School Detention Planning System

Ryan Bowen

Centre number: 52423
Candidate number: 2322

Contents

Analysis of the problem

Problem identification

The current problem of a lack of sophisticated detention management systems is one that is widespread and relevant in schools across the country. Although almost every school uses detentions as a method of deterrent for bad behaviour, almost none of them uses long term storage of detention records, preventing them from carrying out any kind of useful statistical analysis. Not only this, but they also tend to use outdated methods of recording detention, by using physical, hand-written paper slips. I intend on creating a detention management system which would allow the booking, recording and data analysis of detentions in a given school.

How the problem is solvable by computational methods

In order for a problem to be solvable by computational approach, the solution should always be able to produce the required outputs by use of various inputs and formulas, regardless of fluctuations in the inputs. If the inputs are not valid, then program should be able to factor that in, by use of validation. Therefore, the program must be able to produce the required output given a specified amount of necessary inputs. This must always be the case, and any extra detail that may be present in the real world can be removed from the problem by abstraction in order to make the number of inputs manageable and keep the problem from becoming too complex to solve.

One of the parts of this problem is the method of creating a “detention slip”, which is essentially a template for information about a given detention, that is then manually filled in by the issuer of the detention. Therefore, a solution to this problem would require a similar, but more efficient, method of creating detention slips. By using abstraction, a detention slip may be viewed as a form where the only relevant variables are the following inputs: to whom the detention is addressed, the form or class of the student, the date of the detention, the start and end time of the detention, the location of the detention, the issuer of the detention and finally the reason for a detention. Different schools may view this differently, for example some may argue that the date that the detention is given is important. However, this is the list of necessary inputs for creating a detention slip which I have identified and will therefore be referring to from now on.

As these inputs are the only variables that are relevant for creating a detention slip, it may be concluded that the process of creating this specific output would always be achievable by a computer. It does not matter how the variables vary, the slip will always be produced as per the inputs. The actual act of creating the slip from the inputs would have a time complexity of O(1)

(constant time), which would make this individual problem completely solvable by computational means. In summary, for the detention slips, the solution would never become too complex to solve, and so the desired output would always be produced, assuming the inputs pass the various validation tests.

Another output that this program will produce will be various database queries that are displayed to the user in different forms. In general, any CRUD (create, read, update and delete) statements used in a database have the worst-case time complexity of O(log(n)), with smaller statements on smaller databases having complexities such as O(n) and even O(1). This means that any of these queries written to databases will always be achievable by a computer. It also means that my problem is not an intractable one, as there will be a polynomial-time frame for the solution of any given data set, as justified by the complexity of individual algorithms which I have begun to discuss now, and will discuss in further detail later.

Once the database is populated, statistical analysis will be required to create output for the users. The statistics produced may vary from graphs produced, which will visually show trends and relationships between data, to simple statements such as “student x has the most detentions” or “form x has y% of all detentions”. Once again, these outputs will all be dependent on a limited number of specific variables which are determined by the data within the database. The variables required will always be the same, and by use of validation in the program, the variables within the database will always be present and of the expected form. This means that once again, the problem will never become unsolvable, and thus the required outputs will always be produced. This has been achieved by use of abstraction; analysing all the variables that are involved when considering a detention, and then simply picking out only the relevant variables, ignoring the extra detail. An example of extra detail, which the solution will not take into consideration, may be the lesson that the student was in when the offense that caused the detention was committed. This is because it is not necessary for this to be known in order to create the output (detention slip, statistical analysis or long-term storage of detention records) that the customers (i.e. school) wants. Complexities of statistical analysis algorithms that I may end up used may include the following:

Size of a given data set – O(n)

Mean of a given data set – O(n)

Largest value of a given data set – O(n)

Minimum value of a given data set – O(n)

Although these may not be the only algorithms I will be using, these will allow me to give most required statistical outputs. Therefore, it is safe to conclude that my problem is in no way intractable. Therefore, I should be able to create a polynomial-time solution.

This solution may be viewed as prototype, for reasons that it will not have internet capabilities and that each school is different. All other features than internet access will be working, but due to how

each school varies, specifics cannot be finalized if I want to create a complete solution to the problem. Therefore, a lot of different assumptions will be made for each school. It is because of this, that I will be creating this prototype with a specific school in mind so that it may be used as a case study of sorts. This school is the Royal Grammar School in High Wycombe (RGS). The structure of the student body at the RGS is so that each student belongs to a form of approximately 30 students, with each form belonging to a year group. Each form tutor also has a teacher associated with it. This is important to note as I will be using this structure when I attempt to represent the student and teacher body within a database. Another assumption is that the school will have allocated rooms for detentions outside of school hours. Finally, I will make the assumption that the schools will have their own network which the database of the program will be stored on. This means that any user with access to the network will be able to access the database from their own devices and therefore use the program accordingly. Since I don't have access to any school networks, I have to develop a prototype, working under the assumption that each school would be able to integrate the program into their system in the event that they were to use the final version. Although I understand each school functions differently, I must make these assumptions in order to create a solution.

Why the problem is amenable to computational approach

One of the reasons that the problem is amenable to a computational approach involves the idea of having to create detention slips. One current solution to this problem is to print out physical paper slips for each detention. At the RGS, three slips are created for each detention. One for the student, one for the head of year and one for the form tutor of the student. Considering there are approximately 200 students in each of the 7 year groups at the RGS, printing out three slips for each detention can be very wasteful. Therefore, it becomes very efficient to use a computational approach, as not all detention slips have to be printed. Many teachers may find it far easier to email the student, or his parents, the slip as opposed to having to hand it to him in person. Not only is this easier, but it is also far less harmful to the environment. All this and it would also be far cheaper if much less money is required to spend on paper. That being said, the option for printing out slips will still be an available by using a computational approach, due to the potential of including printing within the program.

Another reason is related to how the method of only recording details on physical slips makes it near impossible to make references to the slips. This is because for a school that has approximately 1,400 students, it is very difficult to find enough space to store detention slips for any length of time, let alone indefinitely. The most efficient method of storing records right now is by use of a database on a computer system. Huge amounts of records may be stored electronically, taking up very little room, as well as allowing the ability to easily copy, send and delete where necessary. Considering this, the only effective approach to this problem is by use of computers. If the schools did decide to get a large enough space for long-term storage of detention slips, not only would it be highly costly, but it would also be time-consuming to manually search for a slip without the use of a computer. On

top of this, the records of detentions would only be accessible on-site, unlike in my proposed computational solution which would potentially allow remote access by use of the internet.

The next reason comes from the statistical analysis of the created records within the database. Initially this would only be possible if a computational approach had been used to create the database in the first place. Aside from this however, any calculations will be significantly faster when automated by a computer. Although possible for a hand-drawn graph to be made using data from the records, there would be multiple downsides to it. The first being that it would take far more time for a human to draw a graph than a computer, and even then it would be prone to having errors. The next is that once the copy is created, it is only available physically. If the user wanted to send this to anyone, create a copy or store it electronically then they would need a computer regardless, as well as a scanner in order to convert it to a digital copy.

Another point about the storing of records without the use of a computer system, is how difficult it is to navigate through. If they are stored physically, then they would most likely be stored by student's name alphabetically, as this would make searching for student's detentions the easiest. However, this would make it very difficult to search for detentions by year group, for example. This is since you can only order things in one way at a time. However, using a database system, such as SQL, you can use search queries to find records using any order you want. This makes viewing records far easier and quicker. A simple "SELECT" query in SQL would let me return all the detention forms related to a specific year group, for example.

The final reason that a computational method is highly amenable is due to how comparisons must be made regarding times and dates of detentions. This is due to how a problem with the current system is that there is no method of preventing students and rooms from being booked multiple times on the same date and at the same time. This creates problems where students don't show up to detentions, as they can only attend one detention at a time. My proposed solution would potentially be able to flag up detention records that overlap for students and rooms. This is easy for a computer as only an "if" statement is required to determine whether detentions would overlap or not. The same cannot be said for a non-computational method, which would involve having someone manually check all the records before each detention is booked, to identify if any would be overlapping or not. This point alone is enough to justify why a computational approach would be the best.

Stakeholders

Identifying stakeholders

As I am using RGS as a case study, which is a school that I have frequent contact with, I thought it appropriate to approach various staff and students from this school as they may have an interest in my proposed solution. As many of them were, I have been able to create a group of stakeholders.

The interest had by staff is quite self-explanatory, as they will be the users of my program. They will be the people who benefit from the detention print-outs, statistical analysis and storage of detention records. However, they are not the only end users, as although students won't use the program itself, they will still feel the effects of it if it were to be implemented into their school. Thus, I feel it necessary to hear their thoughts and opinions on my proposed solution. For it to be a successful solution, the students would need to approve of it, as well as the teachers.

Two teachers have agreed to work as stakeholders for my development. The first is Tim Allen, who is a computing teacher, as well as a form tutor. This will make him very insightful into how the current system works, and therefore can be improved. The second teacher is Same Hunt, who is an economics teacher. With both teachers providing me with vital information about the problem, it will be easier for me to understand the needs of the users.

Next, I approached a student who is in year 9 at the RGS. His name is Tom Jacobs, and he has multiple years at the school with the current detention system, and so he has a lot of relevant experience.

Finally, I got in contact with Alex Brown, who is the head of IT at RGS. This makes him very relevant to the program, as he would oversee installation and ensuring It runs properly were it to be integrated into the school.

Communicating with stakeholders

Over the period of development, I will be using interviews to communicate with my stakeholders. This is because I can design the questions however I please, asking specific or general questions where necessary. I can also change interviews slightly as I go on. For example, if someone responds to one of my questions, I can ask further questions about their response to gain a better idea of their needs. I can also ask questions catered to each stakeholder, as their inputs will be different.

First interview with stakeholders

Interview with Tim Allen (form tutor and computing teacher)

What are the issues that you see in the current system of detentions at the RGS?

"Well I think it's a standout issue that there is no real way of stopping students getting double booked on the same date. I've had a lot of detentions where a student has not turned up and I later find out that it was because they had already had a detention from another member of staff."

How important is it that there is security for a program of this nature?

"It isn't life changing information that would be stored, however that doesn't mean it isn't better for it to be kept private. If you didn't have security, then it means anyone could potentially use your program to book someone a detention. Be it a student playing a prank on a peer, so I do think security is important."

What security features would you suggest in that case?

"I would like to see an administrator account in this program, that allows users to reset their passwords safely for example. Or allows new teachers to make an account on this system, securely."

Are there any specific features you would find to be beneficial for this program?

"Yes, I have always thought the idea of a personal reminder for something like this would be highly useful. I often forget about detentions that I have issued students, and if say I were to get an email reminder on the day of a detention I've issued I think it would be very useful."

Interview with Sam Hunt (economics teacher)

What are the issues that you see in the current system of detentions at the RGS?

"I think the biggest issue is that currently not enough members of staff have access to which students have however many detentions. The reason this is an issue is teachers like myself have no idea if a student is simply misbehaving in their class or are just misbehaving in general for example. I also want to make a point that sometimes parents are too easily left in the dark about a child's detention, since the student is the only one who ultimately has to inform their parents."

What features of a graphical user interface do you think this program requires?

"I have quite a few. Firstly, since you want this to be as accessible as possible, I would say use as little colour as possible so that colour blind users won't struggle to read. On top of that, may teachers have to wear glasses due to poor eyesight, if you use large fonts then they will not have to use their glasses to use the program. I would also suggest using small windows in the GUI so that teachers can have other applications open while using your program."

Are there any specific features you would find to be beneficial for this program?

"I think it would be great if your program could carry out statistical analysis of sorts to create graphs for example, with the data that is recorded in your program. This would let teachers find out which students have the most detentions or something along those lines."

Interview with Tom Jacobs (year 9 student)

What are the issues that you see in the current system of detentions at the RGS?

"Sometimes when I get set a detention, I turn up to find that my detention has been moved to another room because the initial room was already in use. I think there should be a way to avoid things like that since it just wastes my time."

Are there any specific features you would find to be beneficial for this program?

"Yeah, sometimes when I have a detention with another student and they don't turn up, they get away with it. I think it would be good to have a way to automatically give a student another detention, possibly of higher severity, if they don't turn up to the initial one."

Is there anything else you would like to add to that feature?

"It might also be worthwhile to have the program record how many detentions they don't turn up to. Then If they tried to explain it by saying something like 'I was ill' then it would become pretty obvious if they had already missed a lot of detentions."

Interview with Alex Brown (head of IT)

What are the issues that you see in the current system of detentions at the RGS?

"Definitely that there is no kind of record kept for detentions. I don't know what the point of issuing detentions is if they are completely forgotten about after they have been carried out."

What is your recommendation for how you would keep records of detentions?

"I think the only way is to use a database on the school's computer systems. That was staff can access the database by their own devices at home if needed. There are a lot of good database engines out there, I know SQLite for example would do the job."

What makes SQLite a good database engine for this task?

"It's the most used database engine in the world if I remember. That means that most people are going to be familiar with if they have ever had to work with databases. More importantly though, it runs on any operating system so there is no limitation to what devices could use it. That kind of builds on my point earlier as staff could access it on their own devices whether it be a Windows device, Apple or anything else."

Are there any features that would make your job of managing this system easier were it to be installed?

"Yes. As the installation phase of using this kind of software can be quite difficult, it would be far easier if the new system had an effective method of importing current records to the new store. For example, instead of having to manually input all the individual students with their relevant details into the new potential database that is provided by your program, it would be far easier if there was an automated way, using current stores."

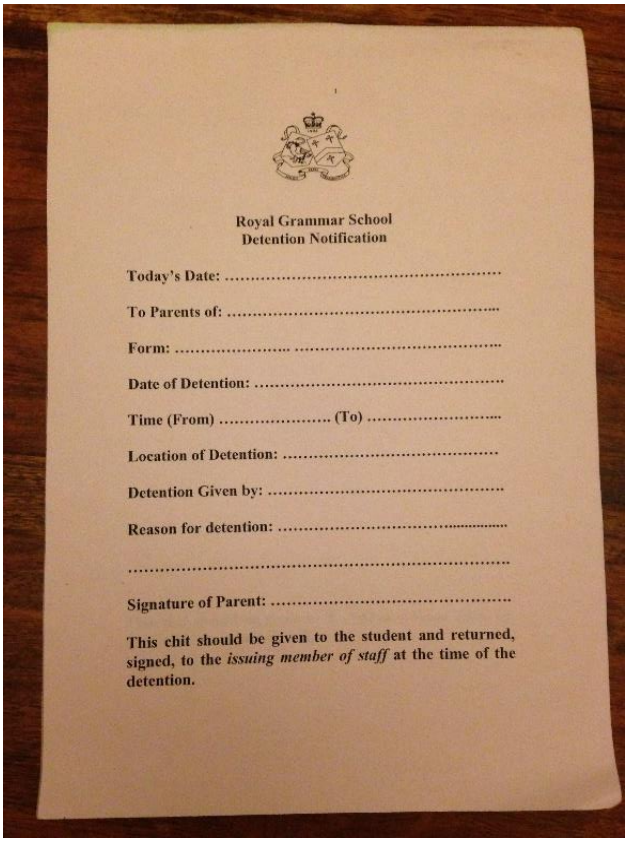
Researching the problem

Researching similar solutions

Royal Grammar School High Wycombe

As talked about earlier, the Royal Grammar School currently has its own methods towards the current problem of booking detentions for students. Although many problems with this system were identified, it still has a solution nonetheless. As a result, I believe it necessary and beneficial to have an analysis of how it's current system works.

The picture on the right is of a typical detention slip that is used at the RGS. As shown, it is simply a template which will always be printed the same, but then the issuer will fill in the details as required. The idea of having a form is quite efficient, as it is easy to work with. Everyone will be familiar with it due to it being the standard, and due to their being limited fields to fill in, there will be no mistakes. This makes it very user friendly, and so as far as simply notifying a student of a detention, I believe it does its job very well. However, due to it being a physical slip, it falls short of being efficient, as it must be given to the student in person. As well as this, the prior mentioned faults of it being wasteful and incredibly hard to store long-term still very much apply. That being said, as long as the form is worked with in a digital form, I think it could be very efficient.



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.

The next part of the school's system involves how it records the details of any given detention. As previously discussed, there is no long-term storage of any details of detentions at all. However, if there were to be storage, of slips, they would have information of various fields relating to the detention, as shown in the slip. For example: they would have the form, date of detention, time and more. The only thing missing from the physical slip which would be required for storage in a database would be a primary key. In this case, it would be a detention ID. Because of this, the slip shown above is likely going to represent how a detention record would look in a database, once it has been filled in.

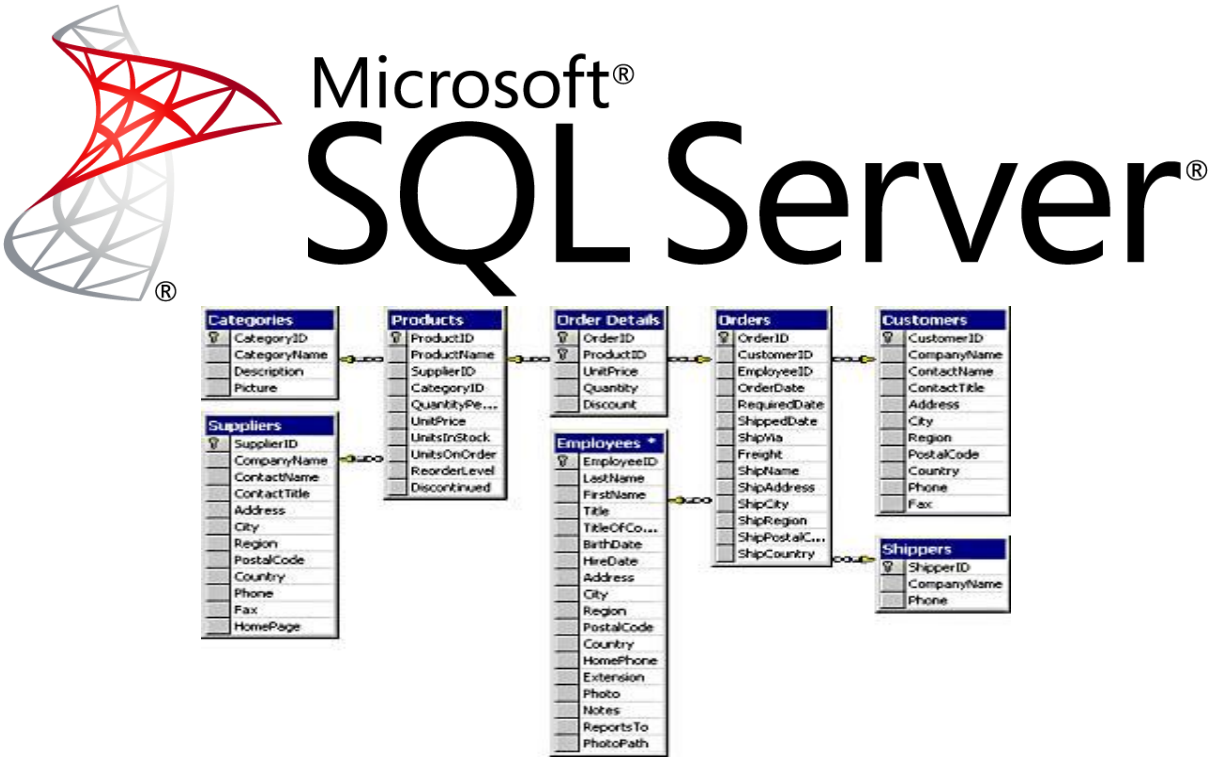
A final aspect to consider of the RGS's detention system is its usage of locations for detentions. Typically, a room is used for a detention that is thought to be vacant by the issuer of the detention. Although the idea of using rooms which are free is good, in practice, rooms may often be "booked" by multiple people at a given time. In this event, the detention must be moved to another room at the time. I would like to take a similar approach of ensuring there are empty rooms, but since I am using a computational approach, it will be possible to reliably ensure that any room is free.

As a conclusion of the RGS methods of detentions, I would say that although it is very flawed in execution, it has a lot of points which can be used as a foundation to be built upon for a proposed solution. For example, by using the same base form, but using it as a digital version. By using the same fields of information that their standard detention forms record, but storing the in a database. And finally, to use their method of booking rooms for detentions, but also applying a more reliable method of insuring that rooms are not overbooked.

Student Information Management System (SIMS)

Prior to Philip Neal's creation of SIMS in 1984, schools had no reliable and recognised computational method of managing their information. A combination of the fact that 80% of schools in the UK currently use SIMS[1] and that it was awarded the "innovation award in the Education Resources Awards" in 2012, suggests that the solution was a great success. Due to the how nature of the problem that SIMS set out to solve is so similar to the problem that I am faced with, I will be considering its key aspects.

After carrying out research into SIMS on the internet[2], I have been able to identify how it handles its storage functionality. SIMS uses Microsoft SQL Server (SQL), a relational database management system, in order to handle all of its storage of records.

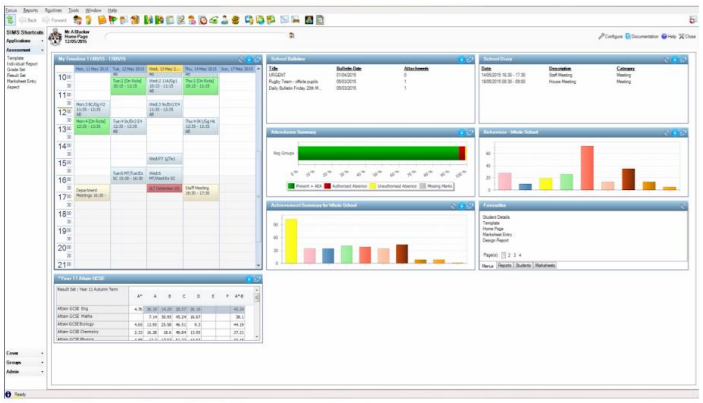


The reason for this is firstly because it simply must have a database system which can be large enough to support a whole school's worth of data, as well as being accessible from a network, as opposed to from a single device. SQL is very versatile, which allows it to be used in many ways as the user sees fit. It conforms to the ACID rules (the four primary attributes ensured to any transaction), which makes it reliable in that all transactions will be managed properly.

