROR", "No Changes Made")

        newhashpass = self.hashpass(self.updata[-2])
        print(newhashpass)
        self.updata[-2] = newhashpass
        print(self.updata)

        if self.tabletype == 0:
            cmd = '''
                    UPDATE Students
                    SET Forename = ?,
                        Other_Names = ?,
                        Surname = ?,
                        YearID = ?,
                        FormID = ?,
                        Date_of_Birth = ?,
                        Username = ?,
                        Password = ?,
                        User_Level= ?,
                        GenderID = ?
                    WHERE StudentID = ?
                            '''
            cur.execute(cmd, (
                self.updata[1], self.updata[2], self.updata[3], self.updata[4],
self.updata[5], self.updata[6],
                self.updata[7], self.updata[8], self.updata[9], self.updata[10],
self.updata[0],))
            con.commit()

        if self.tabletype == 1:
            cmd = '''
                    UPDATE Teachers
```

```
                    SET Forename = ?,
                        Surname = ?,
                        Username = ?,
                        Password = ?,
                        User_Level = ?
                    WHERE TeacherID = ?
            '''
            cur.execute(cmd, (
                self.updata[1], self.updata[2], self.updata[3], self.updata[4],
self.updata[5], self.updata[0],))
            con.commit()


class EditSplashScreen(QtGui.QMainWindow, EAWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.MMButton.clicked.connect(lambda: returntostart(self))
        self.CMPButton.clicked.connect(self.cpfunc)
        self.CUADButton.clicked.connect(self.cuadfunc)
        self.ChangePasswordWindow = None
        self.ChangeUserDataWindow = None

    def cpfunc(self):
        self.ChangePasswordWindow = ChangePasswordScreen()
        self.ChangePasswordWindow.show()
        self.close()

    def cuadfunc(self):
        if currentuser.usertype != 'Teacher':
            QtGui.QMessageBox.information(self, "ERROR", "Sorry Cannot access this
option as you are a Student")
        else:
            self.ChangeUserDataWindow = EditUsersScreen()
            self.ChangeUserDataWindow.show()
            self.close()


class ChangePasswordScreen(QtGui.QMainWindow, CPWindow):
    def __init__(self, parent=EditUsersScreen):
        QtGui.QMainWindow.__init__(self)
        self.setupUi(self)
        self.MMButton.clicked.connect(lambda: returntostart(self))
        self.CPButton.clicked.connect(self.cpmethod)


    def cpmethod(self):
        currentpassword = self.CurPassField.text()
        confirmcurrentpassword = self.ConCurPassField.text()
        if currentpassword != confirmcurrentpassword:
            self.ErrorLabel.setText("Current Passwords Do not Match Try again")
            pass
        newpassword = self.NewPassField.text()

        confirmnewapassword = self.ConNewPassField.text()
        if newpassword != confirmnewapassword:
            self.ErrorLabel.setText("New Passwords do not match Try again")
        else:
            hashedpass = EditUsersScreen.hashpass(self, newpassword)
            if currentuser.usertype == "Student":
                cmd = 'UPDATE Students SET Password = ? WHERE StudentID = ?'

            else:
```

```
                cmd = 'UPDATE Teachers SET Password = ? WHERE TeacherID = ?'
            cur.execute(cmd, (str(hashedpass), currentuser.userid,))
            con.commit()
            returntostart(self)


class CreateNewUserScreen(QtGui.QWidget, Ui_CreateWindow):
    def __init__(self, parent=ChangePasswordScreen):
        self.parent = parent
        QtGui.QWidget.__init__(self)
        self.setupUi(self)
        self.ConfirmButton.clicked.connect(self.append)
        self.ReturnButton.clicked.connect(lambda: returntostart(self))

    def append(self):
        FName = self.FNameInput.text()
        SName = self.SNameInput.text()
        UName = self.UNameInput.text()
        Password = self.PasswordInput.text()
        pcheck, ucheck = __miscfuncitons__.checkdata(UName, Password, cur)
        if not pcheck["Overall"]:
            error_string = "Password is not valid %s"
            error_string = error_string.replace("%s", str(pcheck))
            print(error_string)
            QtGui.QMessageBox.information(self, "ERROR", error_string)
        elif ucheck:
            QtGui.QMessageBox.information(self, "ERROR",
                                          "Username is not valid, less than 8
characters or already in use")
        else:
            Hashed_Password = self.parent.hashpass(self,Password)
            if (FName or SName or UName or Password) == "":
                self.label_7.setText("Incorrect Values Try again")
            else:
                if self.GenderMaleButton.isChecked():
                    GenderID = 1
                elif self.GenderFemaleButton.isChecked():
                    GenderID = 2
                else:
                    GenderID = 3
                if self.IfTeacherButton.isChecked():
                    cmdtype = "Teacher"
                else:
                    cmdtype = "Student"

                if cmdtype == "Student":
                    OName = self.ONameInput.text()
                    DoB = self.DoBInput.text()
                    Year = self.YearInput.text()
                    Form = self.FormInput.text()
                    print(FName, SName, OName, UName, Password, Hashed_Password, DoB,
Year, Form)
                    Year = int(Year)
                    YearID = Year - 6
                    cmd = 'SELECT FormID FROM Forms WHERE YearID=? AND Form_Name = ?'
                    cur.execute(cmd, (YearID, Form[2:],))
                    data = cur.fetchall()
                    print(Form, data)
                    FormID = data[0][0]
                    # GET NEXT STUDENTID
                    if FName or OName or SName or UName or Password or DoB or Year or
Form == "" or None:
                        self.label_7.setText("Incorrect Values Try again")
```

```
                    cmd = 'SELECT StudentID FROM Students'
                    cur.execute(cmd)
                    data = cur.fetchall()
                    StudentID = data[-1][0] + 1
                    cmd = 'INSERT INTO Students (StudentID, Forename, Other_Names,
Surname, YearID, FormID, Date_of_Birth, Username, Password , User_Level,GenderID) ' \
                          'VALUES (?,?,?,?,?,?,?,?,?,?,?)'
                    cur.execute(cmd,
                                (StudentID, FName, OName, SName, YearID, FormID, DoB,
UName, Hashed_Password, 0,
                                 GenderID,))
                    con.commit()

                    self.label_7.setText("User Created")

                if cmdtype == "Teacher":
                    if currentuser.acctype >= 1:
                        cmd = 'SELECT TeacherID FROM Teachers'
                        cur.execute(cmd)
                        data = cur.fetchall()
                        TeacherID = len(data) + 1
                        cmd = 'INSERT INTO Teachers (TeacherID,
Forename,Surname,Username,Password,User_Level) VALUES (?,?,?,?,?,?)'
                        cur.execute(cmd, (TeacherID, FName, SName, UName,
Hashed_Password, 1,))
                        con.commit()

                    else:
                        pass  ######user below teacher


class PersonalEventsScreen(QtGui.QMainWindow, PEWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.MMButton.clicked.connect(lambda: returntostart(self))
        startdate = datetime.datetime.today().date()
        self.currentendbound = None
        self.currentstartbound = None
        self.data = []
        self.DataModel = None
        ts = 0
        if self.MRad.isChecked():
            ts = 1
        if self.WRad.isChecked():
            ts = 0
        curbound = self.currentstartbound
        self.PrevButton.clicked.connect(
            lambda: self.main(self.getlastbound(ts, curbound), curbound))
        self.NextButton.clicked.connect(
            lambda: self.main(self.currentendbound, self.getnextbound(ts,
self.currentendbound)))
        changetime = timedelta(days=startdate.weekday())
        self.InitialiseButton.clicked.connect(
            lambda: self.main(startdate - changetime, self.getnextbound(ts,
startdate)))

    def getlastbound(self, ts, sd):
        if self.MRad.isChecked():
            ts = 1
        if self.WRad.isChecked():
            ts = 0
        print(ts, "TSTS")
```

```
        if ts == 0:  # if time show = Week
            lmonday = sd - timedelta(days=sd.weekday(), weeks=1)
            try:
                lmonday = lmonday.date()
            except:
                lmonday = lmonday
            print("LBIP", sd)
            print("LBOP", lmonday)
            return lmonday

        if ts == 1:  # if time show = Month

            lmonth = sd + relativedelta.relativedelta(months=-1)
            try:
                lmonth = lmonth.date()
            except:
                pass
            print(lmonth)
            return lmonth

            pass

    def getnextbound(self, ts, sd):
        if self.MRad.isChecked():
            ts = 1
        if self.WRad.isChecked():
            ts = 0
        if ts == 0:  # if time show = Week
            startdate = str(sd)
            weekday = 0
            d = datetime.datetime.strptime(startdate, '%Y-%m-%d')
            t = timedelta((7 + weekday - d.weekday()) % 7)
            nweek = (d + t).strftime('%Y-%m-%d')
            print(nweek)  #
            nmonday = nweek

            return nmonday

        if ts == 1:  # if time show = Month

            return sd + relativedelta.relativedelta(months=1)

    def main(self, sd, ed):
        self.currentstartbound = sd
        self.currentendbound = ed
        grouptype = None
        if self.DRad.isChecked():
            grouptype = 3
        elif self.IERad.isChecked():
            grouptype = 1
        elif self.EERad.isChecked():
            grouptype = 2
        if currentuser.usertype == 'Student':  ## if studentEvents
            if grouptype is not None:
                cmd = '''SELECT (SELECT Events_Type.Description FROM Events_Type WHERE
Events_Type.TypeID= Events.TypeID),
                                        Events.Description,(SELECT Rooms.RoomNameNumber
FROM Rooms WHERE Rooms.RoomID = Events.RoomID)
                                       ,Events.Date,Events.Time,Events.Duration FROM
Events
                                        INNER JOIN CommitmentsStudent
                                        WHERE Events.EventID = (SELECT
CommitmentsStudent.EventID WHERE CommitmentsStudent.StudentID = ?)
```