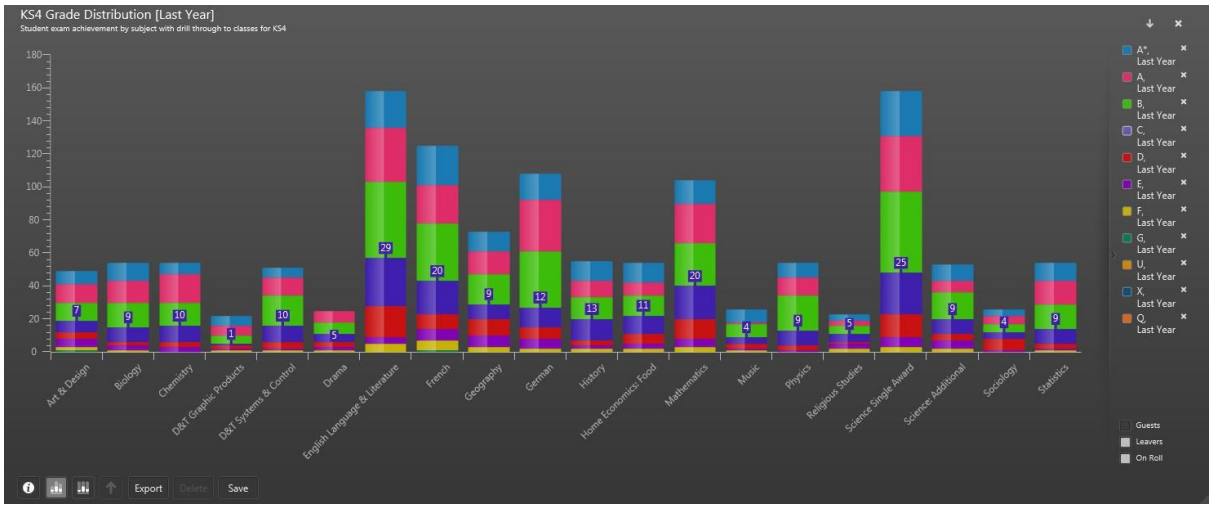
This is vital for a system which holds such personal data. Due to the data protection act of 1998[3][4], companies such as SIMS are held accountable for any inaccuracies in data. This makes it a very high priority to take data management seriously. SQL allows users of SIMS to easily navigate through their

data, regardless of how big the store may be. It is very user friendly, as it was originally designed to be used by people without expert knowledge of computer systems. Although it does its job very efficiently, the specific version that is used by SIMS is not appropriate for my project, as it is very pricey. However, there are many free alternatives that exist in the public domain which would suffice. For example, SQLite and SQL Express. I will be considering these alternatives for when I decide what to use for the database of my program. Another reason that I would not use Microsoft SQL server is that it is only compatible with Microsoft Windows at this point in time. Alternatives like the prior mentioned SQLite however, are compatible with all popular operating systems.

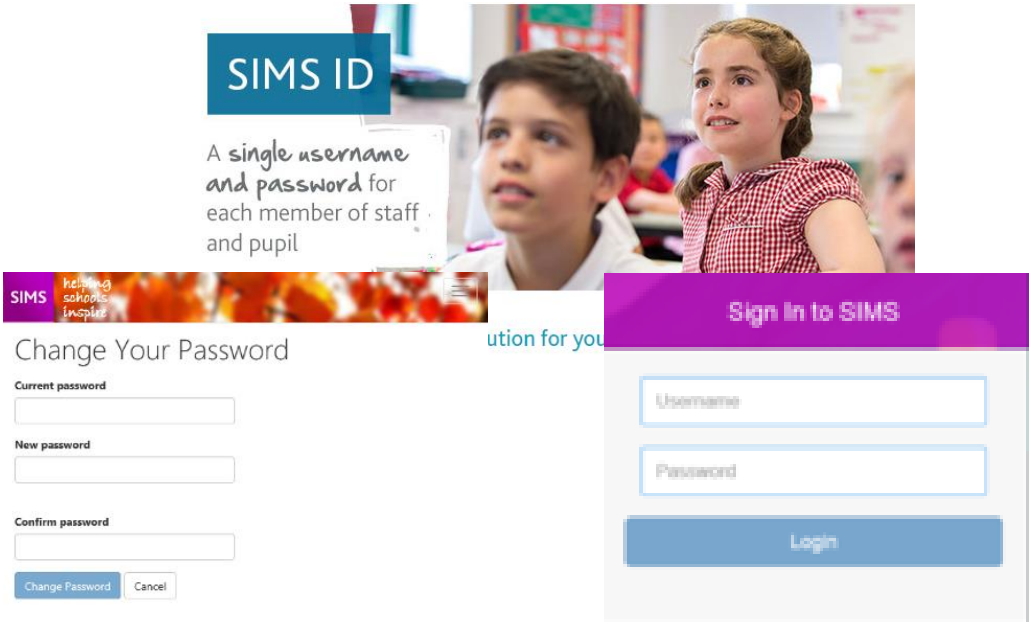
A big part of SIMS is “SIMS Discover”. This allows graphs to be created with data obtained from students results. For example, it can display graphs as shown in the image below. I think this is a very useful feature, as data analysis can be a very involved process that most people are not familiar with. My end users are not experts in this field, and so if my program is able to visually show trends to them, it will make it far easier for them to make sense of what the data recorded actually shows. SIMS will not be able to plot graphs based on detention records however, as they do not record this kind of information exactly as I plan to. However, despite the nature of the data being recorded in my program and SIMS being different, I still plan on showing graphical analysis in a similar way.



Below is an example of a graph produced by SIMS Discover. It shows trends in grades achieved by KS4 students. Once again, although the nature of data is different, the same principle of visually representing data with graphs will still apply.



The next feature of SIMS that I find to be very relevant to my project is “SIMS ID”. This is a system which assigns a username and password to each user, so that they may securely use the program.



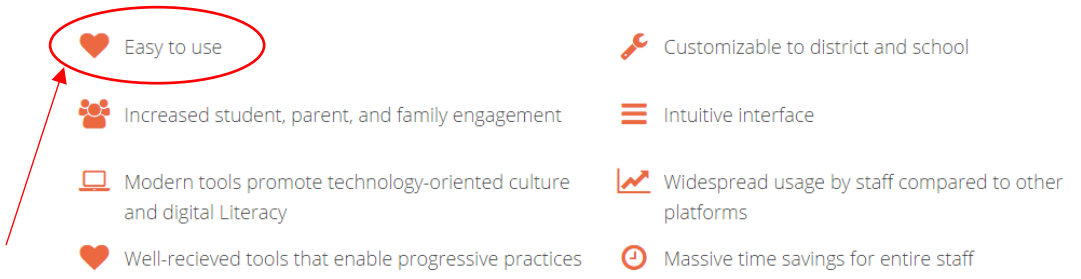
This is essential for a program such as mine and SIMS, as personal, confidential data is stored on different people. For example, email addresses and health records may be stored for different teachers and students. As previously mentioned, due to the data protection act of 1998, it is essential that data like this is kept secure. If anyone were to be able to access the system, then they would be able to view any records that they want to within the database. This method of having different users of the system also gives rise to the possibility of having users with different levels of authorisation. For example, administrative users may exist, which have more rights than standard users. Other features must also be included when there are secure login systems, such as the ability to change passwords. If a user’s password were to become compromised, then the only solution is to change their password. Therefore, without the ability to change passwords, a login system can never be truly secure. Other examples may include features such as password resets, which are all available in a program such as SIMS. As a result, these are definitely features which I will plan on integrating into my project.

Alma

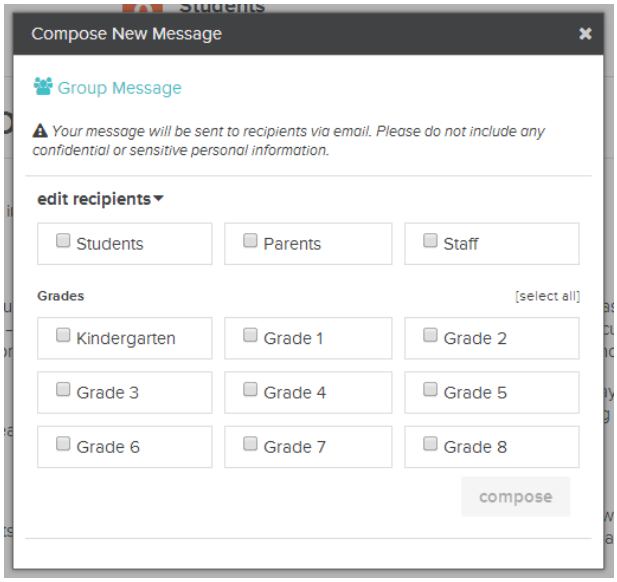
Alma is another example of an MIS (management information system) used specifically for students within schools[5].

I have chosen to investigate this program, as it advertises itself on being simplistic and easy to use. As teachers are not typically experts in the subject of computing and IT, I think it is very important for computer systems to be very self-explanatory and easy to use. Therefore, I will be looking specifically at Alma's GUI and design to understand what makes it easy to work with.

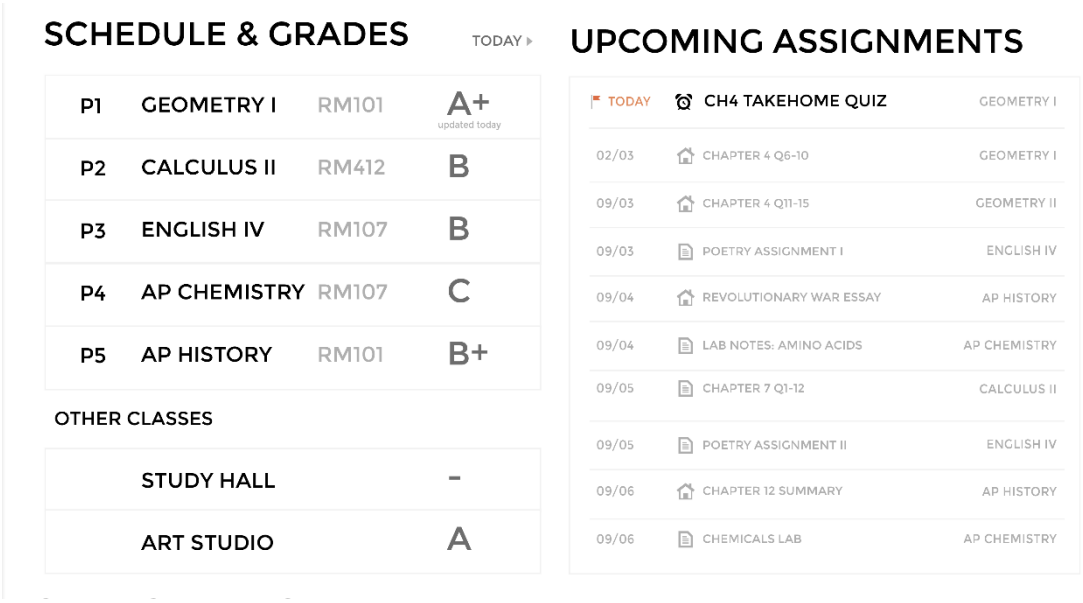
By analysing the below screenshots, I will be able to conclude why Alma is able to declare that it is easy to use and simplistic, as stated on their site and in reviews. The first image shows a pop-up that appears when the user sates that they want to compose a message. It then prompts the user to choose the recipients. As shown in the image, very few colours are used. Only blue is used at the top, other than that, it is all black and white. This is most likely due to accessibility, as colour-blind users will not struggle to see what is being shown in the GUI. Other than that, very little text is being



shown. This is because it makes the window less over-whelming, and thus the user finds it easier to make sense of. I plan on incorporating these styles into my program, as our end users are very similar.



The next image shows a window which displays grades for different subjects, and some upcoming assignments. Much like the first window, there is little to no colour present at all on the window. As well as this, the font very read-able due to how it is relatively large and clearly printed. I also plan on implementing this style of text in my program, as then it will be more accessable and easier to work with.



Essential features of proposed solution

Based on the initial interview, and research into solutions of similar problems, I have been able to identify the essential features of my proposed solution.

Firstly, as previously mentioned, the solution absolutely must be able to produce digital detention forms, similar to the physical ones currently used by the RGS. The detention form will still give the option to be printed, as well as the option to email it themselves or simply save it to their device. Like Alma's approach to design, I will attempt to make the design of this form very simple, with few colours and easily readable font.

This approach to design will not simply apply to the detention form however, as I will attempt to keep this philosophy of simple design throughout every part of the graphical user interface that I will develop. I have decided that I will use a graphical user interface as my approach to what the user will interact with. This is justified by how my end users will generally not be that confident with computer systems, and if they are, they still may not be educated into the subject to the point that they would feel confident using a command line interface. Despite the fact that a command line interface would provide more control to the user, it would simply make many unable to use it. Whereas a GUI approach is very accessible, especially if combined with the simple design that I plan to implement as based upon Alma's approach. Giving a specific example of how I may apply a GUI approach would be using a form for the user to give inputs for a detentions details. By use of options such as combo boxes and dials, it removes any possibility for the user to run into any kind of errors when inputting data. This is massively beneficial, as users find it very frustrating when they run into

errors, especially if there is no explanation for the error. Even if an error is created in my GUI-based solution, I can potentially use pop-up windows to help the user. Not only did I get a lot of this inspiration from researching Alma, Sam Hunt proposed many of these interviews in my interview with him. A further idea that he provided, which I will most likely take on, is using small windows so that the users will be able to have their own windows still in the foreground whilst they are using the program. This is due to how the teachers will often be having to refer to other windows as they are using my program. For example, they may need to refer to their calendar when booking a detention, or be composing an email regarding a student's behaviour, whilst they are referring to detention stats that are produced by my program.

I will attempt to have an approach which utilizes a relational database management system in order to store all the records of detentions, and the relevant information of students and teachers alike. The records for details of users will also have to be kept. This idea was one that was based upon the idea that SIMS successfully uses Microsoft Structured Query Language for this purpose, as well as Tim Allen suggesting that SQL be used for this purpose within his interview.

Similar to "Sims Discover" in SIMS, I will be including procedures within my program that allow teachers to create graphs based on chosen criteria, for example students or year groups. This is because detentions are used as a method of preventing bad behaviour, and thus it is important to be able to identify where the problems are so that focused improvements can be made. As justified earlier, the easiest way to create graphs is by computational methods. This was another point that was emphasized by Sam Hunt in my interview. Graphs will not be the only output however, as the user may only need a simpler analysis of the data to be output. For example, a simple 1-line statement of year group which has the most detentions overall.

Another part of SIMS which my program will also contain is a secure login method, similar to "SIMS ID". This idea was brought to my attention by Tim Allen in his interview, due to how he feels data regarding a student's behaviour should not be open to everyone, especially not other students. I strongly agree with this, and so it will definitely be implemented. Much like in SIMS ID, I will keep it a single ID method, where each user only has one account, meaning they only have one username and one password to worry about. I will use different methods of validation in order to ensure passwords are kept valid as well as complex, as to increase security. As again suggested by Tim Allen, I will also attempt to include methods of both changing and, in the event of a password being forgotten, being reset. Different types of users, such as administrative users, will also exist in order to provide more privileges within the system. These too are all features that appear within SIMS ID. Another note to be made about this system, is that due to it being a computational solution, it could potentially be at risk to a cyber-attack of sorts. Although this seems far-fetched, it is definitely worth considering, and therefore I will ensure that passwords are not stored in their raw forms, but in hashed form instead. This means that unless the decryption, the hashed passwords and the user names are known, the accounts will not be able to be breached with malicious intent. Again, although this seems like an unnecessary precaution as the chances my system would be any kind of target are near to none, due to there being personal details potentially being stored within the database, I feel it is a necessary inclusion to store hashed passwords.

Tom Jacobs suggested in his interview that there should be consideration for student's attendance of detentions. I support this suggestion, as my system could potentially highlight students that often skip detentions. As this is an offense of school policies it should be dealt with. This offense is worthy of a detention itself, and so not enforcing the detentions would be very counter-productive.

A final essential feature to be mentioned is the ability for the program to check given rooms and students for whether they are free or not at the specified times and dates. I understand this is very important, as Tim Allen mentioned that student's often get double booked within the current system at the RGS. I could prevent this by including procedures that automatically validate detentions when they are being booked. It was Tom Jacobs who had mentioned that his detentions often got moved to other rooms as they had also been double booked, hence why I will include methods of checking room vacancies as well.

Limitations of proposed solution

Due to this being a solo project involving only myself, there will be limitations to what I can achieve. A combination of me being a novice at software development with me being a full-time student means that I must forgo potentially features of the program which could potentially be beneficial. Although the users and my stakeholders may never see these features, I see it necessary to discuss them so that they understand what could potentially be included in the future, be it by me or someone else who continuous to work on my project prior to its competition.

Firstly, despite having the store of detention records be available through a network, it will simply not be achievable at this time. This was mentioned earlier under the heading of problem identification, as my solution to the problem will simply be a prototype. Although it is a complete requirement of this solution for teachers to be able to access the database from their own devices, and therefore update it, due to the nature of every school network, assuming they have one, being different, it is an inefficient use of time to attempt to recreate the desired effect of having the program work over a network. However, if anyone were to implement this system into their school, this process would have to occur, and so it is worth mentioning that in the full version of this proposed solution, this would be a significant feature of the program. But once again, I will not be achieving this for the prototype. It will only work locally on a single given device, due to my lack of time and expertise.

The next feature that will not make it into the final version of this prototype, is the ability to automatically import any already existing school information into the database. This idea was brought to my attention by Alex Brown, and I completely agreed with his reasoning of why it would be useful. As previously mentioned, the RGS has approximately 1400 students. It would take a very long time to manually insert 1400 individual records of students in order to populate the database of my proposed solution. However, populating the database is mandatory for the program to work. The reason I cannot create this future of automatically importing current school records into the database, is similar to the network problem. It is because each school will have a different database system that manages the school's information. For example, the RGS has each year group split into forms, however, many schools may not work like this. Therefore, the program would have to have a different set of fields within its database, depending on how the school itself works. These reasons are the primary contributors that have led me to make this project as a prototype, because for instillation in each different school, it would require adaptation.

Finally, another feature that I do not plan on including, despite it again being potentially beneficially, is the automated email system suggested by Tim Allen. The reason I have not decided to implement this in this prototype is simply because I do not see it is as worthwhile enough, despite the simplicity of the task. I could very well achieve this by using already existing modules within python that others

have made to achieve this task. However, in the modern day, everyone who has access to popular email providers, such as Outlook or Gmail, has access to free calendars and reminders. I do not think I would be adding anything worthwhile to this program by adding automated emails, and so I do not think it is worth spending my limited time on. However, because it is a relatively simple task, it may easily be added to the prototype post completion, if individuals were to want it themselves. My solution is being designed in order to provide a workaround for an existing issue that does not currently have a popular solution to it. Reminders however, is already a heavily visited problem which has many solutions in its own right.

Proposed solution

Requirements

In order to run my proposed solution on a given device, you will need to have the portable python 3.2.5.1 software pack[6], which includes python version 3.2.5[7] and PyQt[8] version 4.8.4. The download for this pack may be found in the bibliography section of this document.

As well as this, SQLite Version 3.9.2[9] must also be installed. All the prior mentioned software should be available to work with most major operating systems, such as Windows 7 or later, MAC OS and Linux.

Hardware requirements are very basic, as this program will not be demanding. However, basic peripherals, such as a monitor and a keyboard are a necessity to give input and view output. The monitor resolution size recommendation is 1366x768 or above, though if a smaller monitor is used, it will be an inconvenience to the user at most. Specs of a given device however would be the following: A minimum of 4GB RAM, a hard drive of varying size (purely dependant on how large the schools database would be, though a recommendation of at least 1GB), and a CPU with a recommended dual core 2GHz processor or better, though the programs performance will diminish if the processor is any slower.

Success criteria

If I wish to declare my solution to be successful, I must present a list of measurable success criteria to my stakeholders, which, will serve as a list of objectives for me to complete. I will attempt to complete all these objectives by the end of the development process, and then be sure to justify that I have achieved each individual objective. I may do this using screen shots the serve as proof of an objective being completed.