```
                                        AND Events.Date BETWEEN ? AND ? AND
Events.TypeID = ? '''
                    cur.execute(cmd, (currentuser.userid, str(sd), str(ed), grouptype,))
                else:
                    cmd = '''SELECT (SELECT Events_Type.Description FROM Events_Type WHERE
Events_Type.TypeID= Events.TypeID),Events.Description,
                     (SELECT Rooms.RoomNameNumber FROM Rooms WHERE Rooms.RoomID =
Events.RoomID)
                     ,Events.Date,Events.Time,Events.Duration FROM Events
                     INNER JOIN CommitmentsStudent
                     WHERE Events.EventID = (SELECT CommitmentsStudent.EventID WHERE
CommitmentsStudent.StudentID = ?) AND Events.Date BETWEEN ? AND ? '''
                    cur.execute(cmd, (currentuser.userid, str(sd), str(ed),))
            else:  # if teacher
                print(grouptype)
                if grouptype is not None:
                    cmd = '''SELECT (SELECT Events_Type.Description FROM Events_Type WHERE
Events_Type.TypeID= Events.TypeID),Events.Description,
                             (SELECT Rooms.RoomNameNumber FROM Rooms WHERE Rooms.RoomID =
Events.RoomID)
                            ,Events.Date,Events.Time,Events.Duration FROM Events INNER
JOIN CommitmentsTeacher
                            WHERE Events.EventID = (SELECT CommitmentsTeacher.EventID
WHERE CommitmentsTeacher.TeacherID = ?) AND (Events.Date BETWEEN ? AND ?)
                            AND (Events.TypeID = ?)'''
                    cur.execute(cmd, (currentuser.userid, str(sd), str(ed), grouptype,))

                else:
                    cmd = '''SELECT (SELECT Events_Type.Description FROM Events_Type WHERE
Events_Type.TypeID= Events.TypeID),Events.Description,
                             (SELECT Rooms.RoomNameNumber FROM Rooms WHERE Rooms.RoomID =
Events.RoomID)
                            ,Events.Date,Events.Time,Events.Duration FROM Events INNER
JOIN CommitmentsTeacher
                             WHERE Events.EventID = (SELECT CommitmentsTeacher.EventID
WHERE CommitmentsTeacher.TeacherID = ?) AND Events.Date BETWEEN ? AND ?'''
                    cur.execute(cmd, (currentuser.userid, str(sd), str(ed),))

            self.data = cur.fetchall()
            print(self.data)
            self.DataModel = QtGui.QStandardItemModel(self)
            self.tableView.setModel(self.DataModel)
            # self.load_data()
            __miscfuncitons__.load_data(self, self.data, "Event
Type,Description,Location,Date,Time Start,Duration")


class AdvancedAnalyticsScreen(QtGui.QMainWindow, AAWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.MMButton.clicked.connect(lambda: returntostart(self))
        self.BButton.clicked.connect(self.mainfunc)
        self.GEnter.clicked.connect(self.getgname)
        self.GName = ""
        self.GType = None
        self.EType = None
        self.TFrame = None
        self.CType = None
        self.StartDateQT = None
        self.StartDate = None
        self.EndDate = None
        self.StatsShown = None
```

```
        self.userdata = None
        self.comdata = None
        self.durdata = None
        self.alldata = None
        self.Model = None
        self.data = None

    def getgname(self):
        self.GName = self.GIn.text()

    def mainfunc(self):
        # grouptype
        if self.SRad.isChecked():
            self.GType = 0
        if self.FRad.isChecked():
            self.GType = 1
        if self.YGRad.isChecked():
            self.GType = 2
        if self.TRad.isChecked():
            self.GType = 3
        # EventType
        if self.IERad.isChecked():
            self.EType = 1
        if self.EERad.isChecked():
            self.EType = 2
        if self.DERad.isChecked():
            self.EType = 3
        if self.ARad.isChecked():
            self.EType = 4
        # TimeFrame
        if self.DRad.isChecked():
            self.TFrame = 0
        if self.WRad.isChecked():
            self.TFrame = 1
        if self.MRad.isChecked():
            self.TFrame = 2
        if self.YRad.isChecked():
            self.TFrame = 3
        # ChartType
        if self.PRad.isChecked():
            self.CType = 0
        if self.BRad.isChecked():
            self.CType = 1
        # StartDate
        self.StartDate = self.calendarWidget.selectedDate()
        self.StartDateQT = self.StartDate
        self.StartDate = self.StartDate.toPyDate()
        print(self.StartDate)
        # EndDate
        if self.TFrame == 0:
            self.EndDate = QtCore.QDate.addDays(self.StartDateQT, 1)
        if self.TFrame == 1:
            self.EndDate = QtCore.QDate.addDays(self.StartDateQT, 7)
        if self.TFrame == 2:
            self.EndDate = QtCore.QDate.addMonths(self.StartDateQT, 1)
        if self.TFrame == 3:
            self.EndDate = QtCore.QDate.addYears(self.StartDateQT, 1)
        self.EndDate = self.EndDate.toPyDate()
        print(self.EndDate)
        # StatsShown
        if self.TMRad.isChecked():
            self.StatsShown = 0
        else:
```

```
            self.StatsShown = 1
        self.retdata()

    def retdata(self):
        self.userdata = []
        if self.GType == 0:  ##if student
            namesplit = self.GName.split(" ")
            print(len(namesplit), namesplit)
            if len(namesplit) == 1:
                cmd = 'SELECT * FROM Students WHERE Username = ?'
                cur.execute(cmd, (self.GName,))
            elif len(namesplit) == 2:
                cmd = 'SELECT * FROM Students WHERE Forename = ? AND Surname = ?'
                cur.execute(cmd, (namesplit[0], namesplit[1],))
            self.userdata = cur.fetchall()
        if self.GType == 3:  # if teacher
            namesplit = self.GName.split(" ")
            if len(namesplit) == 1:
                cmd = 'SELECT * FROM Teachers WHERE Username = ?'
                cur.execute(cmd, (self.GName,))
            elif len(namesplit) == 2:
                cmd = 'SELECT * FROM Teachers WHERE Forename = ? AND Surname = ?'
                cur.execute(cmd, (namesplit[0], namesplit[1],))
            self.userdata = cur.fetchall()

        if self.GType == 1:  # if form
            s = self.GName
            match = regex.compile("[^W\d]").search(s)
            namesplit = [s[:match.start()], s[match.start():]]
            cmd = 'SELECT * FROM Students WHERE Students.FormID =(SELECT Forms.FormID
FROM Forms WHERE Forms.Form_Name = ? AND Forms.YearID = (SELECT Years.YearID FROM
Years WHERE "Year" = ?))'
            cur.execute(cmd, (namesplit[1], namesplit[0],))
            self.userdata = cur.fetchall()

        if self.GType == 3:  # if year
            cmd = 'SELECT * FROM Students WHERE Students.YearID = (SELECT YearID FROM
Years WHERE "Year" = ?)'
            cur.execute(cmd, (self.GName,))
            self.userdata = cur.fetchall()
        self.comdata = []
        tempdata = []
        for each in self.userdata:  # selects all commitments related to user
            if self.GType != 3:
                if self.EType != 4:
                    cmd = "SELECT * FROM CommitmentsStudent WHERE StudentID = ? AND
(SELECT TypeID FROM Events WHERE CommitmentsStudent.EventID = Events.EventID ) = ? AND
(SELECT Events.Date " \
                          "FROM Events WHERE CommitmentsStudent.EventID =
Events.EventID) BETWEEN ? AND ?"
                    cur.execute(cmd, (each[0], self.EType, str(self.StartDate),
str(self.EndDate),))
                else:
                    cmd = 'SELECT * FROM CommitmentsStudent WHERE StudentID = ? AND
(SELECT Events.Date FROM Events WHERE CommitmentsStudent.EventID = Events.EventID)
BETWEEN ? AND ?'
                    cur.execute(cmd, (each[0], str(self.StartDate),
str(self.EndDate),))
                tempdata = cur.fetchall()
            else:
                if self.EType != 4:
                    cmd = 'SELECT * FROM CommitmentsTeacher WHERE TeacherID =? AND
(SELECT TypeID FROM Events WHERE CommitmentsTeacher.EventID = Events.EventID) = ? AND
```

```
(SELECT Events.Date ' \
                        'FROM Events WHERE CommitmentsTeacher.EventID =
Events.EventID) BETWEEN ? AND ?'
                    cur.execute(cmd, (each[0], self.EType, str(self.StartDate),
str(self.EndDate),))
                else:
                    cmd = 'SELECT * FROM CommitmentsTeacher WHERE TeacherID = ? AND
(SELECT Events.Date FROM Events WHERE CommitmentsTeacher.EventID = Events.EventID)
BETWEEN ? AND ?'
                    cur.execute(cmd, (each[0], str(self.StartDate),
str(self.EndDate),))
                tempdata = cur.fetchall()
            self.comdata.append(tempdata)
            print(self.comdata)
            self.comdata = self.comdata[0]
            self.durdata = []
            self.alldata = []
            for each in self.comdata:
                cmd = 'SELECT Duration,Time FROM Events WHERE EventID = ?'
                cur.execute(cmd, (each[2],))
                tempfetch = cur.fetchall()
                self.alldata.append([tempfetch, each])
            if self.StatsShown == 0:  # if stats on lesson time missed
                tottime = 0
                for each in self.alldata:
                    tottime += int(each[0][0][0])
                print(tottime)
                totevents = len(self.alldata)
                d = 0
                ie = 0
                ee = 0
                print(self.alldata)
                for i in self.alldata:
                    print("I", i)
                    cmd = 'SELECT TypeID FROM Events WHERE EventID = ?'
                    cur.execute(cmd, (i[1][2],))
                    temp = cur.fetchall()
                    print("TEMPSS", temp)
                    if temp[0][0] == 1:
                        ie += 1
                    if temp[0][0] == 2:
                        ee += 1
                    if temp[0][0] == 3:
                        d += 1

                modeldata = [self.GName, tottime, totevents, d, ie, ee]
                self.data = [modeldata]
                print(self.data)
                __miscfuncitons__.load_data(self, self.data,
                                            "Group Name,Total Time,Total
Events,Detenions,Internal Events,External Events")
            if self.StatsShown == 1:  # Event Attendance Data
                attended = 0
                missed = 0
                print("ALLDATA::", self.alldata)
                for each in self.alldata:
                    if each[1][3] != "None":
                        if each[1][3] == 0:
                            missed += 1
                        elif each[1][3] == 1:
                            attended += 1
                total = attended + missed
                self.data = [total, attended, missed]
```