1. Program will determine if files are successfully located on the device, and if not, the user is alerted of this and advised to re-attempt installation of program
2. The user will only be able to access the system if they have an account
3. User will be prompted to enter login credentials…
4. …SQL statements will be used to determine validity of login credentials…
5. …pop-up window will alert user if the specified user doesn't exist…

6. …or if the password is wrong
7. A sign out button will allow the user to securely log out of their account
8. Passwords of users will be stored as a hash within the database, to ensure security
9. Regular expressions will be used to enforce password security, both in which characters may be used…
10. …and the length of a password and complexity of the password
11. Users will be able to change their passwords once logged in, at any time
12. A help button will provide users with an admin's email so that they may get help logging in
13. Administrative users will exist, which have more privileges than standard users
14. Admins will be able to reset other user's passwords "123", for if they forget theirs
15. Admins will be able to upgrade other accounts to administrative users
16. Admins will be able to create new users…
17. …specifying their account type and adding their details to the database…
18. …with a completely unique user ID…
19. …and a hashed version of "123" as their password
20. Windows will be small, requiring no scrolling and never occupying more than half of the screens space, provided they have the recommended screen resolution of 1366x768 or higher.
21. Interface will use very little colours, limited to white, grey, black and light blue, outside of the RGS placeholder logos, to cater for colour-blind users
22. Back buttons will allow users to return to their previous window
23. Pop-ups will be used to give information to users, as they are not miss able
24. Minimal clutter will be enforced by limiting number of buttons per window to 5…
25. …and maximum number of widgets per window to 9
26. Reduced amount of user typing will be enforced by only *requiring* 1 word inputs at most (they may give longer inputs if they view it as necessary), and text displayed to the user will also be minimal, to further increase simplicity to the user
27. Frames will be used to create a "sunken" effect, grouping widgets together, further reducing clutter
28. Once logged in, only 2 clicks at most will be required to visit any other window to reduce complexity to the user
29. Combo boxes will be used where possible, to reduce the possibility of user error…
30. …if combo boxes not applicable, line edits will be used, however validation will be used to ensure valid inputs are used
31. A "manage users" window will only be accessible to administrative users, which allows management of users and creation of new users
32. SQL database will have no many-many relationships that cause data redundancy.
33. Database will record all relevant details of teachers in a table…
34. …students in a table…
35. …form groups in a table…
36. …detention rooms in a table…
37. …and detentions in a table
38. Program will allow create, read, update and delete statements to be carried out on the database
39. Read statements (SELECT), will be used during user authentication…
40. …and creation of any kind of output that requires data from the database
41. Update statements (UPDATE, INSERT) will be used when creating new users…
42. …changing passwords…

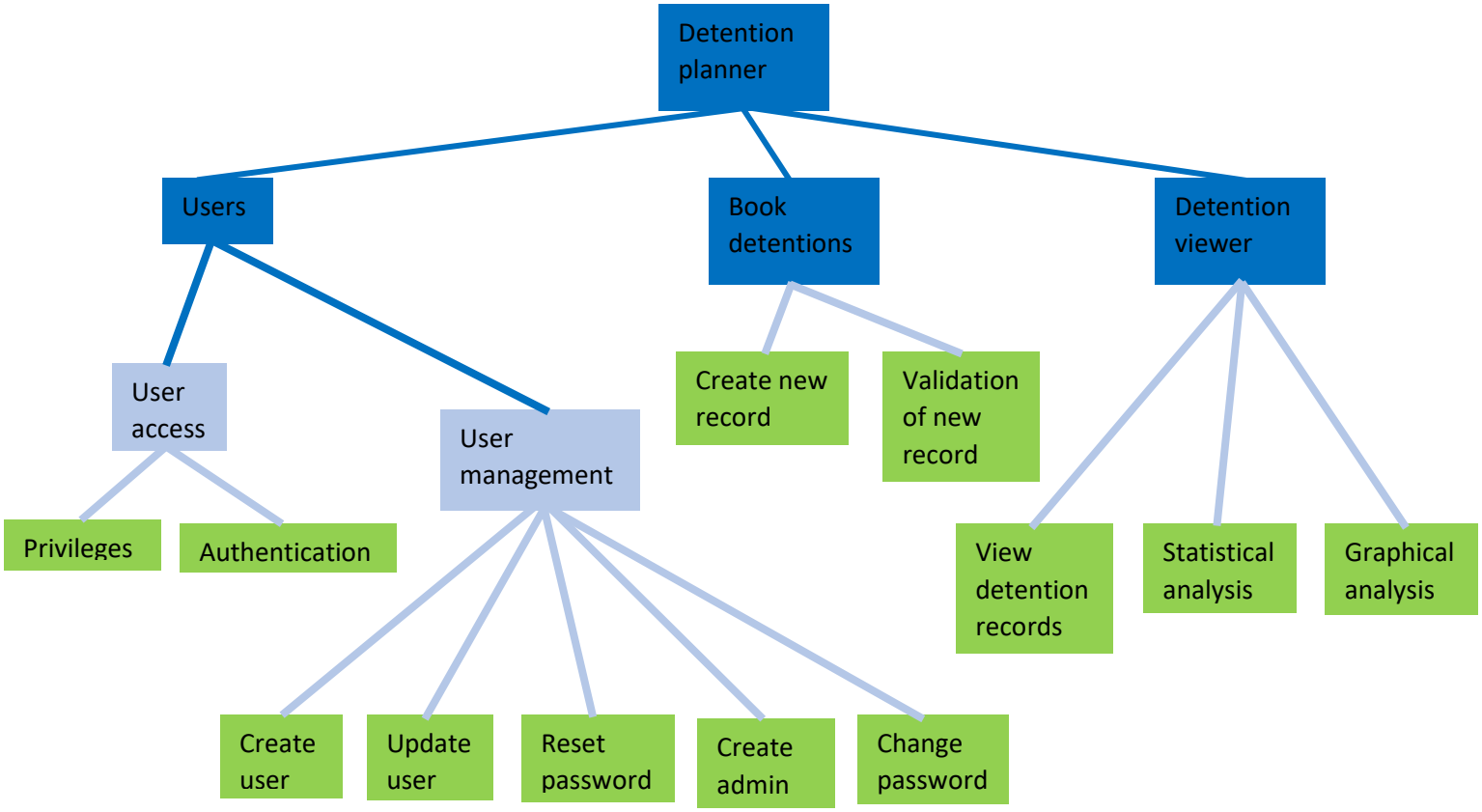
43. …resetting passwords…
44. …creating administrative users…
45. …creating a new detention record…
46. …which will also increment the offending student's detention total
47. Delete statements will allow deletion of detention records…
48. …and thus, the removal of the detention from the student's record as well
49. When a detention is booked, a form will be created using HTML, CSS and JavaScript…
50. …which has been designed to replicate the physical form currently used by the RGS…
51. …which may then be saved as an XML file…
52. …to be emailed…
53. …or printed out depending on the user's preference
54. Graphs may be created to display data from the database regarding detentions…
55. …which will be created by the program using HTML, CSS and JavaScript…
56. …the option to print out these graphs will also be available in the same way as the detention forms
57. Simple statistical analysis statements may be created to give users an analysis of detention data…
58. …such as "student x has the most detentions" …
59. …or calculating percentages where applicable
60. A raw view of the database will be accessible to the user

Design of the solution

Decomposing the problem

To plan my solution to the problem, I will need to use abstraction and decomposition to break down the problem into smaller parts that will make a computational approach possible.

Initially I have created a top down design model, so that I may show an abstract plan of what my program will do, with the lower down boxes representing tasks that will closely resemble individual sub-routines. I will go into detail of these sub-routines later, using pseudocode to show how each individual task may be achieved. This diagram may not accurately resemble everything that the solution shall be able to achieve, but it gives a good general idea, and thus will help when planning my approach to the problem.



As shown in the diagram above, there are three different main sections involved in this solution, which are users, detention bookings and detention viewings. Detention viewings is a vague term, but what it represents is the viewing of detention records, be it as a graph, a statement of statistical analysis, or a raw database view. From a glance at this diagram, it may be concluded that the majority of this solution is revolved around a database. For example, everything to do with users is completely related to the database, as all their credentials are stored there within. Similarly, booking detentions is essentially just creating a new record within the database, with some validation.

Finally, detention viewing is simply reading data from the database. However, this is a very abstract view of the solution, but from this top down design model, I have been able to decide that the most appropriate approach is to start with the database.

For me to create a database, I must fully understand what data I will be needing to keep. Thanks to my analysis of the RGS's current system, I understand exactly what I will need to store. Later, I will show how I plan on developing my database, using an ERD (entity relationship diagram) to show exactly how it will work. For now, I am just deciding in what order I will approach all my tasks when it comes to development.

Another aspect which my program is hugely dependant on is the UI (user interface). As discussed in my analysis, this is due to the nature of my end users, therefore I must build my solution around this aspect. The top down design model's different boxes may be each represented by an individual window if I were to view it as a GUI (graphical user interface). For example, user access would simply be a login screen. All the boxes underneath user management may be viewed as windows as well. Then, a single window would be necessary for both detention bookings and detention viewings. Once I have designed my GUI with all the widgets which will allow for functionality, it will be easy to develop my solution so that the initial skeleton GUI will become a functional program. For example, if I create a rough design for a login interface, then it will simply become a matter of writing the code to make this GUI work, considering I will already have the database fully developed by this point.

Once I have completed both the database and the GUI, the next logical step would be to write the code that will make the GUI function. In other words, writing the code to achieve functionality for each individual box. I will start with the "users" section, as this should be the simplest part of the program. Once I have achieved all the sub-tasks related to "users", I shall move onto detention bookings. I choose this before detention viewings, since the records required for detention viewing will only be existent if detention booking is functional. i.e. you can book detentions without being able to analyse detention records, but you cannot analyse detention records if none are able to be created.

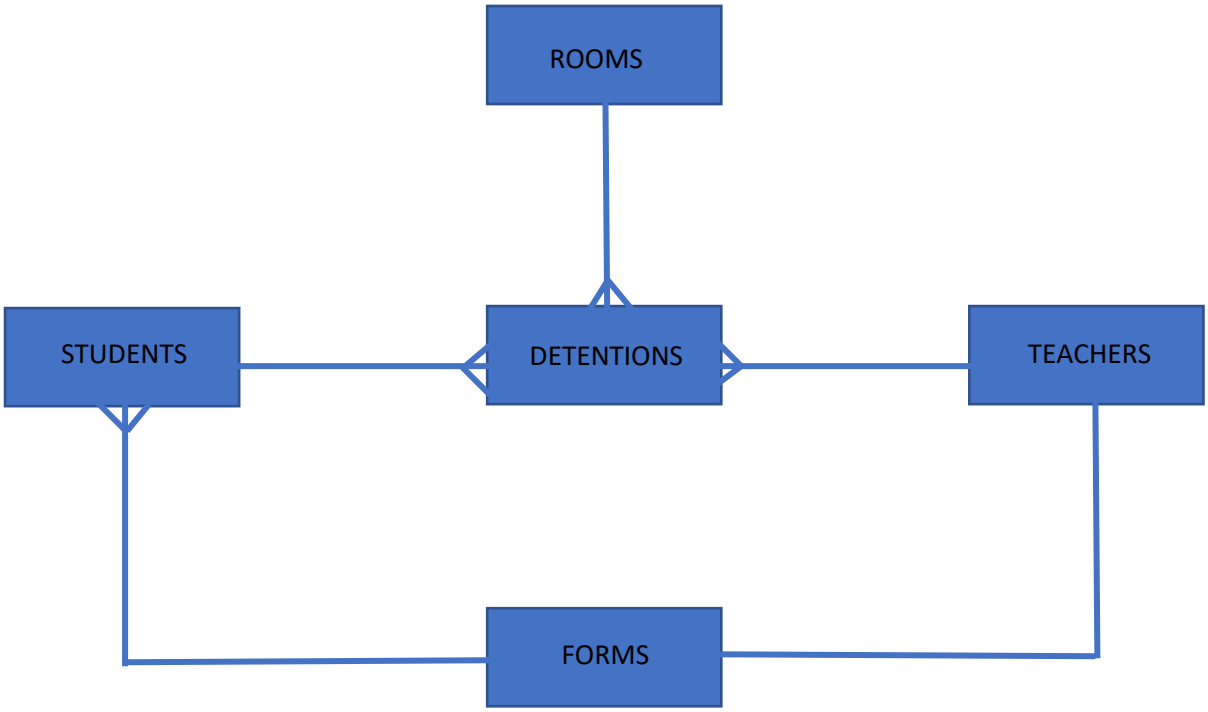
Describing the solution

Data structure

Initially, I will plan the relevant data structure that my solution will use. As discussed in the decomposition of the problem, this is the first logical step towards creating my solution.

All data requiring long-term storage will be saved within a database. Although I have decided upon using SQLite, I will stick to a general approach that is not specific to any kind of software. The database will have 5 different entities, to ensure it will be fully normalized (i.e. in third normal form) meaning that there will be no data redundancy.

I have created an ERD (Entity Relationship Diagram) to show an overview of the data structure. It will also display how there are no many to many relationships within the database.



I will now show a more specific view of each individual potential entity present within the database, with their respective fields.

Detentions

The first table is the "Detentions" table, which will act as the transaction table. The purpose of this table is to record all detentions and the relevant information surrounding them. Most queries used to search the database will be relevant to this table.

Field	Data type	Purpose
DetentionID	Integer	Serves as the primary key
Date/time start	Integer	Stores start of detention
Date/time finish	Integer	Stores end of detention
RoomID	Integer	Stores location of detention + foreign key
TeacherID	Integer	Stores teacher of detention + foreign key
StudentID	Integer	Stores student of detention + foreign key

Teachers

This table is going to be used to store all relevant information about teachers, while also storing vital data that allows them to have a function account to the program, such as 'TeacherUser', 'Administrator' and 'Password'.

Field	Data type	Purpose
TeacherID	Integer	Serves as the primary key
SecondName	String	Stores second name of teacher
FirstName	String	Stores first name of teacher
TeacherUser	String	Stores unique username of teacher
Administrator	String	Stores whether teacher is an admin
Password	String	Stores hashed password of teacher
Email	String	Stores email address of teacher
FormID	Integer	Stores teacher's form + foreign key

Forms

The "Forms" table will be a small table, which is required to keep the database in third normal form. It will only have 3 fields in it for this reason.

Field	Data type	Purpose
FormID	Integer	Serves as the primary key
FormName	String	Stores the actual name of the form
TeacherID	Integer	Stores form tutor + foreign key

Rooms

This table, like the prior table, will be a small table, used primarily for keeping the database in third normal form.

Field	Data type	Purpose
RoomID	Integer	Serves as the primary key
Block	String	Stores name of building the room is in
RoomNumber	Integer	Stores the room's number

Students

Finally, the "Students" table will record all the relevant data to the students at the school. This table will be used a lot for queries about statistics of which students are receiving detentions.

Field	Data type	Purpose
StudentID	Integer	Serves as the primary key
SecondName	String	Stores the second name of the student
FirstName	String	Stores the first name of the student
CurrentYear	Integer	Stores which school year the student is in
DetentionCount	Integer	Stores amount of detentions received
FormID	Integer	Stores form group of student + foreign key

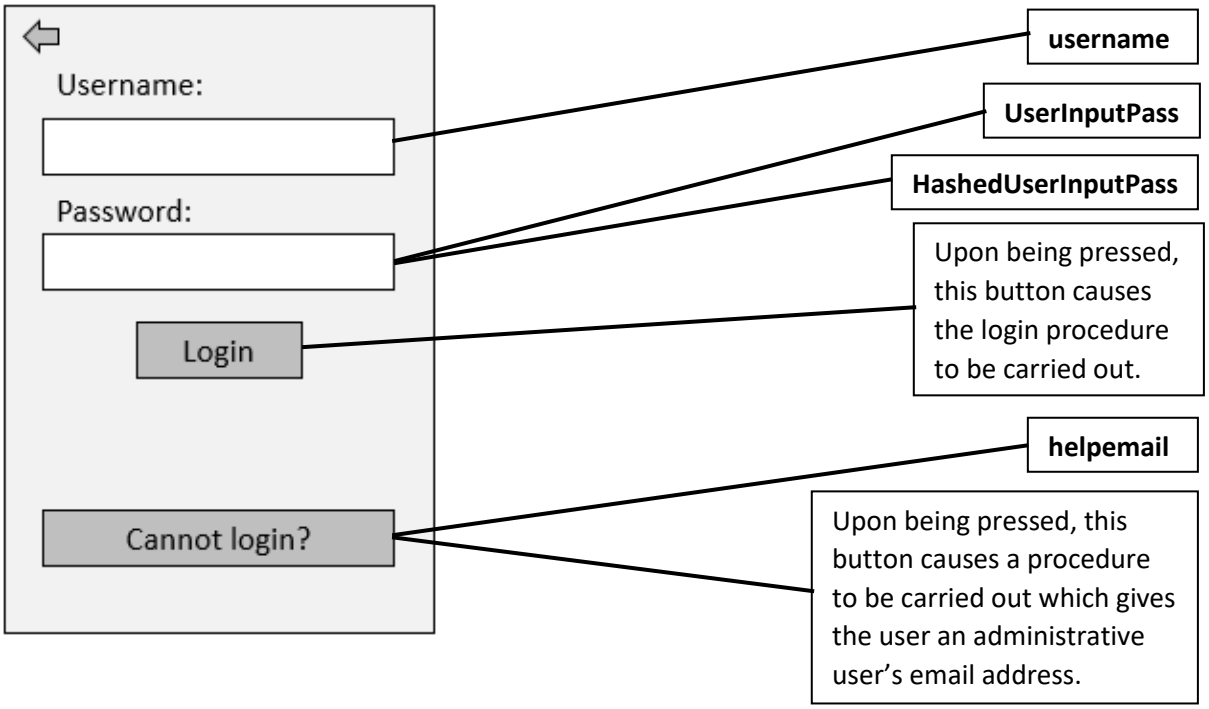
Structure and functionality of solution

To effectively describe the structure of my solution, I will create various UML (Unified Modelling Language) diagrams, one for each window present in my GUI. Specifically, I will be using activity diagrams, as these best represent the functionality of my solution. For each window, I will create a mock-up of what it may look like in the finished version. This is because it will help to describe the structure and functionality of the solution. By creating a rough GUI, it will be easier to understand the relevance of my pseudocode, and where it is going to be relevant. I will create these mock-up GUI windows in Microsoft PowerPoint, by using the shape tools. As different widgets vary between different GUI developing software, I will assume that I have text input boxes, combo boxes, buttons and labels. By use of my top down design model, I have been able to identify 8 different windows that will be required. The windows are as following:

- Login screen
- Main menu
- Change password
- Manage users
- Create user
- Detention booker
- Detention viewer
- Detention form

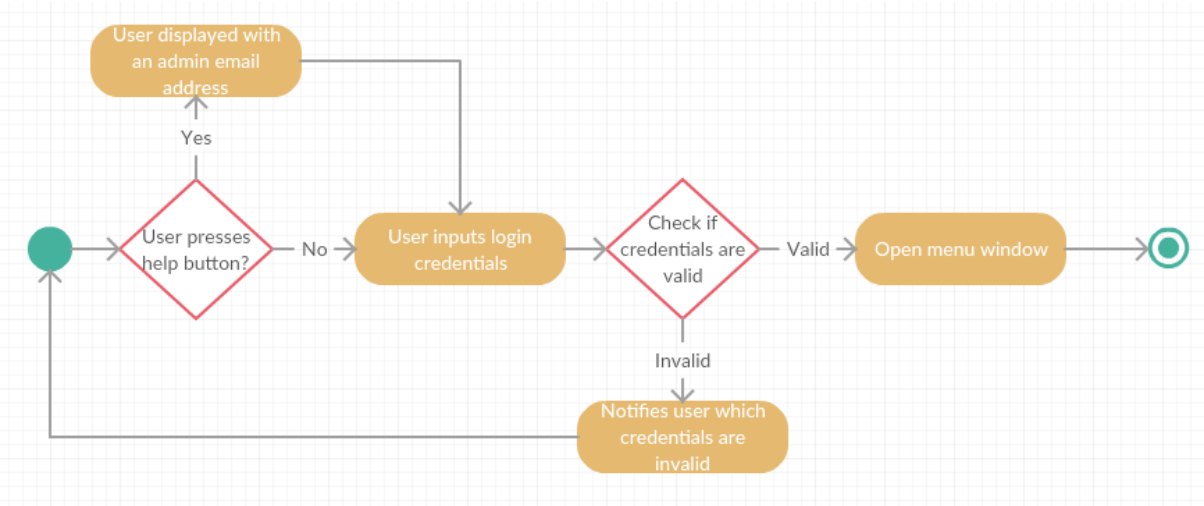
Login window

This screen will be the initial window that the user will see when they login. As stated in my objectives, only holders of accounts should be able to use the program. Therefore, it is paramount that this window is displayed first, so that only after authentication is complete, may the user proceed to use the functionality of the program.



Variable name	Data type	Purpose	Example
username	String	User input of the username of their account	SmithL2
UserInputPass	String	User input of their password	Password123
HashedUserInputPass	String	Hashed version of UserInputPass	Fb2947h9wef9af3823r 293r8hsfa0sdjasdi3asf 93820hjdfgjsccu8io02
Password	String	SQL query grabs the actual hashed version of the user's password as stored in the database	Result of: "SELECT Password FROM Teachers WHERE TeacherUser = ?" Where ? is username.
helpemail	String	Is the result of an SQL query which grabs an administrator's email	Result of: "SELECT Email FROM Teachers WHERE Administrator = 'YES'" e.g. brownb@mail.com

The following UML activity diagram shows the general process of the login procedure, with each box being potentially represented by single algorithms that I will be able to discuss. Although I will create pseudocode which will give a more specific description of what will occur, these diagrams will help show the structure of the solution, as well as identify various usability features.



This diagram will contain two different sub-routines. One for authentication, i.e. checking user credentials, and one for the help button. I will now show how these algorithms may look using pseudocode.

Help email

The following function will be executed when the “cannot login?” button is pressed. The aim of this function is to use a database read statement to find the first administrative user’s email address from the database, and then return it to the user. Although it is a function, it may be viewed as a method for the “LoginWindow” class.

```
01  function HelpEmail()
02        OPEN FILE "TeachersTable"
03        Teachers = FILE.READ("TeachersTable")
04        for i=0 TO LENGTH(Teachers)
05              if Teachers[i,4] = "YES" THEN
06                    return Teachers[i,7]
07              else
08                    pass
09              endif
10        next i
11        print("There are currently no admins.")
12  endfunction
```

As stated earlier, I will be using a general approach to writing pseudocode for database queries, despite being sure I will use SQLite. In code like this, I am treating the database as CSV files. The “TeachersTable” is simply the table shown previously under the data structure section. This function will read from this CSV file, returning a 2D array, which is indexed by “`Teachers[i,j]`” where i is the number of the record, and j is the field. Line 5 uses the j index of 4 due to this being the “Administrator?” field, which keeps a value that determines whether a user is an admin or not, “YES” or “NO”. line 6 uses the j index 7, as this is the field which records the email address of each teacher. If all records are explored and an admin isn’t found (i.e. i reaches the number of records with no match), then the function notifies the user that there are currently no admins. I put this in place as a validation of sorts, however, this should never be the case with usage of the system, as there should always be an admin present.