```
            print(attended, missed)
            __miscfuncitons__.load_data(self, self.data, "Total
Events,Attended,Missed")

self.webView.settings().setAttribute(QWebSettings.LocalContentCanAccessRemoteUrls,
True)
            html = """
            <html>
              <head>
                <script type="text/javascript"
src="https://www.gstatic.com/charts/loader.js"></script>
                <script type="text/javascript">
                  google.charts.load('current', {'packages':['corechart']});
                  google.charts.setOnLoadCallback(drawChart);

                  function drawChart() {

                    var data = google.visualization.arrayToDataTable([
                      ['%q', '%w'],
                      ['Attended',     4],
                      ['Missed',      5],
                    ]);

                    var options = {
                      title: 'Attendance Data'
                    };

                    var chart = new
google.visualization.PieChart(document.getElementById('piechart'));

                    chart.draw(data, options);
                  }
                </script>
              </head>
              <body>
                <div id="piechart" style="width: 900px; height: 500px;"></div>
              </body>
            </html>
            """

            html = html.replace("%q", "Events")
            html = html.replace("%w", "Attened/Not Attended")
            html = html.replace("%a", str(attended))
            html = html.replace("%m", str(missed))
            self.webView.setHtml(html)

        # self.alldata =  [   [   [(duration,timestart)]
,(commitmentid,studentid,eventid,attended?)]  ,...]  <----template layout

        print(self.alldata, "alldta")



class CreateEventScreen(QtGui.QMainWindow, AEWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.ListTeach = []
        self.ListPart = []
        self.studcount = 0
        self.ReturnButton.clicked.connect(lambda: returntostart(self))
        self.SFY_Confirm_Button.clicked.connect(
            lambda: self.ListPart.append(self.getlistpart(self.SFY_Input.text())))
```

```
        self.T_Confirm_Button.clicked.connect(
            lambda: self.ListTeach.append(self.getlistteach(self.T_Input.text())))
        self.AddButton.clicked.connect(self.addevent)
        self.PartData = None
        self.Etype = None
        self.TeachData = None
        self.date = None
        self.timestart = None
        self.duration = None
        self.description = None
        self.EtypeID = None
        self.eventid = None
        self.tlb = None
        self.STUDID = None
        self.tub = None
        self.TEACHID = None
        self.datetime_object = None

    def getlistpart(self, inp):
        self.Etype = ""
        if self.S_Rad.isChecked():
            self.Etype = "Student"
        if self.F_Rad.isChecked():
            self.Etype = "Form"
        if self.Y_Rad.isChecked():
            self.Etype = "Year"
        self.PartData = []
        if self.Etype == "Student":
            inp = inp.split()
            if inp[0] == "":
                self.OutputLabel.setText("Invalid Name")
                QtGui.QMessageBox.information(self, "Error", "Invalid Name")
            self.SFY_Input.setText("")
            try:
                cmd = 'SELECT StudentID FROM Students WHERE Forename =? AND Surname =
?'
                cur.execute(cmd, (inp[0], inp[1],))
            except:
                cmd = 'SELECT StudentID FROM Students WHERE Username = ?'
                cur.execute(cmd, (inp[0],))
            checkdata = cur.fetchall()
            if not checkdata:
                print("No user with that name")
                QtGui.QMessageBox.information(self, "Error", "No such user")
            else:
                if self.PartData.__contains__(inp):
                    QtGui.QMessageBox.information(self, "Error", "Participant already
present")
                    print("Person already added")
                else:
                    self.studcount += 1
                    print(checkdata)
                    return checkdata
        if self.Etype == "Form":
            if inp == "":
                QtGui.QMessageBox.information(self, "Error", "No such form")
            self.SFY_Input.setText("")
            cmd = 'SELECT FormID FROM Forms WHERE Form_Name LIKE ?'
            cur.execute(cmd, (inp,))
            checkdata = cur.fetchall()  ##for EType being form or above need iterative
function to generate database
            if not checkdata:  ##entries where it goes through every student who is a
member of those groups
```

```
                print("NO SUCH FORM")
                QtGui.QMessageBox.information(self, "Error", "No such form")
            else:
                cmd = 'SELECT StudentID FROM Students WHERE FormID = ?'
                print("FORMCHECKDATA", checkdata)
                cur.execute(cmd, (checkdata[0][0],))
                studdata = cur.fetchall()
                self.studcount += len(studdata[0])
                return studdata

                # self.PartData.append(inp)

        if self.Etype == "Year":
            if inp == "":
                QtGui.QMessageBox.information(self, "Invalid Year Entry", "Invalid
Year Entry")
            self.SFY_Input.setText("")
            cmd = 'SELECT YearID FROM Years WHERE "Year" =?'
            cur.execute(cmd, (inp,))
            ydata = cur.fetchall()
            if not ydata:
                QtGui.QMessageBox.information(self, "Error", "No Such Year Group")
                print("No Such Year Group")
            cmd = 'SELECT StudentID FROM Students WHERE YearID =?'
            cur.execute(cmd, (ydata[0]))
            studdata = cur.fetchall()
            self.studcount += len(studdata[0])
            return studdata

    def getlistteach(self, inp):
        if self.T_Rad.isChecked():
            self.Etype = "Teacher"
        self.T_Input.setText("")
        self.TeachData = []
        inp = inp.split()
        print(inp)
        if inp[0] == "":
            QtGui.QMessageBox.information(self, "Invalid Name", "Invalid Name")
        if len(inp) == 1:
            cmd = 'SELECT TeacherID FROM Teachers WHERE Username = ?'
            cur.execute(cmd, (inp[0],))
        else:
            cmd = 'SELECT TeacherID FROM Teachers WHERE Forename =  ? AND Surname = ?'
            cur.execute(cmd, (inp[0], inp[1],))
        checkdata = cur.fetchall()
        print(checkdata)
        if not checkdata:
            print("No such member of staff")
            QtGui.QMessageBox.information(self, "No such member of staff", "No such
Member of Staff")
        else:
            self.TeachData.append(inp)
            return checkdata

    def checkeventtime(self):
        breakstart = datetime.time(11, 45, 0)
        breakend = datetime.time(12, 15, 0)
        lunchstart = datetime.time(13, 30, 0)
        lunchend = datetime.time(14, 10, 0)
        aschoolstart = datetime.time(15, 50, 0)
        aschoolend = datetime.time(17, 0, 0)
        schoolstart = datetime.time(8, 30, 0)
        schoolend = datetime.time(17, 0, 0)
```

```
        daystart = datetime.time(6, 15, 0)
        dayend = datetime.time(22, 0, 0)
        if self.EtypeID == 3:
            bb = __miscfuncitons__.time_in_range(breakstart, breakend, self.tlb) and
__miscfuncitons__.time_in_range(
                breakstart, breakend, self.tub)
            lb = __miscfuncitons__.time_in_range(lunchstart, lunchend, self.tlb) and
__miscfuncitons__.time_in_range(
                lunchstart, lunchend, self.tub)
            ab = __miscfuncitons__.time_in_range(aschoolstart, aschoolend,
                                                 self.tlb) and
__miscfuncitons__.time_in_range(aschoolstart, aschoolend,

self.tub)
            print(bb, lb, ab)
            if not bb:
                if not lb:
                    if not ab:
                        return False
                    else:
                        return True
                else:
                    return True
            else:
                return True
        elif self.EtypeID == 2:
            b = __miscfuncitons__.time_in_range(daystart, dayend, self.tlb) and
__miscfuncitons__.time_in_range(
                daystart, dayend, self.tub)
            if not b:
                return False
            else:
                return True
        elif self.EtypeID == 1:
            b = __miscfuncitons__.time_in_range(schoolstart, schoolend, self.tlb) and
__miscfuncitons__.time_in_range(
                schoolstart, schoolend, self.tub)
            if not b:
                return False
            else:
                return True

    def addevent(self):
        ### triggered when addevent button is clicked
        if self.IE_Rad.isChecked():
            self.EtypeID = 1
        if self.EE_Rad.isChecked():
            self.EtypeID = 2
        if self.D_Rad.isChecked():
            self.EtypeID = 3

        self.date = str((self.DateIn.date()).toPyDate())
        self.duration = self.DurationIn.text()

        self.timestart = str((self.TimeStart.time()).toPyTime())
        self.datetime_object = datetime.datetime.strptime(self.timestart, "%H:%M:%S")
        self.tub = self.datetime_object + timedelta(minutes=int(self.duration))
        self.tub = self.tub.time()
        self.tlb = self.datetime_object.time()
        print(self.tub, self.tlb)
        ## check time and duration to be suitable
        '''Detentions must start and end between 11.45->12.15, 13.30->14.10, 15.40-
>17.00'''
```

```
        print(self.checkeventtime())
        if self.checkeventtime():
            self.description = str(self.Desc_Box.toPlainText())
            print(self.ListPart, self.ListTeach)
            # iteratively goes through each in the list of students andor teachers and
adds an associated commitment linked to their account
            cmd = 'SELECT RoomID FROM Rooms WHERE RoomNameNumber = ?'
            cur.execute(cmd, (self.RoomEntry.text(),))
            RoomID = cur.fetchall()
            print("ROOMID", RoomID)
            ## CHECK ROOM AVAILABILITY
            self.tlb = self.timestart
            self.tub = str(self.tub)
            cmd = 'SELECT * FROM Events WHERE RoomID = ? AND Date = ? AND Events.Time
BETWEEN ? AND ?'
            cur.execute(cmd, (RoomID[0][0], self.date, self.tlb, self.tub,))
            checkdata = cur.fetchall()
            if checkdata:
                QtGui.QMessageBox.information(self, "ERROR", "THIS ROOM IS ALREADY
BOOKED AT THIS TIME")
                pass
            else:
                cmd = 'SELECT EventID FROM Events'
                cur.execute(cmd)
                etempdata = cur.fetchall()
                print(etempdata)
                etempdata = etempdata[-1:][0]
                print(etempdata)
                self.eventid = int(etempdata[0]) + 1
                if not RoomID:
                    RoomID = 5
                cmd = 'INSERT INTO Events (EventID, RoomID, TypeID, "Date", "Time",
Description, Duration) VALUES (?,?,?,?,?,?,?)'
                cur.execute(cmd,
                            (self.eventid, RoomID[0][0], self.EtypeID, self.date,
self.timestart, self.description,
                             self.duration,))
                con.commit()
                print(self.ListPart)
                if self.Etype == "Teacher":
                    self.add_teachers()
                if self.Etype != "Teacher" and self.studcount != 0:
                    self.add_students()
                    self.add_teachers()
                if self.Etype != "Teacher" and self.studcount == 0:
                    QtGui.QMessageBox.information(self, "Error", "No students to add")
                    print("No students to add")
        else:
            QtGui.QMessageBox.information(self, "ERROR", "Event not at a valid time")

    def printform(self):
        infodict = {
            "studentid": self.STUDID,
            "duration": self.duration,
            "timestart": self.timestart,
            "desctiption": self.description,
            "date": self.date,
        }

        DetForm = DetentionForm(infodict)
        # DetForm.__init__(infodict)
        DetForm.show()
```