This algorithm uses the common algorithm of traversing a 2D array, as well as the concept of reading from text files. It also uses elements of a linear search algorithm, as it increments through a list to find a specified result, in this case, the index of a record where Administrator = “YES”.

Authentication

The following algorithm will be another method of the "LoginWindow" class. It will be executed in the event that the "Login" button is pressed. The aim is for the user's credentials to be checks against the database to determine whether the login attempt is valid or not. It will then respond accordingly, by giving them access to the next part of the system, or notifying them what details are incorrect. Although many secure sites choose not to outline which credentials are wrong, I believe it will be more of a convenience than a risk here.

```
01  function Login()
02        global username
03        global password
04        username = input("enter username")
05        UserInputPassword = input("enter password")
06        HashedUserInputPass = HASH(UserInputPassword)
07        OPEN FILE "TeachersTable"
08        Teachers = FILE.READ("TeachersTable")
09        for i=0 TO LENGTH(Teachers)
10              if Teachers[i,3] = username THEN
11                    password = Teachers[i,5]
12                    if HashedUserInputPass = password THEN
13                          return "Login successful"
14                    else
15                          print("Password is incorrect.")
16                          return "Login failed"
17                    endif
18              else
19                    pass
20              endif
21        next i
22        print("Username does not exist.")
23        return "Login failed"
24  endfunction
```

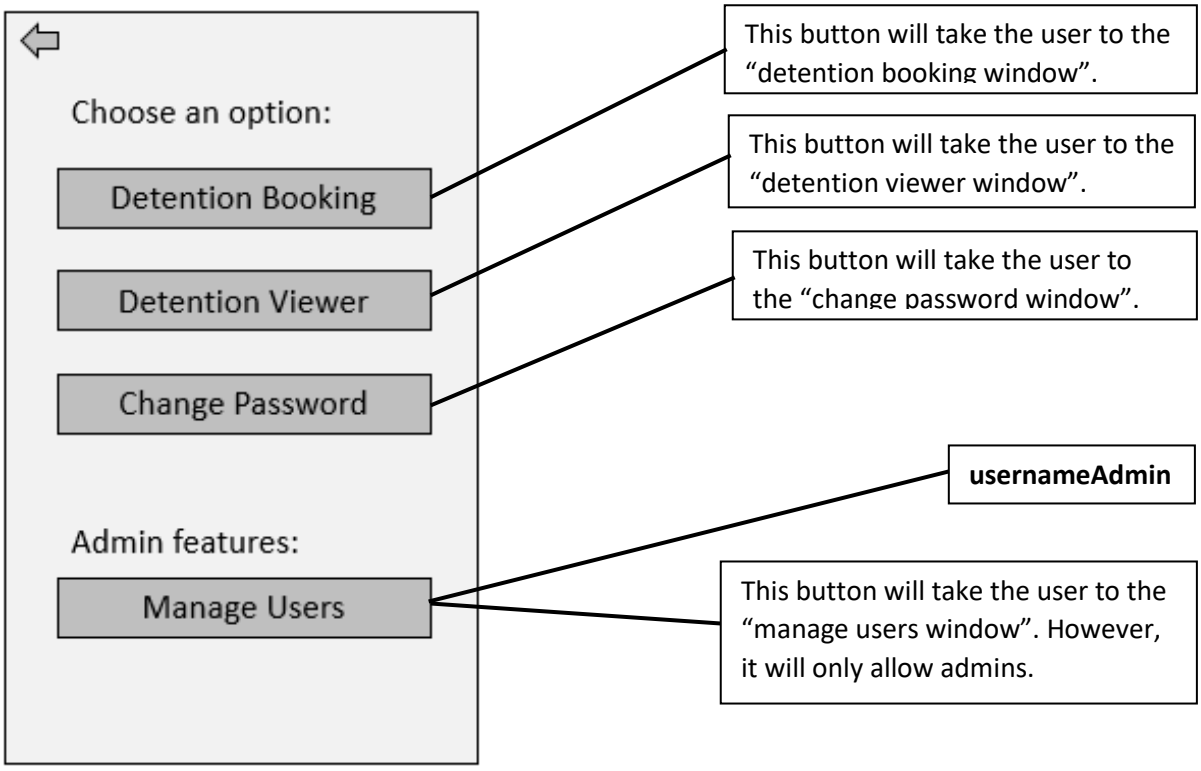
This code is similar in the way that it is also utilising the methods of traversing a 2D array, with the array being the same table “TeachersTable”. Therefore, it is also utilizing the same concept of a linear search algorithm. In line 11, the 4th row “`[i,5]`” in the array represents the “Password” field. This field stores the hashed version of the user’s password. Similarly, line 10 uses a field known as “Usernames”, which records unique usernames for each teacher. Line 6 uses an inbuilt function “`HASH()`”, which has the self-explanatory purpose of creating a unique hash from a given string. I will be using an already existing module for this, due to the nature of a hash function requiring a great enough deal of complexity that it would always create unique outputs, as well as being hard to decrypt. I would not be able to create this with my limited experience of programming.

The reasons for using global variables for username and password, as shown in lines 2 and 3, is due to how they are variables which are going to be constantly reused throughout the program. Therefore, it will save time by not having to constantly pass the variables. It will also increase readability of the program, as anyone reading the code will know what these variables are, and that they will remain constant throughout the program.

Although validation may be relevant here, the code for validation is not necessary at this point in the program. This is because I will ensure that passwords and usernames are valid on the point of creation. Therefore, all password and usernames will be of the expected form. By not ensuring validation of inputs in the login screen, I have effectively increased the complexity of everyone’s login credentials by increasing the number of potential passwords and usernames that user’s may have, since there is no limit as far as the login screen is concerned. For example, although passwords will only exist with a length between 7 and 14, (except for a reset or initial password which will be 123) anyone attempting to login with malicious intent will potentially be forced to factor in any length of password when attempting to break in.

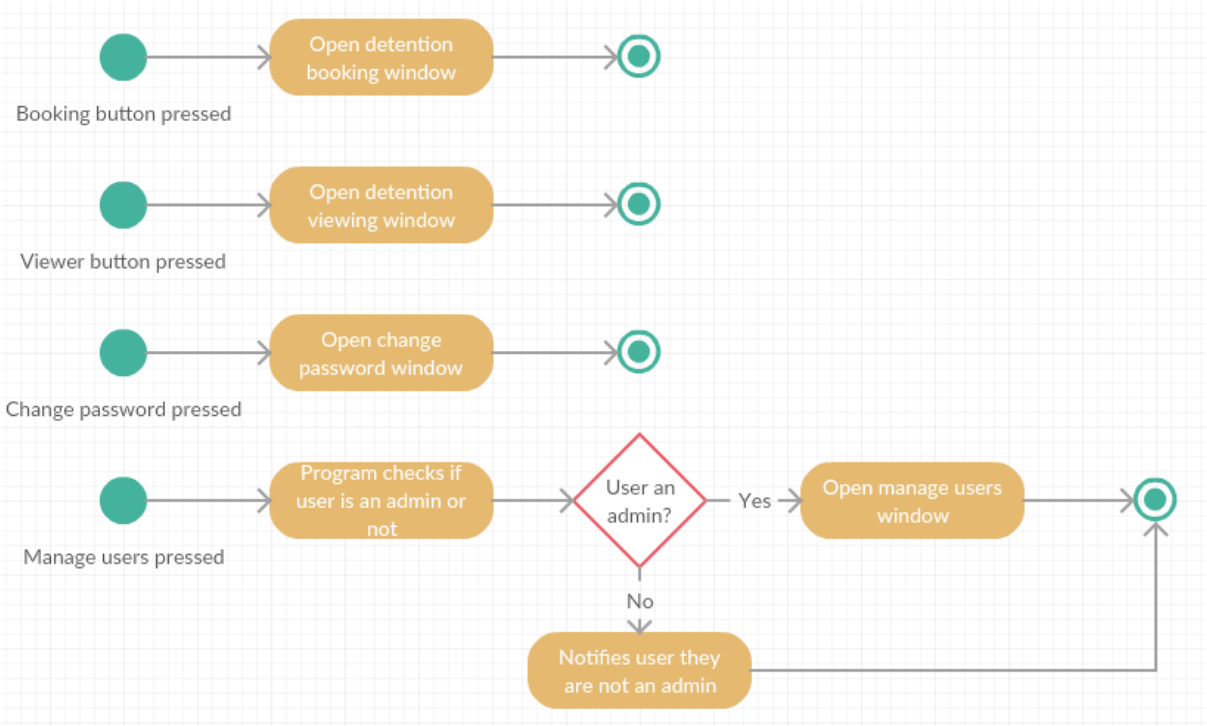
Menu window

The menu window will be the first window presented to the user after a successful login. It will be used to allow user's a simple way to navigate to any window that they want to at 1 click. The whole UI interface has been designed so that any window will take at most 2 clicks to navigate to from this menu. Manage users will take the user to an admin specific menu, if they are an admin.



Variable name	Data type	Purpose	Example
usernameAdmin	String	Stores the result of an SQL query which determines if the user is an admin or not.	Result of: "SELECT Administrator FROM Teachers WHERE TeacherUser = ?" Where ? is username e.g. YES

The following UML activity diagram shows the general functionality of the above menu window.



As shown above, this is a relatively simple activity diagram. This is due to the nature of the window's purpose, which is simply to provide a way of navigation to view each window. Thus, the primary function of each button is simply to open a window upon being clicked. As a result, I do not feel it necessary to write out the pseudocode for the first 3 paths shown in the diagram. The code is purely dependant on the software used to create the GUI's. As far as pseudocode would be concerned, the process would simply be "open next window, close current window".

However, a significant process does occur in the final path, as the program is required to check whether the current user is an administrator or not. If they are not, the system must not allow access to the user. I will split this final path into two separate algorithms. The first will be a general function that the program may make use of, which will take an input of "username", and determine whether the user is an administrative user or not. The second algorithm will then use this function to determine whether the user shall gain access to the "manage users" window or not.

Admin check

The following algorithm is similar to the authentication algorithm, in that it will utilise traversals of a 2D array, and reading from a text file. It will also use a linear search. It will be a function, which will return a value of “TRUE” or “FALSE”, for whether the user is an admin or not.

```
01  function CheckAdmin(username)
02        OPEN FILE “TeachersTable”
03        Teachers = FILE.READ(“TeachersTable”)
04        for i=0 TO LENGTH(Teachers)
05              if Teachers[i,3] = username THEN
06                    if Teachers[i,4] = “YES” THEN
07                          return “TRUE”
08                    else:
09                          return “FALSE”
10                    endif
11              else:
12                    pass
13              endif
14        next i
15  endfunction
```

Like the “help email” algorithm, this code searches through the teachers table, however, this algorithm searches by the username, and then returns the value of the “administrator?” field. When I create this code, I may choose to create “usernameAdmin” as a global variable, as opposed to simply returning “TRUE” or “FALSE”. The reason being because the system will not always require the user to be determined as a admin or not. It will only be necessary when the user attempts to do something that they must be an admin for. If the user has a long session doing many of these activities, then the global variable of “usernameAdmin” would simply need to be checked each time, as opposed to using the above function to determine whether they are an admin each time. The reason this is true is due to how the user can’t have their admin status changed during their own use of the system, simply because to create an admin, the user must already be an admin. If the user never uses an admin feature, then this function will not be required to be called at all, and therefore it will not matter whether the variable is global or local. However, if I were to use it as a local variable, then I would need to pass it numerous times through functions, as well as use inheritance to pass this attribute between classes. This would create more lines of code and thus make it harder to follow the code. Therefore, I believe it may be more efficient to use “usernameAdmin” as a global variable, in the same way as “username”.

Open "manage users" window

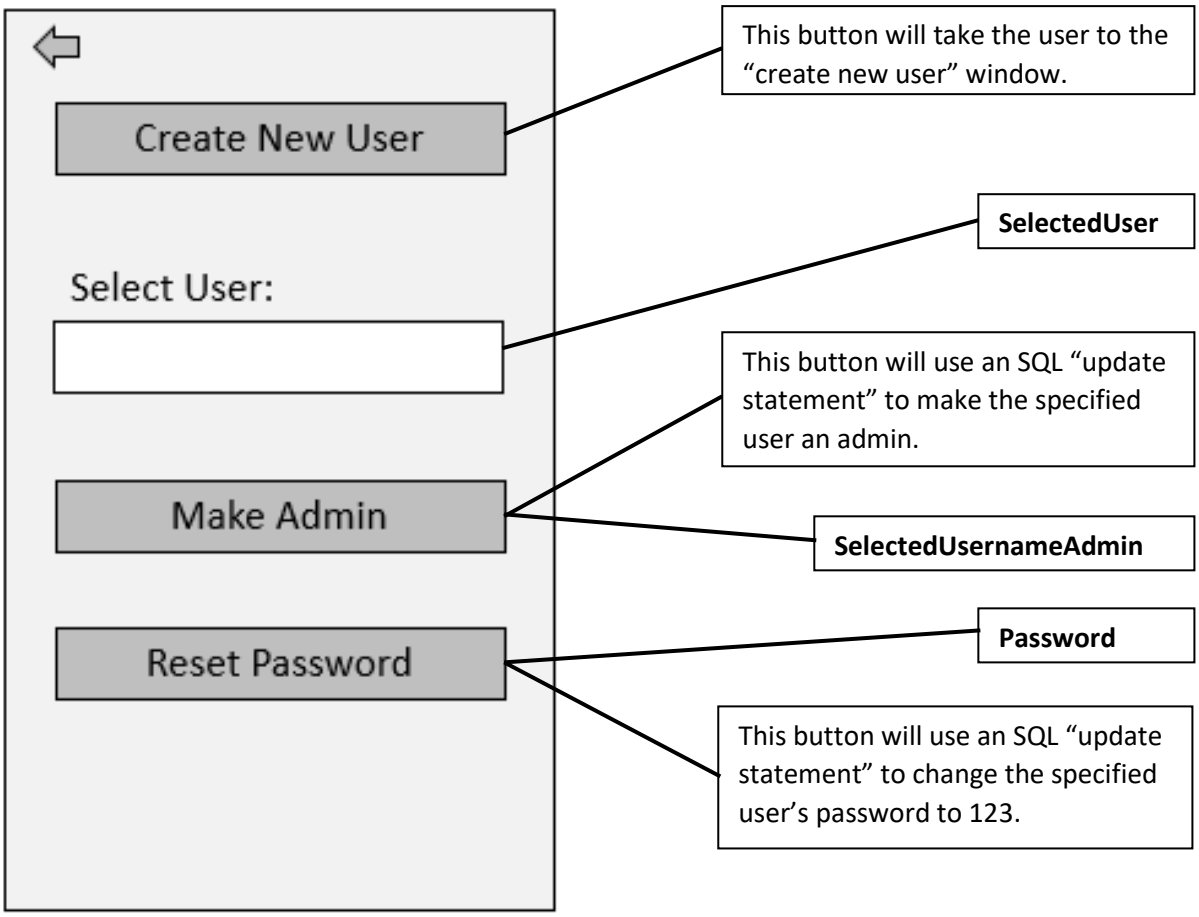
This algorithm is very like what the other 3 buttons would use on the "manage users" window, however this one will use the above function to determine whether the window will be opened or not.

```
01  procedure OpenManageUsersWindow(username) //called by button
02        if CheckAdmin(username) = "TRUE" THEN
03              print("Window would open here.")
04        else
05              print("Must be an admin to access this window.")
06        endif
07  endprocedure
```

As seen here, this algorithm is very short. For most buttons that link to another window, including back buttons, the only lines of code that would be used are lines 3 and 5, hence why I don't feel it necessary to demonstrate that using pseudocode.

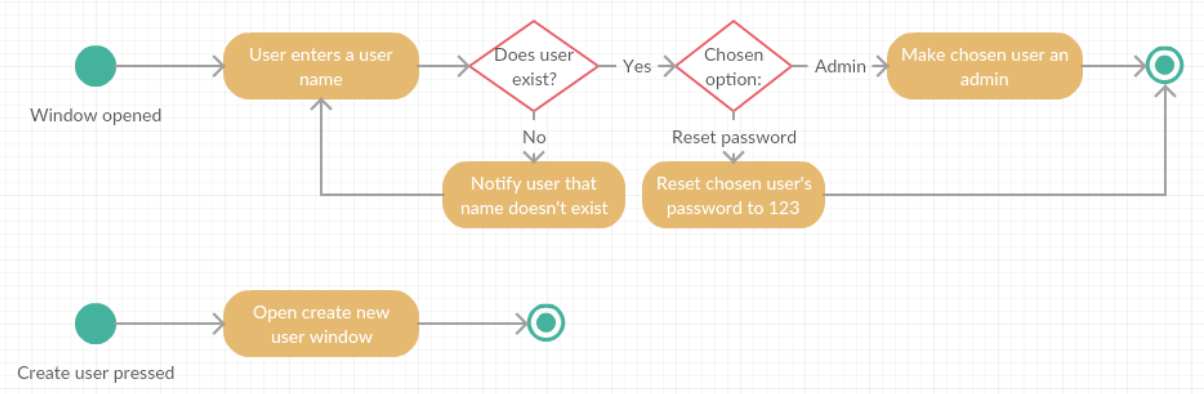
Manage users window

The next window I will be discussing is only accessible to administrative users, and will give these users the option to either visit another window (create new user) to create a completely new user, or to change the details of a specified current user.



Variable name	Data type	Purpose	Example
SelectedUser	String	Stores the user's input of the chosen user	ScottH6
Password	String	Stores the specified user's password so that it may be changed.	Password123
Selected-UsernameAdmin	String	Stores whether the specified user is an admin or not, so that it may be changed.	YES

The following UML activity diagram shows the general functionality of the above “manage users” window.



One of the paths shown are very simple, due to it only being a button that opens another window, much like in the menu window. However, the top path is far more complicated. I will split this into multiple different algorithms. The first of which will be to determine whether the input user exists. The second, will be to reset the chosen user’s password to 123. Finally, the third will be to determine whether the user is an admin or not, and if they aren’t, then it will make them an admin.

Check specified username

This initial function will be called when either of the “reset pass” or “make admin” buttons are pressed. Its only input will be taken from the “select user” line edit, and it will return either true or false for whether the user exists or not.

```
01  function CheckUsername(username)
02        OPEN FILE “TeachersTable”
03        Teachers = FILE.READ(“TeachersTable”)
04        for i=0 TO LENGTH(Teachers)
05              if Teachers[i,3] = username THEN
06                    return “VALID”
07              else
08                    pass
09              endif
10        return “INVALID”
11        next i
12  endfunction
```

Reset password

This algorithm will utilise the function above, calling it to determine whether the username is valid or not. Although no error would be produced if the username was invalid, validation here is necessary so that the user is aware they made an error. Therefore, they will be encouraged to retry so that they may achieve the desired result. This algorithm will be slightly different from all the previous ones, due to it being the first that will need to write to the database, as opposed to simply read from it.

```
01  procedure ResetPassword(username)
02        OPEN FILE "TeachersTable"
03        Teachers = FILE.READ("TeachersTable")
04        if CheckUsername(username) = "VALID" THEN
05              for i=0 TO LENGTH(Teachers)
06                    if Teachers[i,3] = username THEN
07                          Teachers[i,5] = HASH("123") //password
08                          TeachersNew = FILE.WRITE("TeachersTable")
09                          TeachersNew.write(Teachers)
10                    else
11                          pass
12                    endif
13              next i
14        else
15              print("username doesn't exist.")
16        endif
17  endprocedure
```

The 5th field (considering the 1st field is considered the 0 field) is the password within the "Teachers" table, hence why it is referenced in line 7 of the code. Lines 8 and 9 simply show the altered version of the "Teachers" array being written into the same file that was read from. As stated on OCR's pseudocode guide, when a file is written to, it can be assumed that the contents are being entirely replaced.