```
    def add_students(self):
        print(self.ListPart, "LISTPART")

        for Student in self.ListPart:
            if Student:
                print(Student, "THIS IS THE STUDENT DATA ")
                self.STUDID = Student
                cmd = 'SELECT ComID FROM CommitmentsStudent'
                cur.execute(cmd)
                tempdata = cur.fetchall()
                x = tempdata[-1:][0][0]
                x = int(x) + 1
                cmd = 'SELECT * FROM Events INNER JOIN CommitmentsStudent ON
Events.EventID = CommitmentsStudent.EventID WHERE CommitmentsStudent.StudentID = ? AND
Events.Date= ? AND Events.Time BETWEEN ? AND ?'
                cur.execute(cmd, (Student[0][0], self.date, self.tlb, self.tub,))
                print(Student[0][0], self.date, self.tlb, self.tub,
"Student,date,tlb,tub")
                checkdata = cur.fetchall()
                print(checkdata)
                if checkdata:
                    if self.EtypeID == 3:
                        if checkdata[0][2] == 3:
                            error_string = "User % has a double detention"
                            error_string = error_string.replace("%",
str(Student[0][0]))
                            QtGui.QMessageBox.information(self, "Error", error_string)
                            print("DOUBLE DETENTION")
                            pass
                        else:
                            confirmscreen = QtGui.QMessageBox()
                            confirmscreen.standardButtons()
                            reply = confirmscreen.question(self, "TEST", "TEST",
confirmscreen.Yes, confirmscreen.No)
                            if reply == confirmscreen.Yes:
                                self.printform()
                            else:
                                pass
                            cmd = 'INSERT INTO CommitmentsStudent (ComID, StudentID,
EventID) VALUES (?,?,?)'
                            print(((len(tempdata)), Student, self.eventid))
                            cur.execute(cmd, (x, Student[0][0],
                                              self.eventid), )
                            cmd = 'DELETE FROM CommitmentsStudent WHERE EventID = ?'
                            cur.execute(cmd, (checkdata[0][0],))
                            con.commit()
                    else:
                        error_string = "User % has a double event"
                        error_string = error_string.replace("%", str(Student[0][0]))
                        QtGui.QMessageBox.information(self, "Error", error_string)
                        print("DOUBLE EVENT")
                        pass
                else:
                    if self.EtypeID == 3:

                        confirmscreen = QtGui.QMessageBox()
                        confirmscreen.standardButtons()
                        reply = confirmscreen.question(self, "TEST", "TEST",
confirmscreen.Yes, confirmscreen.No)
                        if reply == confirmscreen.Yes:
                            self.printform()
                        else:
                            pass
```

```
                    cmd = 'INSERT INTO CommitmentsStudent (ComID, StudentID, EventID)
VALUES (?,?,?)'
                    print(((len(tempdata)), Student, self.eventid))
                    cur.execute(cmd, (x, Student[0][0], self.eventid), )
                    con.commit()
            else:
                pass

    def add_teachers(self):
        for Teacher in self.ListTeach:
            if Teacher:
                self.TEACHID = Teacher
                cmd = 'SELECT ComID FROM CommitmentsTeacher'
                cur.execute(cmd)
                tempdata = cur.fetchall()
                x = tempdata[-1:][0][0]
                x = int(x) + 1
                cmd = 'SELECT * FROM Events JOIN CommitmentsTeacher WHERE
Events.EventID = ' \
                      '(SELECT CommitmentsTeacher.EventID WHERE
CommitmentsTeacher.TeacherID = ?)' \
                      ' AND Events.Date = ? AND Events.Time BETWEEN ? AND ?'
                cur.execute(cmd, (Teacher[0][0], self.date, self.tlb, self.tub,))
                checkdata = cur.fetchall()
                if checkdata:
                    print("Double event")
                    error_string = "Teacher % has a double Event"
                    error_string = error_string.replace("%", str(Teacher[0][0]))
                    QtGui.QMessageBox.information(self, "Error", error_string)
                    pass
                else:
                    cmd = 'INSERT INTO CommitmentsTeacher (ComID, TeacherID, EventID)
VALUES (?,?,?)'
                    cur.execute(cmd, (x, Teacher[0][0], self.eventid,))
                    con.commit()
            else:
                pass


class DetentionForm(QtGui.QMainWindow,
                    DFWindow):  # prints out a detention form specific to the student
being placed in detention
    def __init__(self, _info_obj):
        QtGui.QMainWindow.__init__(self, _info_obj)
        self.setupUi(self)
        print("STARTED")
        print(_info_obj)
        self.STUDID = _info_obj["studentid"]
        self.duration = _info_obj["duration"]
        self.timestart = _info_obj["timestart"]
        self.date = _info_obj["date"]
        print(self.STUDID, self.date, self.duration, self.timestart)
        self.Print_Button.clicked.connect(self.gotoprint)

    def gotoprint(self):  # prints the form filling it out with data from the
createEvent Screen

        ## dd/mm/yyyy format
        cmd = 'SELECT Forename FROM Students WHERE StudentID = ? '
        print(self.STUDID)
        cur.execute(cmd, (self.STUDID,))
        StudentData = cur.fetchall()
```