Make admin

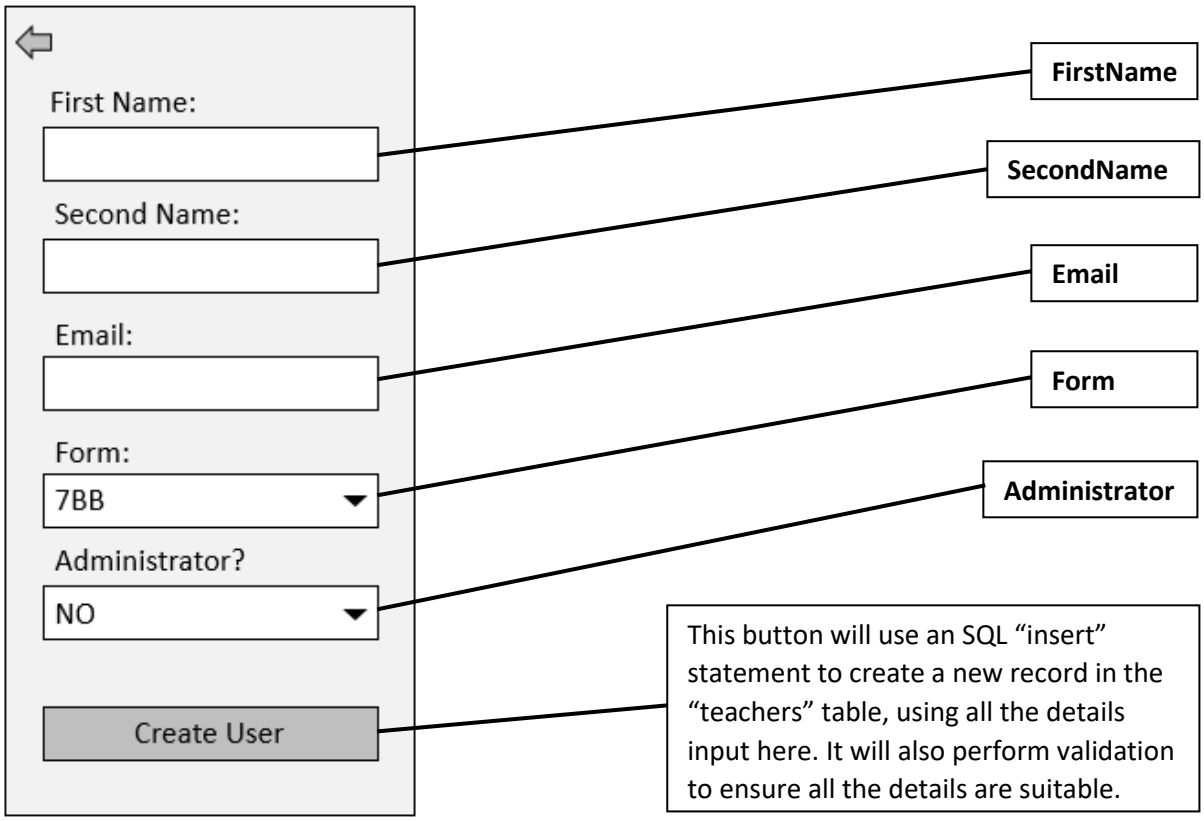
This algorithm will be like the "reset password" algorithm in that it will use traversals of a 2D array, reading from files, a linear search, the use of the "check specified username" function and writing to a file. However, it will have a slight increase of complexity due to this algorithm requiring a check to determine whether the user is an admin or not. As a result, it will utilise the "Admin check" function that I showed earlier. Again, this algorithm wouldn't produce an error if the specified user was already an admin, but I believe it to be beneficial for the user to know whether this is the case or not. This algorithm should identify that scenario.

```
01  procedure MakeAdmin(username)
02        OPEN FILE "TeachersTable"
03        Teachers = FILE.READ("TeachersTable")
04        if CheckUsername(username) = "VALID" THEN
05              if CheckAdmin(username) = "FALSE" THEN
06                    for i=0 TO LENGTH(Teachers)
07                          if Teachers[i,3] = username THEN
08                                Teachers[i,4] = "YES"
09                                TeachersNew = \
                                  FILE.WRITE("TeachersTable")
11                                TeachersNew.write(Teachers)
12                          else
13                                pass
14                          endif
15                    next i
16              else
17                    print("user is already an admin.")
18              endif
19        else
20              print("username doesn't exist.")
21        endif
22  endprocedure
```

Things to note for this algorithm include Teachers[i,4] represents the "administrator?" field, and the "\" in line 9 represents a line break. This is only necessary in this document for presentation.

Create new user window

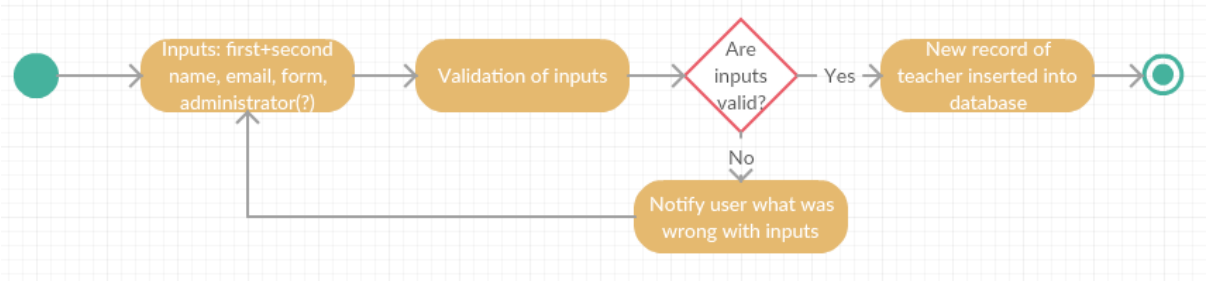
This window will be used to collect user inputs for creating a new user. It is only accessible from the "manage users" window, therefore being an admin is required to access it.



Variable name	Data type	Purpose	Example
FirstName	String	Stores user input of the first name	Lisa
SecondName	String	Stores user input of the second name	Smith
Email	String	Stores user input of the email address	brownb@mail.com
Form	String	Stores user input of the form	9CM
Administrator	String	Stores user input whether the user will be an administrator or not	YES
NextTeacherID	Integer	Increments previous TeacherID to make the next one	17
FirstLetter	String	Stores first letter of first name	L
TeacherUsername	String	Stores a TeacherUser, by combining the second name, first letter of the first name and the TeacherID	SmithL2

As shown in mock-up GUI made in Microsoft PowerPoint by using shapes, I have used combo boxes for the last two input boxes. One for forms, and one for whether they will be an administrator or not. The purpose of using these specific widgets is for validation. Instead of allowing users to input anything, and then determine whether it is valid or not, the combo box effectively does the validity check at the input stage. It simply only allows valid inputs, as the creator of the GUI chooses the inputs. The reason this is positive is because users will initially spend less time writing out their inputs, as they are already there. It means I, as the creator, can control all the inputs that I may expect to receive from that widget, which makes my job far easier for validation. For example, if forms were a line edit, I may have to factor in whether capital letters are used or not, which would either take more time, or I wouldn't do it and users would get frustrated when their input is rejected. Which brings me on to the final point of why combo boxes are positive, which is that they will never create errors. Errors usually cause frustration for the users, especially for my end-users which are not typically computer experts. Therefore, by reducing the errors as much as possible, it will be a far better experience for my end users.

The following UML activity diagram shows the general functionality of the above "create new user" window.



Although the activity diagram appears very simple, it has hidden complexity. This is due to how input validation will be required for the first step (taking the inputs), and then the new insertion record is relatively complex due to how many fields are present. As a result, I will be tackling this activity with multiple different sub-routines of pseudocode.

The first algorithm, will be a function that determines whether there are any integers present within a given string. This is important as both first and second names should not contain any integers.

I will then create an algorithm that is used to create a "TeacherUser", which is essentially the username. The reason this justifies a sub-routine by itself, is because it must be completely unique, and the best method of achieving this is by incrementing the previous TeacherID and adding in on the end of the TeacherUser. This method will ensure that every TeacherUser is unique, however the code will longer, hence why I am making it a single sub-routine.

The final sub-routine for this window will be the procedure that writes this new record for a teacher to the database.

As mentioned above, I will not need to use validation for the inputs of forms and administrators, as combo boxes are used, which provide only valid inputs. However, I will be using validation to ensure there are no non-alphabetic characters contained within the strings used for first and second names. I will not be using any form of validation for emails, besides ensuring the input isn't null, as even if I were to test emails are of the expected form, the only way to test if an email is valid is by using email verification which requires network capabilities. Therefore, validation of the email input would achieve nothing.

Alphabetic characters' validation

This algorithm is going to be used for validating inputs. It will be a function, due to how it will be used multiple times throughout the program, therefore making this code amenable to a modular approach. I will simply call the function with a string as its sole input, and it will return either "VALID" or "INVALID" depending on whether it is only comprised of alphabetic characters. For this algorithm I will assume I have access to a module known as "SPLIT_BY_x"() This function will create a list from a string, where each item that will be taken from the string is separated by the contents of the "x". For example, if I were to use SPLIT_BY_"."(string) where the "string" is a novel, it would create a list where each item is a sentence within the novel. It is used in line 2 of the following function, splitting a string by each letter, to create a list.

```
01  function AlphabeticValidation(string)
02        StringList = SPLIT_BY_LETTERS(string)
03        for i=0 to LENGTH(StringList)
04              if (ASCII_CODE(StringList[i])>64 AND \
                ASCII_CODE(StringList[i])<91) OR \
                (ASCII_CODE(StringList[i])>96 AND \
                ASCII_CODE(StringList[i])<123) THEN
05                    pass //character is alphabetic
06              else
07                    return "INVALID" //non-alphabetic character
08              endif
09        next i
10        return "VALID" //no non-alphabetic characters found
11  endfunction
```

As shown above, this algorithm assumes that I will be able to use a function "ASCII_CODE" which returns an ASCII value for a given character. It will then check this character to determine if it is in the range of 65-90 (i.e. it is a capital alphabetic character) or if it is in the range of 97-122 (i.e. it is a lower case alphabetic character). If it is not found to be in either of these ranges, then it must contain a non-alphabetic character.

Create TeacherUser

Create TeacherUser will be a function which takes various user inputs from the GUI, and then returns a unique TeacherUser, which is comprised of the first name, the first letter of the second name and the TeacherID of the teacher. No validation will occur within this function, as it will only be called when the next algorithm, "Create new user", is run. Create new user will use the "Alphabetic characters' validation" function to determine whether the inputs are valid, and if they are then this function will be called to create the TeacherUser. It will also return "NewTeacherID", as this is necessary for inserting the new record into the database.

```
01  function CreateTeacherUsername(FirstName, SecondName)
02        SecondNameList = SPLIT_BY_LETTERS(SecondName)
03        FirstLetter = SecondNameList[0] //first letter of surname
04        OPEN FILE "TeachersTable"
05        Teachers = FILE.READ("TeachersTable")
06        LatestRecordIndex = LENGTH(Teachers) - 1
07        if LatestRecordIndex > 0 THEN
08              HighestTeacherID = Teachers[LatestRecordIndex,0]
09              NewTeacherID = HighestTeacherID + 1
10        else
11              NewTeacherID = 1
12        endif
13        TeacherUser = FirstName + FirstLetter + str(NewTeacherID)
14        return TeacherUser, NewTeacherID
15  endfunction
```

Line 3 simply shows how the first letter is obtained from the second name by turning it into a list and then using the 0 index to find the first item. Lines 4 to 11 show how the algorithm will obtain the new TeacherID. This is essentially the latest TeacherID incremented. Therefore, I thought the most accurate way of obtaining this would be to use a read statement to my database to return the latest TeacherID within the database. The reason I chose this method, as opposed to simply using the length of the array and then incrementing it (number of records in the table + 1), is due to how if any records besides the latest record had been deleted, then that method would produce an incorrect TeacherID. It would create a TeacherID that is equal to or less than the latest TeacherID, which would end up not being unique. Considering TeacherID is the primary key of the "Teachers" table, it must be unique in order for my database to be fully normalised in 3NF. The method I have implemented is fool-proof however. The only way that it could have produced an invalid TeacherID is if the table had no records, however I factored this in by using an if statement in line 7 which will default the TeacherID to 1 if the new teacher is going to be the first record.

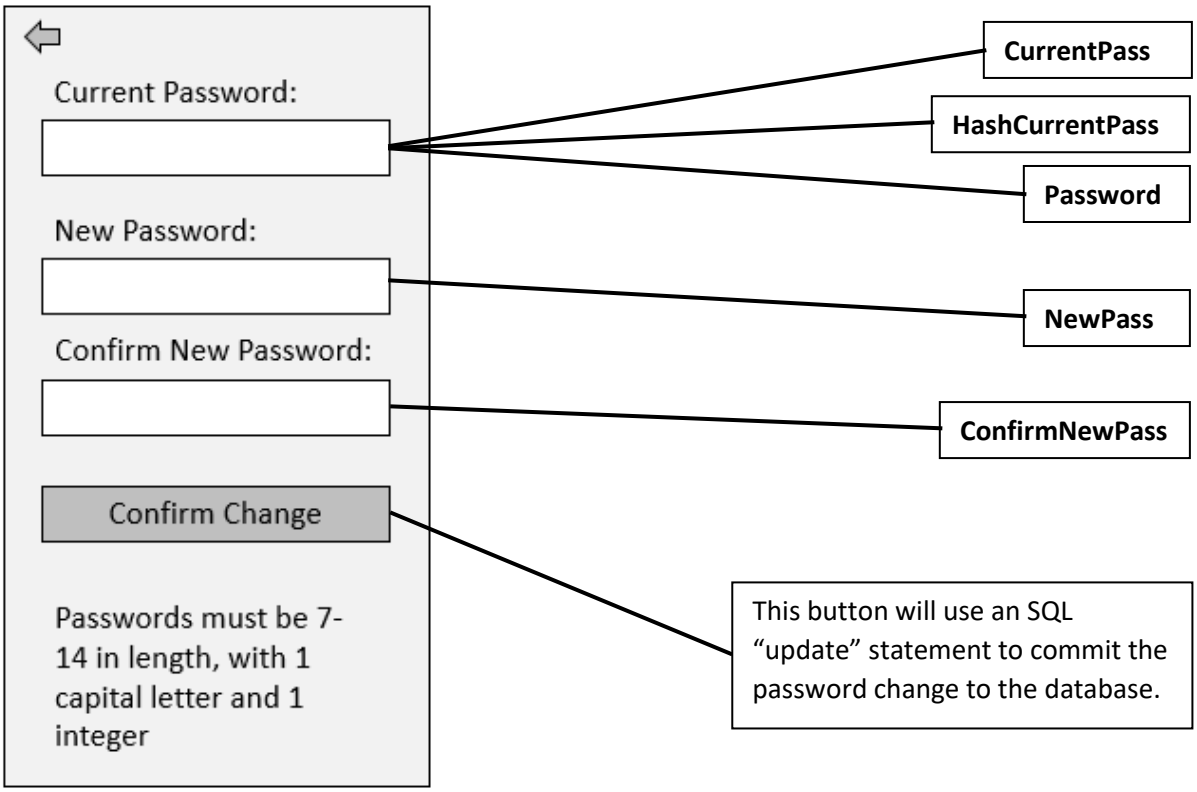
Create new user

I will be using an insert statement here, with the intent of creating a new record within the "Teachers" table, using the credentials input by the user. It will use the two above functions within itself, so that it may determine if the user inputs are valid, and for it to use the "TeacherUser" that will be created from these inputs. I chose to have them separate for the sake of readability, as modularity will make my program easier to follow. This procedure will also carry out validation itself, for it will check the "email" input. It will simply do a presence check to ensure that the email field is not left null. Although this leaves the field to be filled with a fake email, as discussed earlier, the presence check will simply be beneficial due to its purpose of a reminder to input the address.

```
01  procedure CreateNewUser(FirstName, SecondName, Email, FormID,
    Administrator)
02        if AlphabeticValidation(FirstName) = "VALID" AND \
          AlphabeticValidation(SecondName) = "VALID" THEN
03              TeacherUser = CreateTeacherUsername(FirstName,
                SecondName)[0] //returns TeacherUser
04              TeacherID = CreateTeacherUser(FirstName,
                SecondName)[1] //returns NewTeacherID
05              Password = HASH("123") //default password
06              NewRecord = [TeacherID, SecondName, FirstName,
                TeacherUser, Administrator, Password, Email, FormID]
07              OPEN FILE "TeachersTable"
08              Teachers = FILE.READ("TeachersTable")
09              APPEND_Teachers(NewRecord)
10              if Email != "" THEN
11                    TeachersNew = FILE.WRITE("TeachersTable")
12                    TeachersNew.write(Teachers)
13              else
14                    print("Email must not be left empty.")
15              endif
16        else:
17              print("Names contain invalid characters.")
18        endif
19  endprocedure
```

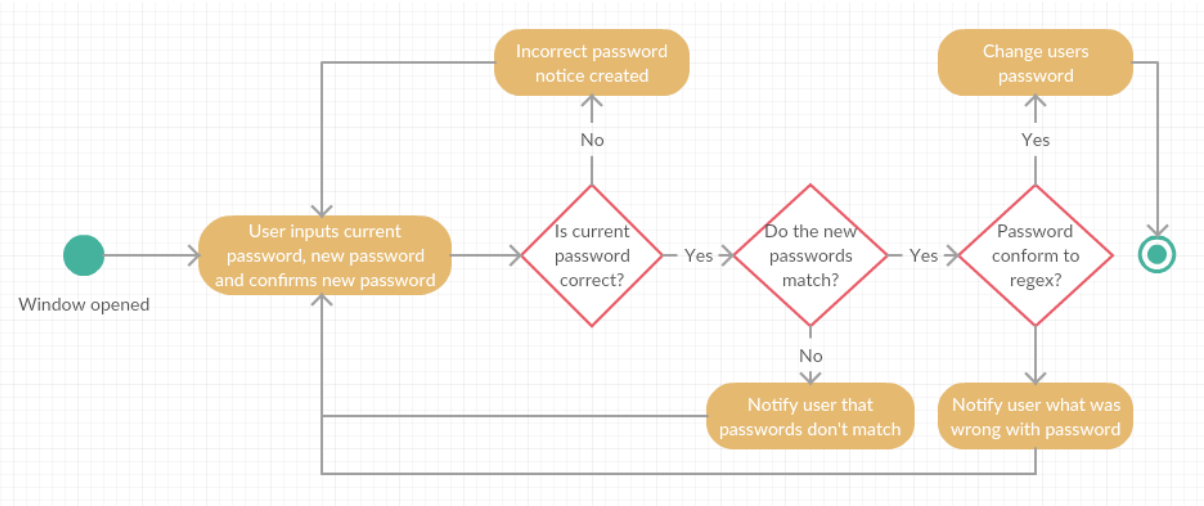
Change password window

Another one of the windows accessible from the menu window is the change password window, which allows the user to change their own passwords whenever they wish. This window will utilise relatively complex validation algorithms as a result.



Variable name	Data type	Purpose	Example
CurrentPass	String	Stores user input of their current password to verify the correct user it attempting to change the password.	Password123
HashCurrentPass	String	Stores hashed versions of CurrentPass	A8asf7y9f32hf928hf293s0192je012m19xm12x9aq23bkv1k2v0v2o012v4dm
Password	String	Stores hashed version of user's actual password as saved in the database, so it may be compared with CurrentPass.	A8asf7y9f32hf928hf293s0192je012m19xm12x9aq23bkv1k2v0v2o012v4dm
NewPass	String	Stores user input of new password	NewPassword12345
ConfirmNewPass	String	Stores user input to confirm new password, compares it with new password	NewPassword12345

The following UML activity diagram shows the general functionality of the above “change password” window.



As seen in the 3rd decision box, regular expressions will be utilised within this window for validation of the passwords. The reason I chose to use regular expressions as opposed to standard validation techniques, such as by using ASCII code like I did for name validation, is because this password validation is more complex. If I were to take a similar approach to this as I did with name validation, the code would become very long and hard to read. So, for the purpose of increasing readability, thus making my code easier to maintain, I have chosen to use regular expressions.

After researching regex using Wikipedia, I have found that regex is support is part of the standard libraries of most popular programming languages, with it being built into the syntax of many others. Therefore, I feel confident making the assumption that I will be able to utilise it. Python, the language I intend on using for developing my solution, is one of these languages that has a library to support regex.

For this activity diagram, I will create three different algorithms. The first will be a function that determines whether the user’s input credentials are valid, i.e. the current password is correct and the new passwords match.

The second will be a function that uses regular expressions to determine whether the password is valid or not.

The final algorithm will be a procedure which uses the previous two functions to determine if the password should be updated, and then an update statement will be used to write the change to the database.

Checking input passwords

As mentioned before, this algorithm will simply check the users input credentials to ensure that they are all correct, however no data validation will occur at this stage. It will be a function which returns a value depending on whether the current password was incorrect, the new passwords didn't match or everything was as expected.

```
01  function CheckInputPasswords(Password, CurrentPass, NewPass,
    ConfirmNewPass)
02        HashCurrentPass = HASH(CurrentPass) //hash user's input
03        if HashCurrentPass = Password THEN
04              if NewPass = ConfirmNewPass THEN
05                    return "VALID"
06              else
07                    print("New passwords don't match.")
08                    return "INVALID"
09              endif
10        else
11              print("Current password is incorrect.")
12              return "INVALID"
13        endif
14  endfunction
```

When this function is called in the main program it will be able to identify which credentials were incorrect, if any, depending on which string is returned (lines 5, 7 and 10). This is important as it can notify the user what exactly they need to change, as opposed to making them re-type everything despite their potentially being one mistake. This will make use of the system easier and less frustrating. Features such as this will allow the users to carry out tasks at a faster speed.

Regex of passwords

Regex will be of the form: r“expression”, and the function REGEX.FIND_ALL_MATCHING(expression, string) will return a list of all capture groups contained in the string for that expression.

```
01  function RegexPasswords(NewPass)
02        CharactersRegex = r"\W+"
03        LengthRegex = r"(\w{7,14})$"
04        DigitRegex = r"\d"
05        CapitalRegex = r"[A-Z]"
06        if LENGTH(REGEX.FIND_ALL_MATCHING(CharactersRegex,
          NewPass))>0 THEN
07              print("Password contains special characters.")
08              return "INVALID"
09        else
10              if LENGTH(REGEX.FIND_ALL_MATCHING(LengthRegex,
                NewPass)) < 1 THEN
11                    print("Password must be 7-14 in length.")
12                    return "INVALID"
13              else
14                    if LENGTH(REGEX.FIND_ALL_MATCHING(DigitRegex,
                      Newpass)) < 1
15                    THEN
16                          print("Password must contain a digit.")
17                          return "INVALID"
18                    else
19                          if LENGTH(REGEX.FIND_ALL_MATCHING
                            (CapitalRegex, NewPass)) < 1 THEN
20                                print("Must contain a capital.")
21                                return "INVALID"
22                          else
23                                return "VALID"
24                          endif
25                    endif
26              endif
27        endif
28  Endfunction
```

Lines 2-5 simply show variables being assigned to the regular expressions. Although I didn't need to use variables in this way, it simply increases readability. The "CharactersRegex" will match any string that contains a special character, as denoted by "\W". Hence why line 6 checks if there are more than 0 capture groups, because this would indicate use of a special character within a string. If this is the case, then the function may return invalid and notify the user why the password was invalid. All the other expressions are checked by ensuring that there are more than 0 capture groups returned, as they are searching for wanted traits of passwords. If they are no capture groups returned for the other 3 expressions, then it means they are missing certain things, therefore making the function return invalid. If all checks are successful, then the function may return valid.

The reason I require all these traits within passwords, is because it means that even if someone attempted to make a very easy, memorable password, then it would still be complex and extremely unlikely to be guessed.

Update password

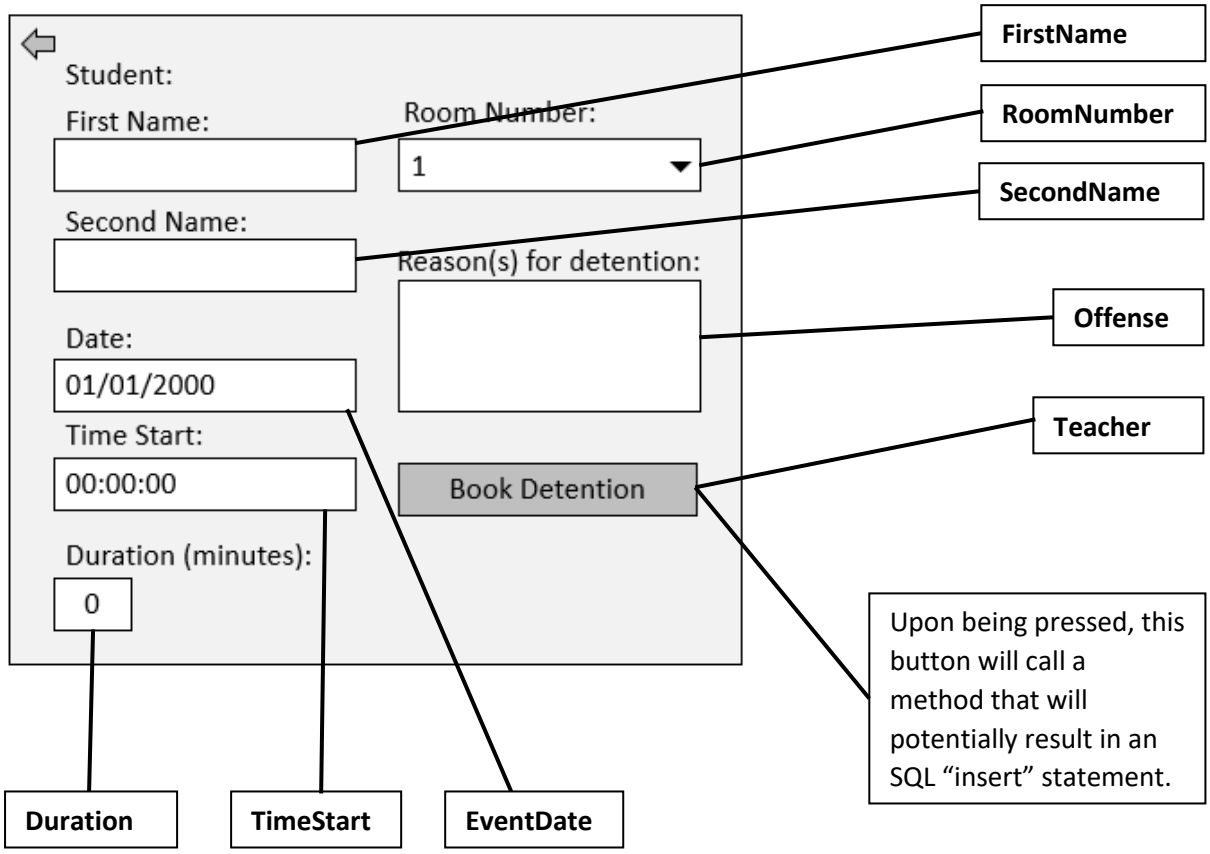
Now that the two validation algorithms have been assembled, it is possible for the third algorithm to update the password within the database, provided the user's inputs are valid.

```
01  procedure UpdatePassword(username, Password, CurrentPass,
    NewPass, ConfirmNewPass)
02        if (CheckInputPasswords(Password, CurrentPass, NewPass,
          ConfirmNewPass) = "VALID") AND (RegexPasswords(NewPass) =
          "VALID") THEN
03              OPEN FILE "TeachersTable"
04              Teachers = FILE.READ("TeachersTable")
05              for i=0 to LENGTH(Teachers)
06                    if Teachers[i,3] = username THEN
07                          Teachers[i,5] = HASH(NewPass)
08                    else
09                          pass
10                    endif
11              next i
12              TeachersNew = FILE.WRITE("TeachersTable")
13              TeachersNew.write(Teachers)
14        else
15              print("Please correct errors and try again.")
16        endif
17  endprocedure
```

Things to note for this algorithm are that the 3 index for each record represents the username, and the 5 index for each record represents the password. As mentioned previously, this procedure uses the two other functions from this window, as well as algorithms such as linear searchers, traversals of 2D arrays, reading to and writing from files, and finally regular expression algorithms from the other function.

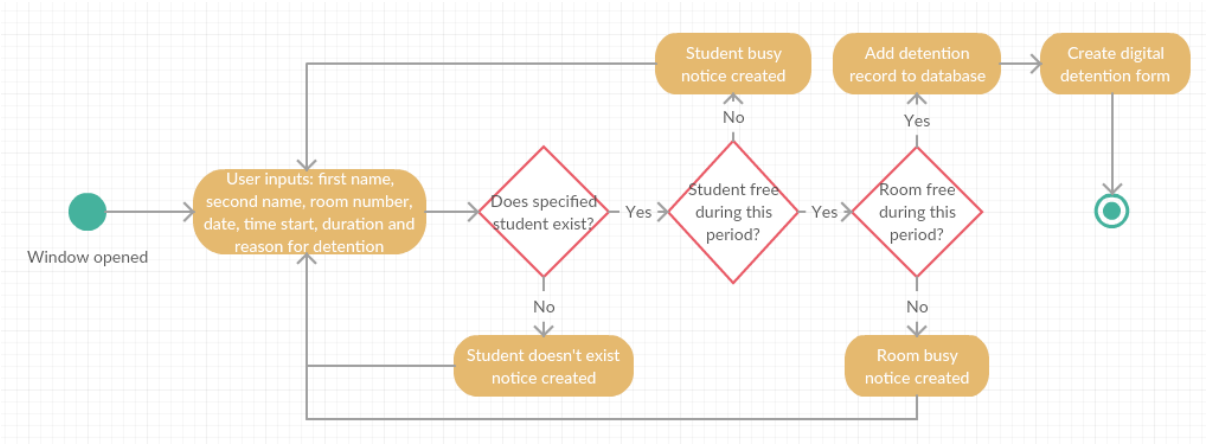
Detention booking window

This window will be used to book detentions. Much like the create new user window, it will therefore require lots of input widgets to receive credentials for the new records.



Variable name	Data type	Purpose	Example
EventDate	String	Stores the date of the detention	25/12/2016
TimeStart	String	Stores the start time of the detention	16:00:00
Teacher	String	Stores which teacher set the detention	Holly Scott
Student	String	Stores which student received the detention	Jason Vase
RoomNumber	Integer	Stores the room number of the location of the detention	6
Offense	String	Stores teacher's quote for why the student was issued a detention	Did not do 2 homework assignments in a row
Duration	Integer	Stores duration, in minutes, of detention	30
FirstName	String	Stores first name of student	Mark
SecondName	String	Stores second name of student	Jobs

The following UML diagram shows the general functionality of the above “detention booking window”.



As seen in the diagram, this is the most complex of the UML diagrams shown so far. This is due to how it requires a lot of input validation, and formatting of strings. The reason for this is because the strings are taken as inputs in the form of HH:MM:SS for the “TimeStart” variable. This is done so that the user finds it easier to input the time. However, a computer cannot simply process a string in this way, and so it must be converted to an integer. My method of doing this was converting each time to a number of minutes after midnight, though I will go into more detail of this when I create pseudocode for the activities.

Algorithms for this section may include validation that a student is free, validation that a room is free, conversions of strings to more suitable formats, insertion of a new detention record in a database, incrementing a student’s total number of detentions (not mentioned in the activity diagram), determining DetentionID of a new record and validation that student exists.

Convert time to minutes

This function will be used to convert a string in the form of HH:MM:SS (representing time) to a total of minutes after midnight. The purpose of this is to allow calculations using these times, as the computer cannot carry out calculations on a string of this format. Therefore, it must be converted to an integer. For the purpose of modularity and readability, I have decided that it is best to make this a function in itself, despite how simple it may be. It will use an inbuilt function known as "POP()", where the contents of the brackets is the data structure. This is a recognised term for removing the last item in a given data structure. This reason that this is used is to get rid of the seconds from the time. This is justified by how no teachers will be considering seconds when booking a detention.

```
01  function ConvertToMins(TimeString)
02        Time= SPLIT_BY_":"(TimeString) //creates list: [HH,MM,SS]
03        POP(Times) //removes seconds part of list
04        HourseConverted = int(Times[0])*60 //hours to mins
05        Mins = int(Times[1])
06        TimeStartMins = HoursConverted + Mins
07        return TimeStartMins
08  endfuntion
```

Find end time in minutes

This function is like the one above, in that is should be relatively simple, however for the purpose of modularity, it will exist in its own function. It will be combined with the above function, to find "TimeStartMins", and duration to work out "TimeEndMins".

```
01  function EndInMins(TimeStartMins, Duration)
02        Duration = int(Duration) //ensures data type
03        TimeEndMins = TimeStartMins + Duration
04        return TimeEndMins
05  endfunction
```

Convert back to time string

This function is the reverse process of “convert time to mins”. It takes in a time which is minutes relative to midnight, and puts it in the form of HH:MM:SS as a string. This is done for presentation purposes, as it is far easier for the user to understand a time in this format as opposed to the minute’s alternative.

```
01  function ConvertToTimeString(TimeInMins)
02        Hours = TimeInMins DIV 60 //returns whole number of hours
03        Mins = TimeInMins MOD 60 //returns remaining minutes
04        if Mins < 10:
05              Mins = "0" + str(Mins)
06        else
07              Mins = str(Mins)
08        endif
09        if Hours < 10:
10              Hours = "0" +str(Mins)
11        else
12              Hours = str(Hours)
13        endif
14        TimeInMins = Hours + ":" + Mins + ":00"
15        return TimeInMins
16  endfunction
```

The code is relatively self-explanatory in this function. However, it is worth noting why lines 4/5 and 9/10 are necessary. If they were not there, then a time such as 09:05:00 would be displayed as 9:5:00, which is not valid. The if statements present simply add the “0” before the integers if necessary.

Another thing to note is on line 14, a constant of “:00” is always added at the end. This is done because no teacher is going to be considering seconds when booking a detention. However, instead of completely ignoring them, I have factored them in and just worked around them by removing them. The reason being because it cannot possibly be detrimental to have this approach, however removing the seconds could result in unforeseen downsides.

Check student

This function will use the input first and second name to find the "StudentID". This is necessary as the "Detentions" table will store record the student by using this, due to it being the primary key of the "Students" table and a foreign key in the "Detentions" table. If the student doesn't exist according to the first and second name input by the user, then it will notify the user of this, requesting they retry their inputs.

```
01  function CheckStudent(FirstName, SecondName)
02        OPEN FILE "StudentsTable"
03        Students = FILE.READ("StudentsTable")
04        for i=0 to LENGTH(Students)
05              if (Students[i,1] = SecondName) AND
                (Students[i,2] = FirstName) THEN
06                    StudentID = Student[i,0]
07                    return StudentID
08              else
09                    pass
10              endif
11        next i
12        return "INVALID"
13  endfuncion
```

This function is a form of input validation, by ensuring that the input student exists. Things to note about this algorithm include how the indexes 0, 1 and 2 represent "StudentID", "SecondName" and "FirstName" respectively.

One assumption this algorithm has to make for it to work correctly is that there are no students who share the same first and second names. Although this is probably not the case in reality, it is fine as I will include within any installation instructions that upon setting up the student database, if there are students who share the same names, then they should add an integer at the end of their second name. For example, there would be Mark Jobs, Mark Jobs1, Mark Jobs3 etc. The teacher can then use the database to identify which student it is. Although this may seem inconvenient, is the most suitable solution that I have been able to plan on implementing.

Incrementing student's detentions

The purpose of this algorithm is to update the "Students" table by incrementing the total number of detentions for a student when they are booked into a detention. Since this system will carry out statistical analysis that uses the number of detentions for a student, it is vital that number of detentions is kept up to date. One of my objectives is that the system will carry this out automatically. I will make this a procedure, which is run once the detention has been booked, ensuring that the record is valid. This means that the number of detentions for each student should be accurate. It will use the "StudentID" provided by the "check student" algorithm.

```
01  procedure IncrementDetentions(StudentID)
02        OPEN FILE "StudentsTable"
03        Students = FILE.READ("StudentsTable")
04        for i=0 to LENGTH(Students)
05              if Students[i,0] = StudentID THEN
06                    DetentionCount = Students[i,4]
07                    DetentionCount = DetentionCount + 1
08                    Students[i,4] = DetentionCount
09                    StudentsNew = FILE.WRITE("StudentsTable")
10                    StudentsNew.write(Students)
11              else
12                    pass
13              endif
14  endprocedure
```

Things to note from this procedure include how the index 4 for the Students table represents the "DetentionCount" field. Lines 6-10 simply show this value being retrieved from the table for that specified student, the same value being incremented and wrote back into the record, and the changes committed to the file.

Find RoomID and block

This algorithm will be relatively simple; however, it is completely necessary due to the nature of my database. “Detentions” will store the room location in the field “RoomID”. As the teachers will not know what this field is, they must choose the room by something familiar, in this case, the “RoomNumber”. I will have to use the RoomNumber to find the RoomID, which is the primary key of the “Rooms” table. I will make it into a function which simply returns the RoomID. Due to the input for RoomNumber being a combo box, I will not need to use any validation here, as I know any input will be able to return a value from the database. Another feature of this function is it will return the block of the RoomNumber. This could have been done in a separate function, however, since the table is already being traversed, it is far more efficient to do both at the same time. Therefore, the function will return a list of the form [RoomID, Block]

```
01  function FindRoom(RoomNumber)
02        OPEN FILE “RoomsTable”
03        Rooms = FILE.READ(“RoomsTable”)
04        for i=0 to LENGTH(Rooms)
05              if Rooms[i,2] = RoomNumber THEN
06                    return Rooms[i,0], Rooms[i,1]
07              else
08                    pass
09              endif
10        next i
11  endfunction
```

The only thing to note about this function is that the indexes 0, 1 and 2 of the Rooms table represent the fields RoomID, Block and RoomNumber respectively.

Check if room is free

This algorithm would be very long if treated by itself, however, with use of the 3 functions that I made which work with time strings, it's simpler. It will also require the use of the "FindRoomID" function, as the "Detentions" table uses this as a foreign key to identify which room is in use. For the purpose of the algorithms that use the Detentions table, I will be assuming the field are as follows: DetentionID, EventDate, TimeStart, TimeEnd, Duration, RoomID, TeacherID, StudentID, Offense.

```
01  function CheckRoomAvailability(RoomNumber, EventDate,
    TimeStart, Duration)
02        RoomID = FindRoom(RoomNumber)[0]
03        TimeStartMins = ConvertToMins(TimeStart)
04        TimeEndMins = EndInMins(TimeStartMins, Duration)
05        OPEN FILE "DetentionsTable"
06        Detentions = FILE.READ("DetentionsTable")
07        for i=0 to LENGTH(Detentions)
08              if Detentions[i,5] = RoomID AND
                (Detentions[i,1] = EventDate) AND
                ((Detentions[i,2] > TimeStartMins) AND
                (Detentions[i,2] < TimeEndMins) OR
                (Detentions[i,3] > TimeStartMins) AND
                (Detention[i,3] < TimeEndMins)) THEN
09                    print("Room is not free.")
10                    return "BUSY"
11              else
12                    return "FREE"
13              endif
14        next i
15  endfunction
```

As seen this algorithm is quite complex. Essentially, the if statement at line 8 will check if there any existing detention records on the same date, in the same room, which overlap each other. It does this by checking if the new detention will commence, or end during the period of any other detention. This should be able to deny any overlapping detentions. The indexes of the detentions table are shown in the text above this algorithm.

Check student availability

This algorithm will be very similar to the function that checks if the room is available. The reason I have chosen to check students and rooms separately, is so that the user may notified what the specific issue is. For example, if it didn't identify that the student was already busy, a teacher might to book a busy student in a different room repeatedly, but to no avail. This would potentially frustrate the user, and waste their time.

```
01  function CheckStudentAvailability(StudentID, EventDate,
    TimeStart, Duration)
02        TimeStartMins = ConvertToMins(TimeStart)
03        TimeEndMins = EndInMins(TimeStartMins, Duration)
04        OPEN FILE "DetentionsTable"
05        Detentions = FILE.READ("DetentionsTable")
06        for i=0 to LENGTH(Detentions)
07              if Detentions[i,7] = StudentID AND
                (Detentions[i,1] = EventDate) AND
                ((Detentions[i,2] > TimeStartMins) AND
                (Detentions[i,2] < TimeEndMins) OR
                (Detentions[i,3] > TimeStartMins) AND
                (Detention[i,3] < TimeEndMins)) THEN
08                    print("Student is not free.")
09                    return "BUSY"
10              else
11                    return "FREE"
12              endif
13        next i
14  endfunction
```

The main difference in this algorithm to the function that checks the room availability is how the 7 index of the "Detentions" table is used as opposed to the 5 index. The reason being because these indexes represent "RoomID" and "StudentID".

Find new DetentionID

This function is similar to the one used to create a new, unique TeacherUser. TeacherUser's use the TeacherID, which is a unique, incrementable primary key. DetentionID is the equivalent of TeacherID, and so it uses a similar method to be located, as shown below.

```
01  function FindDetentionID()
02        OPEN FILE "DetentionsTable"
03        Detentions = FILE.READ("DetentionsTable")
04        LatestRecordIndex = LENGTH(Detentions) - 1
05        if LatestRecordIndex > 0 THEN
06              HighestDetentionID = Detentions[LatestRecordIndex,0]
07              NewDetentionID = HighestDetentionID + 1
08        else
09              NewDetentionID = 1
10        endif
11        return NewDetentionID
12  endfunction
```

Create detention record

Procedure will verify new detention record, and then insert it into the database.

```
01  procedure CreateDetention(DetentionID, EventDate, TimeStart,
    Duration, RoomNumber, FirstName, SecondName, Offense)
02        RoomID = FindRoom(RoomNumber)[0]
03        StudentID = CheckStudent(FirstName, SecondName)
04        if StudentID = "INVALID" THEN
05              print("Student does not exist with that name.")
06        else
07              if (CheckRoomAvailability(RoomNumber, EventDate,
                TimeStart, Duration) OR
                CheckStudentAvailability(StudentID, EventDate,
                TimeStart, Duration)) = "BUSY" THEN
08                    print("Invalid detention record.")
09              else
10                    DetentionID = FindDetentionID()
11                    TimeStartMins = ConverToMins(TimeStart)
12                    TimeEndMins = TimeEndInMins(TimeStartMins,
13                    Duration)
14                    NewRecord = [DetentionID, EventDate,
15                    TimeStartMins, TimeEndMins, Duration, RoomID,
                      TeacherID, StudentID, Offense]
16                    OPEN FILE "TeachersTable"
                      Detentions = FILE.READ("DetentionsTable")
                      APPEND_Detentions(NewRecord)
17                    DetentionsNew = FILE.WRITE("DetentionsTable")
18                    DetentionsNew.write(Detentions)
19                    IncrementDetentions(StudentID)
20              endif
21        endif
22  endprocedure //note that TeacherID is a global variable
```

Calculation algorithms

Here I will create various algorithms which are to be used for calculations and statistical analysis of the data recorded by the system.

Find total detentions of students

This algorithm will be used as a general algorithm to obtain the total number of detentions for a given student. It will use the "CheckStudent" function from the detention booking window segment. This function will return the StudentID of a student, given their first and second name, and identify if they don't exist at all.

```
01  function StudentTotalDetentions(FirstName, SecondName)
02        StudentID = CheckStudent(FirstName, SecondName)
03        OPEN FILE "StudentsTable"
04        Students = FILE.READ("StudentsTable")
05        for i=0 to LENGTH(Students)
06              if Students[i,0] = StudentID THEN
07                    DetentionCount = Students[i,4]
08                    return DetentionCount
09              else
10                    pass
11              endif
12        next i
13        print("Student doesn't exist")
14        return "INVALID"
15  endfunction
```

The only thing to note from this algorithm, is that the 4 index of the Students table is the "DetentionCount" field, which records the total number of detentions for a given student.

Find total detentions

This algorithm will have a similar function to the one above, except it finds the total detentions for everyone together. Useful by itself, or for working out percentages of detentions.

```
01  function TotalDetentions()
02        TotalDetentionCount = 0
03        OPEN FILE "StudentsTable"
04        Students = FILE.READ("StudentsTable")
05        for i=0 to LENGTH(Students)
06              Temp = Students[i,4] //4th index = DetentionCount
07              TotalDetentionCount = TotalDetentionCount + Temp
08        next i
09        return TotalDetentionCount
10  endfunction
```

Total detentions of form

This algorithm will be used to identify the amount of detentions for a given form, similar to the previous function which found detentions for a given student.

```
01  function FormTotalDetentions(FormID)
02        TotalFormDetentions = 0
03        OPEN FILE "StudentsTable"
04        Students = FILE.READ("StudentsTable")
05        for i=0 to LENGTH(Students)
06              if Students[i,5] = FormID
07                    TotalFormDetentions = (TotalFormDetentions +
                      Students[i,5])
08              else
09                    pass
10              endif
11        next i
12        return TotalFormDetentions
13  endfunction
```

Percentages of detentions

This algorithm will be used, in combination with the previous algorithms, to find the percentage of total detentions which this source is accountable for. The source may be a form, a student, or potentially a year group. There will be 2 algorithms, as one will work when the source is a student, the other for when the source is a form group.

```
01  function DetentionPercentageForms(FormID)
02        TotalFormDetentions = FormTotalDetentions(FormID)
03        TotalDetentionCount = TotalDetentions()
04        Percentage =(TotalFormDetentions/TotalDetentionCount)*100
05        return str(Percentage)+"%"
06  endfunction
```

```
01  function DetentionPercentageStudent(StudentID)
02        StudentDetentions = \
          StudentTotalDetentions(FirstName,SecondName)
03        TotalDetentionCount = TotalDetentions()
04        Percentage =(StudentDetentions/TotalDetentionCount)*100
05        return str(Percentage)+"%"
06  endfunction
```

Student with most detentions

The purpose of this algorithm is to iterate through the database, and return the StudentID with the highest amount of detentions.

```
01  function MostDetentionsStudent()
02        OPEN FILE "StudentsTable"
03        Students = FILE.READ("StudentsTable")
04        HighestDetentions = 0
05        for i=0 to LENGTH(Students)
06              if Students[i,4] > HighestDetentions THEN
07                    Students[i,0] = HighestDetentionsStudent
08                    Students[i,4] = HighestDetentions
09              else
10                    pass
11              endif
12        next i
13        return HighestDetentionsStudent, HighestDetentions
14  endfunction
```

The method of doing this uses a principle that Dijkstra's algorithm is based on, by checking every "route" (in this case every student), but instead of assuming that all the unchecked routes are infinity, it assumes they are all 0 until checked.