```
        StudentName = StudentData[1]
        self.Stud_Label.setText(StudentName)
        self.Date_Label.setText(self.date)
        cmd = '''SELECT (SELECT Years.Year FROM Years
                INNER JOIN Students
                WHERE Years.YearID == (SELECT Students.YearID WHERE StudentID = ?)
                ),(SELECT Form_Name FROM Forms
                INNER JOIN Students
                WHERE Forms.FormID == (SELECT Students.FormID WHERE StudentID = ?))
                '''
        cur.execute(cmd, (self.STUDID, self.STUDID))
        formname = cur.fetchall()
        formname = formname[0][0]
        yearname = formname[1][0]
        formgroup = yearname, formname
        self.Form_Label.setText(formgroup)
        self.Dur_Label.setText(self.duration)
        cmd = 'SELECT Username FROM Teachers WHERE TeacherID =?'
        cur.execute(cmd, (currentuser.userid,))
        tdata = cur.fetchall()
        tdata = tdata[0]

        self.Date_Label_2.setText(self.date)
        self.Start_Label.setText(self.timestart)
        self.Issue_Label.setText(tdata)
        self.Reason_Label.setText(self.description)
        printer = QtGui.QPrinter()
        dialog = QtGui.QPrintDialog(printer, self)
        if dialog.exec() != QtGui.QDialog.Accepted:
            return
        p = QtGui.QPixmap.grabWidget(self.ToPrint)
        printlabel = QtGui.QLabel()
        printlabel.setPixmap(p)
        painter = QtGui.QPainter(printer)
        printlabel.render(painter)
        painter.end().


class RegistrationScreen(QtGui.QMainWindow, RWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.MMButton.clicked.connect(lambda: returntostart(self))
        self.data = []
        self.updata = []
        self.DataModel = None
        self.flag = True
        self.eid = None
        self.selectevent()
        self.SubButton.clicked.connect(self.registration)

    def selectevent(self):
        currentdate = datetime.datetime.today().date()
        currenttime = datetime.datetime.now().time()
        cmd = '''SELECT * FROM Events e INNER JOIN CommitmentsTeacher t ON e.EventID =
t.EventID WHERE t.TeacherID = ? AND e.Date = ?'''
        ## TEST Returned event times
        cur.execute(cmd, (currentuser.userid, currentdate,))
        cureventdata = cur.fetchall()
        print(cureventdata, "AVANT")
        for event in cureventdata:
            tlb = datetime.datetime.strptime(event[4], '%H:%M:%S')
            tub = tlb + timedelta(minutes=int(event[6]))
```

```
            tlb = tlb.time()
            tub = tub.time()
            if not __miscfuncitons__.time_in_range(tlb, tub, currenttime):
                cureventdata.remove(event)
            else:
                pass
        cmd = '''SELECT s.StudentID, s.Forename, s.Surname, y.Year, f.Form_Name,
c."Attended?" FROM CommitmentsStudent c INNER JOIN Students s ON c.StudentID =
s.StudentID
                  INNER JOIN Years y ON s.YearID = y.YearID INNER JOIN Forms f ON
s.FormID = f.FormID WHERE c.EventID = ? '''
        print(cureventdata, "APRES")
        cur.execute(cmd, (cureventdata[0][9],))
        studentdata = cur.fetchall()
        self.eid = cureventdata[0][9]
        print(studentdata, "STUDENTDATA")
        for student in studentdata:
            listappend = []
            for item in student:
                listappend.append(item)
            self.data.append(listappend)

        for i in range(len(self.data)):
            if self.data[i][5] == 1:
                self.data[i][5] = "/"
            elif self.data[i][5] == 0:
                self.data[i][5] = "\\"
            else:
                pass

        __miscfuncitons__.load_data(self, self.data,
                                    "ID,Forename,Surname,Year,Form,Attended? ('/'Yes /
'\\' No )")

    def registration(self):
        self.updata = []
        for z in range(len(self.data)):
            temprow = []
            for i in range(len(self.data[0])):
                temp = self.DataModel.data(self.DataModel.index(z, i))
                temprow.append(temp)
            self.updata.append(temprow)
        for events in self.data:
            if self.updata == events:
                QtGui.QMessageBox.information(self, "ERROR", "No Changes Made")
        else:
            print(self.updata, "UPDATA")
            for each in self.updata:
                print(each[5], "EACH5")
                if each[5] == "/" or each[5] == "\\":
                    print("TRUE")
                    if each[5] == "/":
                        each[5] = 1
                    else:
                        each[5] = 0
                    print(each[5], "E5")
                    cmd = '''UPDATE CommitmentsStudent
                      SET "Attended?" =? WHERE StudentID = ? AND EventID = ?
                  '''
                    print(each[5], each[1], self.eid)
                    cur.execute(cmd, (each[5], each[0], self.eid,))
                    con.commit()
                else:
```

```
                    error_string = "ERROR With Value for user %s please enter '/' for
present or '\\' for absent"
                    error_string = error_string.replace("%s", str(each[1:3]))
                    QtGui.QMessageBox.information(self, "ERROR", error_string)
                    break


def returntostart(obj):
    obj.close()
    LoginWindow.StartWindow.show()


currentuser = User()
app = QtGui.QApplication(sys.argv)
try:
    con = lite.connect("Coursework_Database")
    cur = con.cursor()
except:
    print("ERROR loading Database file")
LoginWindow = LoginScreen()
LoginWindow.show()
sys.exit(app.exec_())
```