Student with least detentions

Similar to the above algorithm, this will check all the student's total detentions, but it will instead return the student with the least detentions, and the most.

```
01  function LeastDetentionsStudent()
02        OPEN FILE "StudentsTable"
03        Students = FILE.READ("StudentsTable")
04        LowestDetentions = 999 //serves as infinity
05        for i=0 to LENGTH(Students)
06              if Students[i,4] < LowestDetentions THEN
07                    Students[i,0] = LowestDetentionsStudent
08                    Students[i,4] = LowestDetentions
09              else
10                    pass
11              endif
12        next i
13        return LowestDetentionsStudent, LowestDetentions
14  endfunction
```

Range of detentions

This will show the range in the amount of detentions for a student

```
01  function FindDetentionRange()
02        HighestDetentions = MostDetentionsStudent()[1]
03        LowestDetentions = LeastDetentionsStudent()[1]
04        DetentionRange = HighestDetentions - LowestDetentions
05        return DetentionRange
06  endfunction
```

Average detentions of students (mean)

This algorithm will be another that is used for statistical analysis. The mean is a relevant average as it shows the general behaviour of students as a whole, as opposed to looking at individual cases.

```
01  function StudentDetentionsMean()
02        TotalDetentionCount = TotalDetentions()
03        OPEN FILE "StudentsTable"
04        Students = FILE.READ("StudentsTable")
05        NumStudents = LENGTH(Students)
06        StudentDetentionsMean = TotalDetentionCount/NumStudents
07        return StudentDetentionsMean
08  endfunction
```

Overview of algorithms

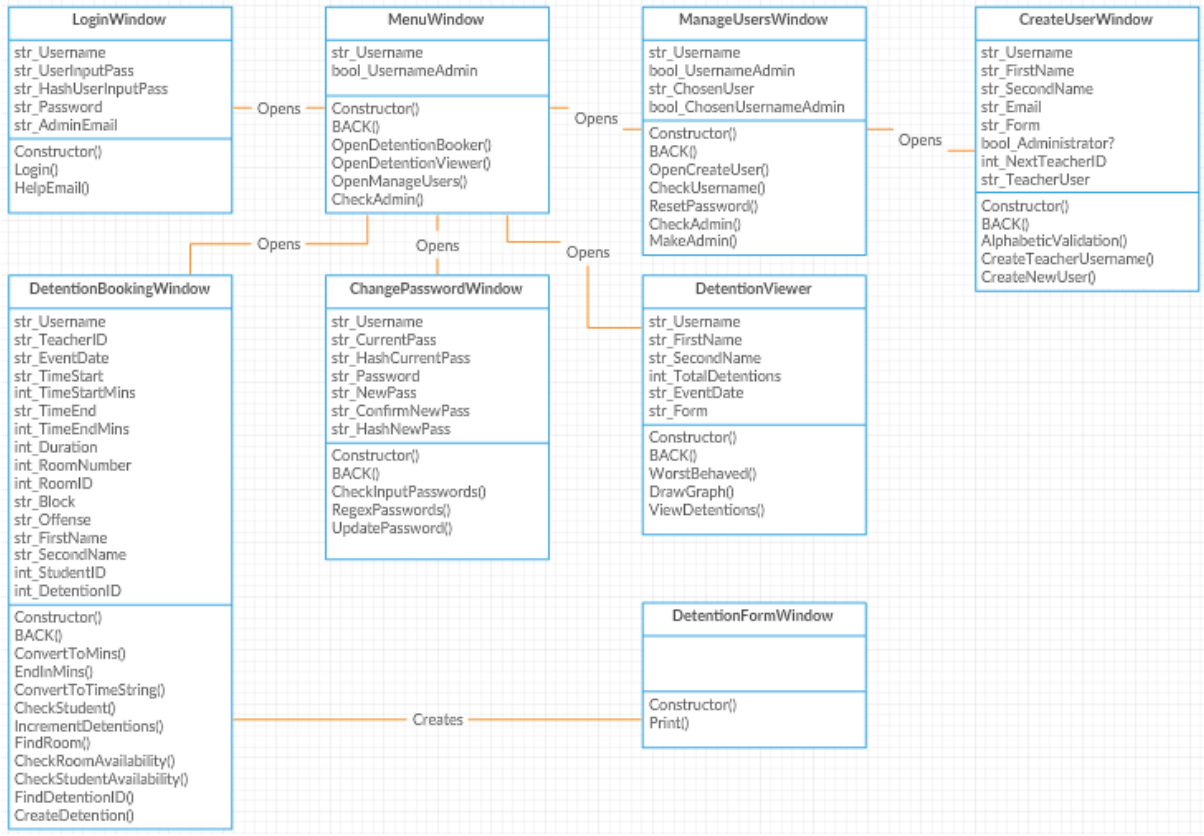
As shown in all the above pseudocode, many different algorithms have been used. For the sake of readability, I will compile a list of all the used algorithms here.

- Searching through a 2D array
- Linear search
- Reading from text files
- Writing to text files
- Replacing items within arrays
- Removing items from arrays
- Adding records to arrays
- Regular expressions
- Presence checks
- Ascii code checks
- Hash function
- Nested loops
- Mean of a given data set
- Length of a given data set
- Largest value in a given data set
- Lowest value in a given data set
- (above two using principles from Dijkstra's algorithm of shortest path)
- Mean value of a given data set
- % of contribution to a given data set
- Object creation, and inheritance
- Graph drawing
- Replacing items within strings
- Converting times to integer based formats which can have calculations carried out upon them
- Checking if time periods overlap one-another, by use of dates too

Object orientated programming

Considering I am using a GUI system based on individual windows, I feel it appropriate to use an object orientated approach. I will create a UML class diagram to demonstrate how my classes may look, though this will only be an abstract model of how it may turn out.

I will also create an algorithm using pseudocode which will demonstrate how part of my program may utilise inheritance. I will not require inheritance anywhere in my program necessarily, since I have a database. This means that any variables which I need to view over extended windows may either be global variables, or variables that I can retrieve from the database. However, for creating a detention form output, I believe it will be easier to simply inherit all the variables from the "detention booking window". This will save processing time as records will not need to be written to a disk, instead temporary variables which have already been defined will be utilised. They will also be of the form expected, since nothing has been done to them since they were processed.



The above diagram shows classes which represent individual windows as part of the GUI. The connections labelled "opens" show how you can navigate from one window to another. The only window which uses any kind of inheritance is "DetentionFormWindow", which is the child class of "DetentionBookingWindow". The reason being due to how all attributes used to create a detention form have been declared for the sake of booking a detention. As a result, a very easy solution for creating the form is to inherit all its attributes from the booking window.

Inheritance to create a detention form

As stated above, the "DetentionFormWindow" will simply be used to produce an output which is purely dependant on the previous "DetentionBookingWindow". Therefore, it will have none of its own attributes, and the only method it contains itself will be "print()". The purpose of print is simply to create an XMS document of the output, and so there is no need to create pseudocode for it. However, I will show, using pseudocode, how these two classes will interact.

```
01  class DetentionFormWindow inherits DetentionBookingWindow
02        public procedure Constructor(self, parent)
03        endprocedure
04        public procedure CreateForm(self, parent)
05              print("RGS DETENTION FORM: PLEASE SIGN AND RETURN TO
                ISSUER OF DETENTION ON SPECIFIED DATE.")
06              print("To parent of:"+super.StudentName)
07              print("Form:"+super.FormName)
08              print("Date:"+super.EventDate)
09              print("Time start:"+super.TimeStart)
10              print("Time end:"+super.TimeEnd)
11              print("Location of
                detention:"+super.Block+super.RoomNumber)
12              print("Detention issued by:"+super.TeacherName)
13              print("Reason for detention:"+super.Offense)
14        endprocedure
15        public procedure Print(self, parent)
16              (CREATE XML DOCUMENT OF "CreateForm" OUTPUT)
17        endprocedure
18  endclass
```

The procedure "CreateForm" which extends through lines 4 and 14 suggests how the output may be displayed. All the attributes are received by inheritance of the "DetentionBookingWindow" class, and therefore no attributes are required to be declared. The constructor method also serves no purpose for the same reason. Since this is simply pseudocode I have shown the output as simply printing the various details of the detention, though this will most likely be far different in the final product, possibly by use of HTML to create a high-quality form. The detention booking class may call the "CreateForm" procedure like so: `DetentionFormWindow.CreateForm(self)`

Test plan

For my solution to be a success, I must be able to achieve my objectives to an extent that makes my stated stakeholders satisfied. To do this, I must provide evidence that I have achieved these objectives. This includes what I tested, how I tested it, what the outcome was (was it a complete success or partial success?), and when the test can be carried out. Some of these tests will naturally only be able to be achieved once the development process has been completed, however some will be able to occur during the development process.

Existing test data

For many of my tests, I will require one of my stake holders to test the functionality of my program using mock-up accounts. One will have to be an admin, and one will be a standard user. The records of these "test accounts" shall be seen here.

TeacherID	Surname	First-Name	TeacherUser	Admini-strator?	Un-hashed password	Email	FormID
1	Brown	Bob	BrownB1	YES	BrownB1	Brownb-@mail.com	1
3	Allen	Tim	AllenT3	NO	AllenT3	Allen-@mail.com	3

The reason using multiple accounts is necessary is due to how various success criteria depend on users having different access levels in the program. Because of this, I must be able to prove that different accounts will test differently.

Some test data present will be based upon the RGS's data, for example, the rooms. These have little impact on how the program will function, provided the rooms have a RoomID. The "block" and "RoomNumber" are just ways for users to recognise what the rooms are. In my database, for testing, the rooms will be as follows:

RoomID	Block	RoomNumber
1	Junior	1
2	Junior	2
3	Junior	3
4	Junior	4
5	Junior	5
6	Junior	6
7	Main	22
8	Main	23
9	Main	24
10	Main	25
11	Main	26
12	Main	27
13	Maths	31
14	Maths	32

15	Maths	33
16	Maths	34
17	Maths	35
18	Maths	36

More necessary test data include population of the “forms” table.

FormID	FormName	TeacherID
1	7BB	1
2	7SL	2
3	8AT	3
4	8HS	4
5	9CM	5
6	9SH	6
7	10AM	7
8	10ET	8

I will only require two students to be created for the purposes of testing, and their credentials will be as follows.

StudentID	SecondName	FirstName	CurrentYear	DetentionCount	FormID
1	Jobs	Mark	7	2	1
2	Sanders	Alex	8	0	2

And finally, 2 existing detention records.

DetentionID	EventDate	TimeStart	TimeEnd	Duration	RoomID	TeacherID	StudentID	Offense
1	2017-03-21	940	960	20	1	1	1	Late
2	2017-03-22	960	990	30	7	1	1	Rude

Input test data

The following test data will be created for the purpose of validation. For example, I will have different password variants, which should be able to prove whether the validation works or not. Some of these may not be standard inputs, for example, I might have an input be “press this button as fast as possible for 5 seconds”, to see if a crash may occur.

These are examples of passwords that I will use to attempt to test my validation.

Password example	Intended error of password
“hello”	No capitals or digits. Insufficient length.
“password123”	No capitals
“Password123”	No errors
“Password”	No digits

“Pass12”	Insufficient length
“Pass$word123”	Special characters used
“Pass%1”	Special characters used, insufficient length
“Password123Password”	Too long

Due to a lot of my success criteria being dependant on validation of detention bookings, I will carry out various attempts to create the following detention records. Some will be designed to be valid, and so that they should create the records successfully, and some should be identified as invalid. They will also serve to create the desired output of the detention slip, so this will be checked too.

DetentionID	EventDate	TimeStart	TimeEnd	Duration	RoomID	TeacherID	StudentID	Offense
3	2017-03-21	950	980	30	2	1	1	Test1
4	2017-03-21	930	950	20	1	1	2	Test2
5	2017-03-22	950	970	20	1	1	1	Test3
6	2017/03/22	945	965	20	8	3	2	Test4

Of the above records, I expect only number 6 to go through, if my program works as intended. 3 and 4 should not go through due to them overlapping with detention record 1 (already input data). Detention 3 is in the same room, at the same date and at the same time. Detention 4 is with the same student, at the same time and on the same date. 5 will be rejected, due to it overlapping the same student with detention 2 (already input data) and being on the same date and at the same time. It is on a different date to detention 1 however, so it will not clash with that detention. Detention 6 should work, as it is in a different room and with a different student than all existing detention records. However, this will only be confirmed during the testing of my program.

The final data that will be input is regarding creating new users. I will attempt to create the following users, and if all works as intended, then the program should reject some and accept others.

TeacherID	Surname	FirstName	Administrator?	Email	FormID
(dependent on number of existing records)	User	New	YES	email@mail.com	5

If all works as intended, record 6 should be implemented successfully.

When will tests occur

All the tests mentioned in the below “testing success criteria”, will be able to be carried out during the development process. Specifically, tests will be carried out after the intended functionality has been implemented, to ensure that it is working as intended.

For example, as soon as the database has been created, I will attempt to fill in the “existing test data” to observe what happens.

Another example may be how the “GUI testing” will be carried out for each widget when the functionality of a given widget has been completed.

That being said, all of the tests within the “testing success criteria” and “GUI testing” will be carried out again during my evaluation.

Testing success criteria

This plan, showing relevant test data, will show what will be tested during the iterative development process. The tests will be done at the end of each feature being developed for a given window, to confirm if they work as intended. If they do not, then amendments may be made to the code, allowing these tests to be carried out once again.

Once development is complete, these tests will be carried out once again post-development, for use in evaluation.

Initialisation and login

Objective	How it will be tested	Test data used	Expected outcome
1.	Attempting to run the program	Program installed and GUI files removed from the file.	Program to alert user that essential resources are missing from the program. Attempt reinstall.
2-6.	Attempting to use the system, both with no account, and with an existing account.	Using both the accounts listed in the "test users" table, under the "existing test data" heading.	Access only granted to system with a successful login, program notifying user with which credentials were incorrect.
7.	Sign out button pressed.	n/a	Taken back into login screen, variables related to login credentials also reset.
8.	Observation of raw database confirms that passwords of users are not stored in their standard form.	Password "BrownB1" compared with password stored within database for that user.	Observation confirms that the password is stored in a completely different form.
12.	The "cannot login?" button was pressed in the login window.	BrownB1 in "test users" table. Email: "BrownB@mailcom"	Pop-up containing the email of an administrator popped up on screen.

GUI

Objective	How it will be tested	Test data used	Expected outcome
20.	Use of the program with a monitor of recommended screen resolution (1366x768) or greater. Tests carried out on multiple devices.	n/a	No scrolling ever necessary.
21.	Use of program.	n/a	No colours outside of white, grey, black and light blue were present outside of the RGS placeholder logos.

22.	Use of program, specifically pressing back buttons.	n/a	Users were able to navigate back to the previous window, from any other window once logged in.
23.	Use of program.	n/a	Pop-up windows created, giving users information about the program.
24.	Observation of number of buttons per window is carried out.	n/a	Number of buttons never exceeded 5.
25.	Observation of number of total widgets per window is carried out.	n/a	Number of widgets never exceeded 9.
26.	Final output provided by the program is attempted to be achieved by using 1 word inputs at most.	n/a	Output achieved as expected.
27.	Statement from external test user received, determining if frames are used to group any widgets together.	n/a	Statement concludes that frames were used to group widgets together.
28.	User attempts to navigate to all windows from other windows, once logged in, by only using 3 clicks.	Test user 1 (“BrownB1”)	User able to reach any window from any point, without using more than 3 clicks once logged in.
29.	External test user gives statement determining if any combo boxes were used for receiving inputs.	n/a	Statement concludes that combo boxes were used to take inputs.

Database

Objective	How it will be tested	Test data used	Expected outcome
32.	Database observed.	n/a	Can be concluded that database is fully normalised.
33-37.	Database observed, screenshots to be used as evidence of different tables.	n/a	Screenshots prove there are 5 distinct tables in the database.

38-48.	Database observations and screenshots provide evidence	n/a	Screenshots provide evidence for conclusion that all the different queries were successful.

Passwords

Objective	How it will be tested	Test data used	Expected outcome
9-10.	Test data used to attempt creating various passwords.	See password examples under "input test data".	"Password123" to be the only accepted passwords. All others will create a pop-up, notifying user of incorrect password.
11.	Change password feature used.	"Password123", and one of the "test users".	Database will show the password "Password123" recorded in the database for the given test user.

Administrative users

Objective	How it will be tested	Test data used	Expected outcome
13,31.	Test users "BrownB1" and "AllenT3" will attempt to access the "manage users" window.	See test users table under "existing test data". "BrownB1" and "AllenT3".	Program will deny access to "AllenT3", but allow "BrownB1". Pop-up informs AllenT3 that they must be an admin.
14.	Test user 1 will attempt to reset password of test user 2.	Test user 1 = "BrownB1". Test user 2 = "AllenT3".	Database will show hashed version of "123" in database for test user 2's password.
15.	Test user 1 will attempt to make test user 2 an admin.	Test user 1 = "BrownB1". Test user 2 = "AllenT3".	Database will show test user 2 to be an administrator, where they were previously not.
16-19.	Test user 1 will attempt to create a new user.	See creating new users table, under "input test data".	The record should be successfully created and added to the database.

Output

Objective	How it will be tested	Test data used	Expected outcome
49-50.	Function used to book a detention, and the output is observed.	See test detention bookings table, under "input test data".	If detention is successful, it will create a HTML for with the credentials of the detention. This should appear similar to the RGS form.
51.	Functionality used to save it as an XML file.	^	Screenshot proves XML file was successfully saved.

52.	User attempts to email the saved XML file.	^	XML file successfully received by recipient, and of the form expected.
53.	User attempted to print XML file.	^	XML file was successfully printed and of the expected form.
54-55.	Use of graph functionality.	See existing detention records, under “existing test data”.	Graph created using HTML, CSS and JavaScript to show trends in student’s detentions.
56.	User attempts to print created XML file of graph.	^	XML file was successfully printed and of the expected form.
57-59.	Statistical analysis functionality used and output observed.	^	Various outputs able to be produced from existing detention records.
60.	Raw view of database functionality was tested, and the output was observed.	See all tables under “existing test data”.	Raw view of the database was available from the DetentionViewer window.

GUI testing

The following tests will be carried out the purpose of examining the stability of the GUI. Many of the tests involving the GUI have already been addressed in the above test plan, however these are more destructive test which don't necessarily refer to my success criteria.

Button testing

For buttons that do not open another window, I will perform "crash tests". These will be done by clicking the button as fast as I can, and observing what happens. Hopefully, the program will continue to function. However, if it does not for any reason, then I may try and find a solution for this problem. An example of one of these buttons may be the "Make Admin" button.

Navigation buttons

Similar to the above tests, I will carry out "crash tests" on buttons which have the purpose of navigating to other windows. An example is the login button, though back buttons and the likes will also have similar tests.

Input widget testing

These types of tests have already been accounted for by use validation tests. I will attempt to input various values into these boxes to observe the outcomes. If any unexpected crash/error is created, then I will attempt to address the problem, and re-attempt the tests.

Developing the solution

As discussed in the design of the solution, I will be creating the database as my first step. Once this is complete, I will go about creating each window with the code that accompanies it. I will only move on from a window once I have achieved the criteria that I initially assigned to it in my analysis of the problem. To conclude that I have achieved the criteria, I will use the test plan that was previously displayed within the design section.

Database

For any of my system to work, I must have a working database, hence why I will begin to develop it first. As discussed in the analysis of the problem, the most appropriate database software for my needs is SQLite 3. I have already concluded how the structure of my database will be (refer to design section of this document), which means all that is left is to create the database.

For the purpose of testing and demonstration, I have populated the database with the records that can be found under the “existing test data” heading in the design section.

Detentions

The first table is the “Detentions” table, which will act as the transaction table. The purpose of this table is to record all detentions and the relevant information surrounding them. Most queries used to search the database will be relevant to this table.

Detentions

```
CREATE TABLE "Detentions" (
        `DetentionID`   INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
        `EventDate`     TEXT NOT NULL,
        `TimeStart`     TEXT NOT NULL,
        `TimeEnd`       INTEGER NOT NULL,
        `Duration`      INTEGER NOT NULL,
        `RoomID`        INTEGER NOT NULL,
        `TeacherID`     INTEGER NOT NULL,
        `StudentID`     INTEGER NOT NULL,
        `Offense`       TEXT
)
```

	DetentionID	EventDate	TimeStart	TimeEnd	Duration	RoomID	TeacherID	StudentID	Offense
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	2017-03-21	940	960	20	1	1	1	Late
2	2	2017-03-22	960	990	30	7	1	1	Rude

Forms

The “Forms” table will be a small table, which is required to keep the database in third normal form. It will only have 3 fields in it for this reason.

Forms

```
CREATE TABLE `Forms` (
        `FormID`        INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
        `FormName`      TEXT NOT NULL UNIQUE,
        `TeacherID`     INTEGER NOT NULL UNIQUE
)
```

	FormID	FormName	TeacherID
	Filter	Filter	Filter
1	1	7BB	1
2	2	7SL	2
3	3	8AT	3
4	4	8HS	4
5	5	9CM	5
6	6	9SH	6
7	7	10AM	7
8	8	10ET	8

Rooms

This table, like the prior table, will be a small table, used primarily for keeping the database in third normal form.

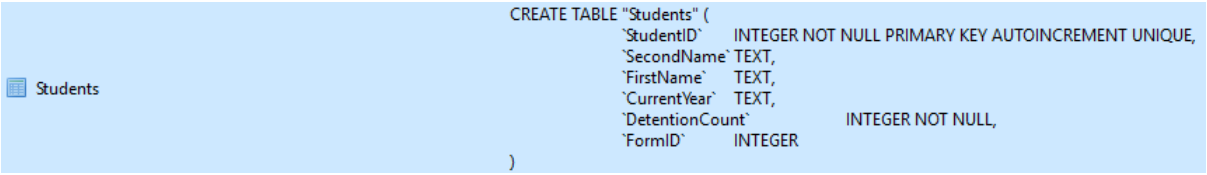
Rooms

```
CREATE TABLE `Rooms` (
	`RoomID`	INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
	`Block`	TEXT NOT NULL,
	`RoomNumber`	TEXT NOT NULL UNIQUE
)
```

	RoomID	Block	RoomNumber
	Filter	Filter	Filter
1	1	Junior	1
2	2	Junior	2
3	3	Junior	3
4	4	Junior	4
5	5	Junior	5
6	6	Junior	6
7	7	Main	22
8	8	Main	23
9	9	Main	24
10	10	Main	25
11	11	Main	26
12	12	Main	27
13	13	Maths	31
14	14	Maths	32
15	15	Maths	33
16	16	Maths	34
17	17	Maths	35
18	18	Maths	36

Students

The “Students” table will record all the relevant data to the students at the school. This table will be used a lot for queries about statistics of which students are receiving detentions.



	StudentID	SecondName	FirstName	CurrentYear	DetentionCount	FormID
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Jobs	Mark	7	2	1
2	2	Sanders	Alex	7	0	2
3	3	White	Hugo	9	0	6
4	4	Taylor	Thomas	7	0	1
5	5	Harris	Matthew	10	0	8
6	6	Mayers	Michael	10	0	8
7	7	Clack	Dexter	8	0	4
8	8	Dixon	Daniel	10	0	7
9	9	Vase	Jason	8	0	3
10	10	Lawrance	Noah	9	0	5

Teachers

This table is going to be used to store all relevant information about teachers, while also storing vital data that allows them to have a function account to the program, such as ‘TeacherUser’, ‘Administrator’ and ‘Password’.

Teachers

```
CREATE TABLE "Teachers" (
	`TeacherID`	INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
	`SecondName`	TEXT NOT NULL,
	`FirstName`	TEXT NOT NULL,
	`TeacherUser`	TEXT NOT NULL UNIQUE,
	`Administrator`	TEXT NOT NULL,
	`Password`	TEXT NOT NULL,
	`Email`	TEXT UNIQUE,
	`FormID`	TEXT
)
```

	TeacherID	SecondName	FirstName	TeacherUser	Administrator	Password	Email	FormID
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Brown	Bob	BrownB1	YES	c1deaf40c8af...	brownbob@m...	1
2	2	Smith	Lisa	SmithL2	YES	d4735e3a265...	smith88@mail...	2
3	3	Allen	Tim	AllenT3	NO	06ba0e411cfc...	allenallen@m...	3
4	4	Hunt	Sam	HuntS4	NO	4b227777d4d...	hunt87@mail....	4
5	5	Cook	Megan	CookM5	NO	ef2d127de37b...	meganc@mail...	5
6	6	Scott	Holly	ScottH6	NO	e7f6c011776e...	hs2@mail.com	6
7	7	Adams	Madison	AdamsM7	NO	7902699be42...	madams@ma...	7
8	8	Edwards	Tom	EdwardsT8	YES	2c624232cdd...	tomedwards...	8

Overview

Name	Type	Schema
Tables (6)		
Detentions		CREATE TABLE "Detentions" (`DetentionID` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, `EventDate` TEXT NOT NULL, `TimeStart` TEXT NOT NULL, `TimeEnd` INTEGER NOT NULL, `Duration` INTEGER NOT NULL, `RoomID` INTEGER NOT NULL, `TeacherID` INTEGER NOT NULL, `StudentID` INTEGER NOT NULL, `Offense` TEXT)
Forms		CREATE TABLE `Forms` (`FormID` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, `FormName` TEXT NOT NULL UNIQUE, `TeacherID` INTEGER NOT NULL UNIQUE)
Rooms		CREATE TABLE `Rooms` (`RoomID` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, `Block` TEXT NOT NULL, `RoomNumber` TEXT NOT NULL UNIQUE)
Students		CREATE TABLE "Students" (`StudentID` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, `SecondName` TEXT, `FirstName` TEXT, `CurrentYear` TEXT, `DetentionCount` INTEGER NOT NULL, `FormID` INTEGER)
Teachers		CREATE TABLE "Teachers" (`TeacherID` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, `SecondName` TEXT NOT NULL, `FirstName` TEXT NOT NULL, `TeacherUser` TEXT NOT NULL UNIQUE, `Administrator` TEXT NOT NULL, `Password` TEXT NOT NULL, `Email` TEXT UNIQUE, `FormID` TEXT)

Due to SQLite3 only supporting the datatypes of TEXT(string) and INTEGER(integer), I will have to handle all validation outside of the database. For example, if the user were to attempt to put “abc” as an input for “is the user an administrator?”, when attempting to create a new user, the database would accept it. Unless, I ensure that only “YES” or “NO” can be input when creating my program.

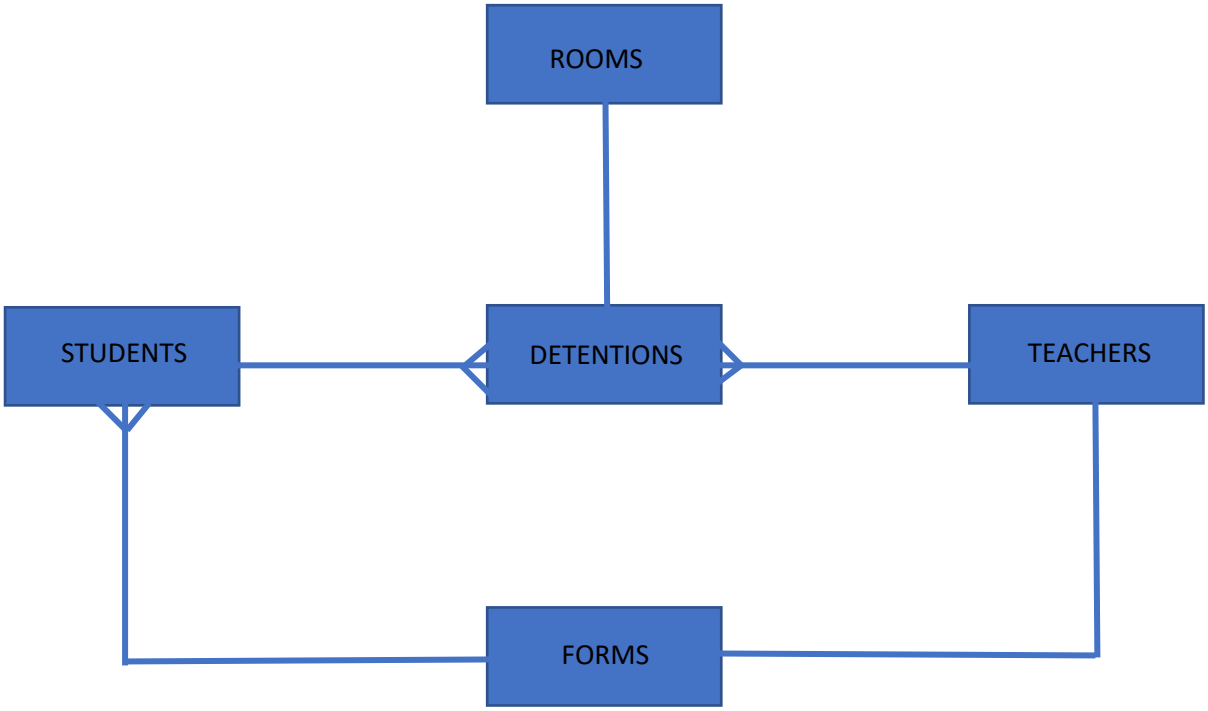
This limitation to SQLite3 explains why I have the “Administrator” field set to hold TEXT, when in reality, I will be treating that field as a Boolean value.

Testing: database

As there are various success criteria which are relevant to the database, I will attempt to test them now that I have fully constructed my database.

32. *" SQL database will have no many-many relationships that cause data redundancy."*

To prove I achieved this success criteria, I will show the ERD of this database, which demonstrates there are no many to many relationships.



As observed, it may be concluded that this database has no many to many relationships.

33. *"Database will record all relevant details of teachers in a table…"*

34. *"…students in a table…"*

35. *"…form groups in a table…"*

36. *"…detention rooms in a table…"*

37. *"…and detentions in a table"*

The above 5 objectives may be proven to be achieved by observing the 5 different tables that I have created within the database.

Graphical User Interface

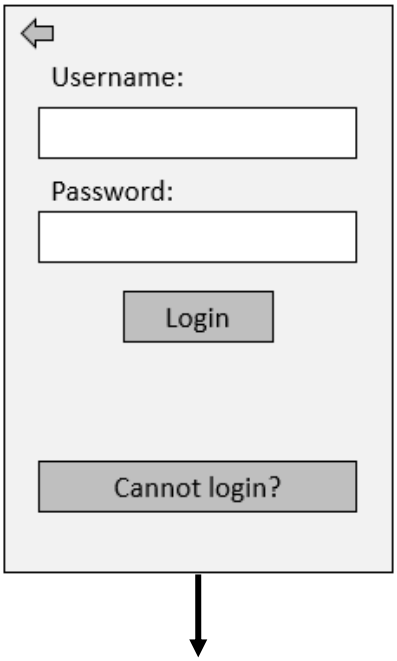
Initially, I created each window using QtDesigner (version 4.8.4), based off my design which may be seen in the design section of this document. When I created each window, I did it considering objectives 20-29, which I will test here to prove that I have achieved them.

Login screen

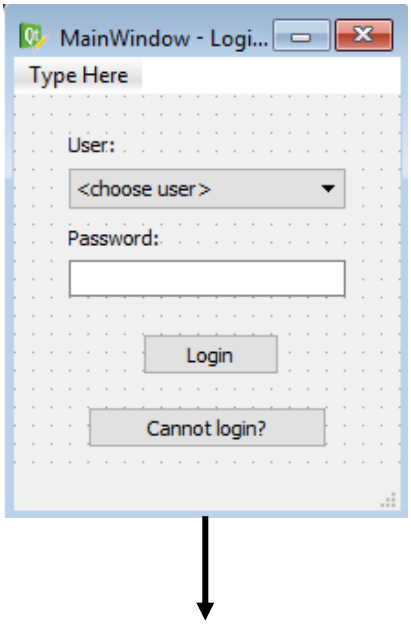
The first logical step for my development process was to create the login screen, as this would be the first window that the user will encounter when using the program. Since most of the features in this program are dependent on the current user as well, it is vital that I can ensure who the user is. The most efficient way of doing this, is to enforce authentication. This simultaneously will enforce that user's must have an account to use the functionality of the system.

Iterations of GUI development

These screenshots demonstrate how the window looked at different stages of development. I changed how I wanted this window to look as development went on, and I will justify why I made each change.



The window on the left was made using Microsoft PowerPoint, with its shape tools. Although not aesthetically impressive, it serves the purpose of displaying the various input widgets that were required for the program to function.



This window was the first real development that I created in QtDesigner. The only difference it has from the above box, is that it uses a combo box for the username input. The reason I initially thought this beneficial was due to how it would allow for easier access to users.

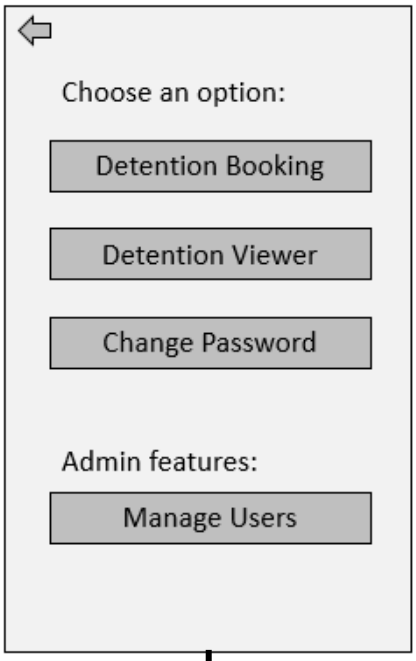
My next iteration of this window included an RGS placeholder logo, to improve the aesthetics of the login screen. This logo could easily be changed to suit the school that is using it.

The next iteration is the final version, which involved the return of the username input to a line edit, as opposed to a combo box. The reason I settled on this decision, is because of both security and efficiency. I came to realise that as more users were added, the list provided by the combo box could become very long to the point where it wouldn't save time to login. Also, if the username is already there, then it would give someone who is trying to illegally access the system an easier time, as they would only have to guess one credential as opposed to two.

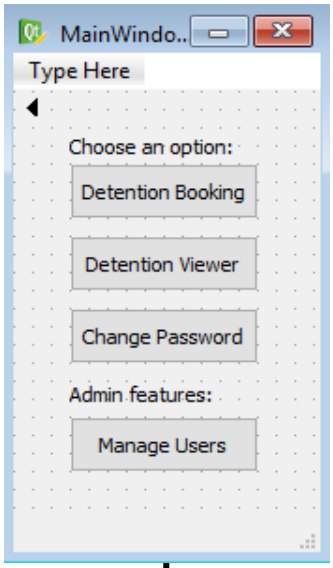
This screenshot simply shows how the window looks when the program is run, the main different being that I have included an RGS placeholder in the top left of the window, to make it more associated with the given school it is being used with.

Menu screen

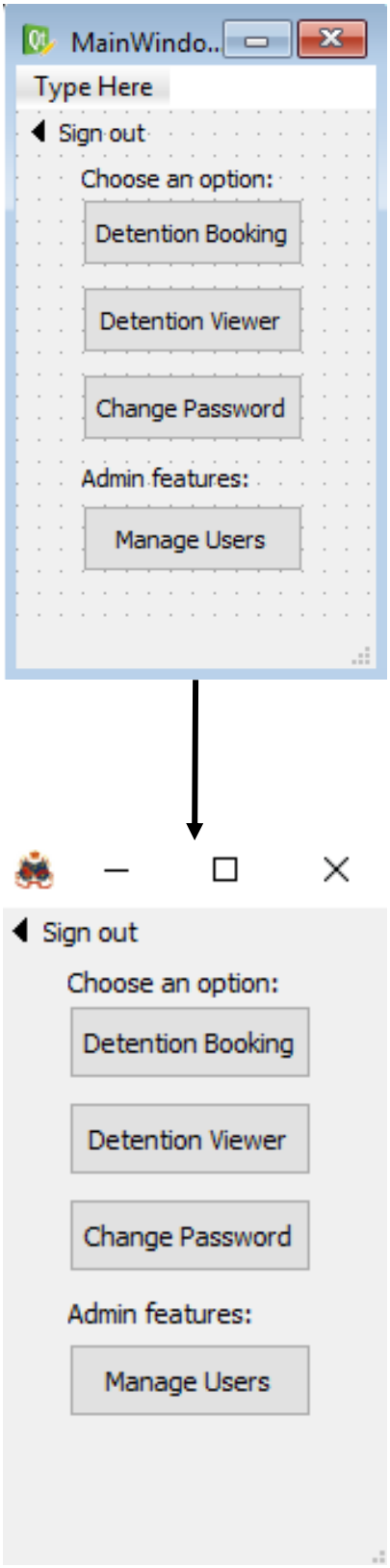
Iterations of GUI development



The initial design for the menu window is as shown here. Again, it has very little to show aesthetically, but serves the purpose of providing the buttons that allow all the necessary navigation throughout the program.



Nothing at all was changed with the first iteration of this design.

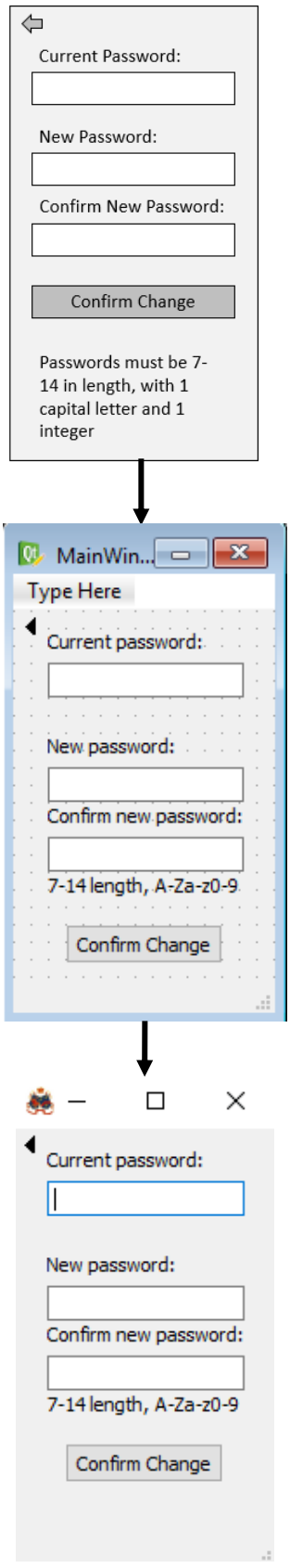


Only a single change was made from the design to final iteration, which was that a label was created saying “sign out” next to the back button. This was done because that is what this back button did. It will clear the credentials of the user from the program, and reset the program to the state that it is when it is first opened.

This image simply shows how the window looks in the program.

Change password screen

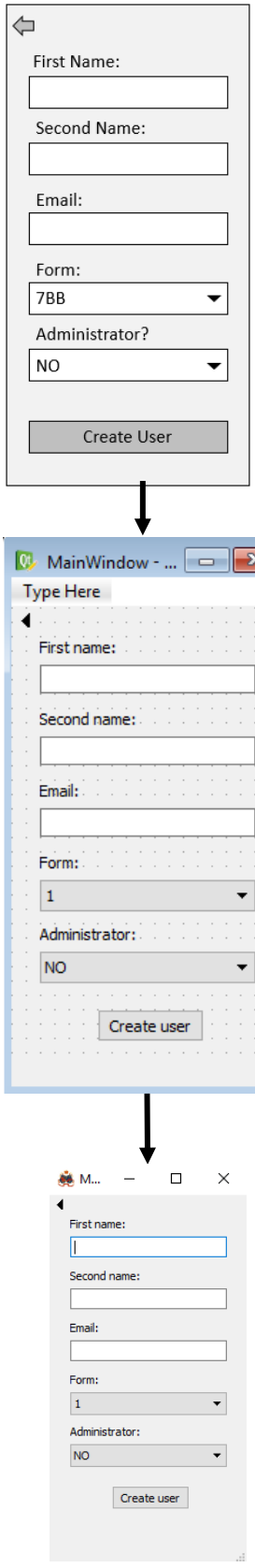
Iterations of GUI development



The only change that may be observed from the initial design is the label that notifies the user about the password formatting has been made more concise. I chose to do this as it appeared far too long in the initial design, when one of my objectives involved reducing the amount of present clutter.

Create new user screen

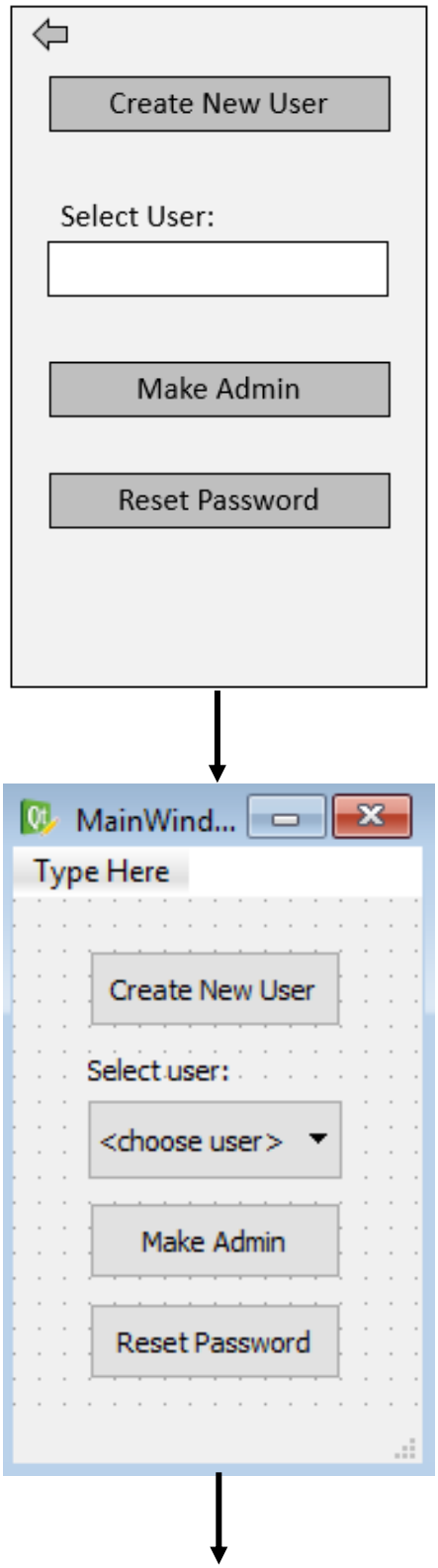
Iterations of GUI development



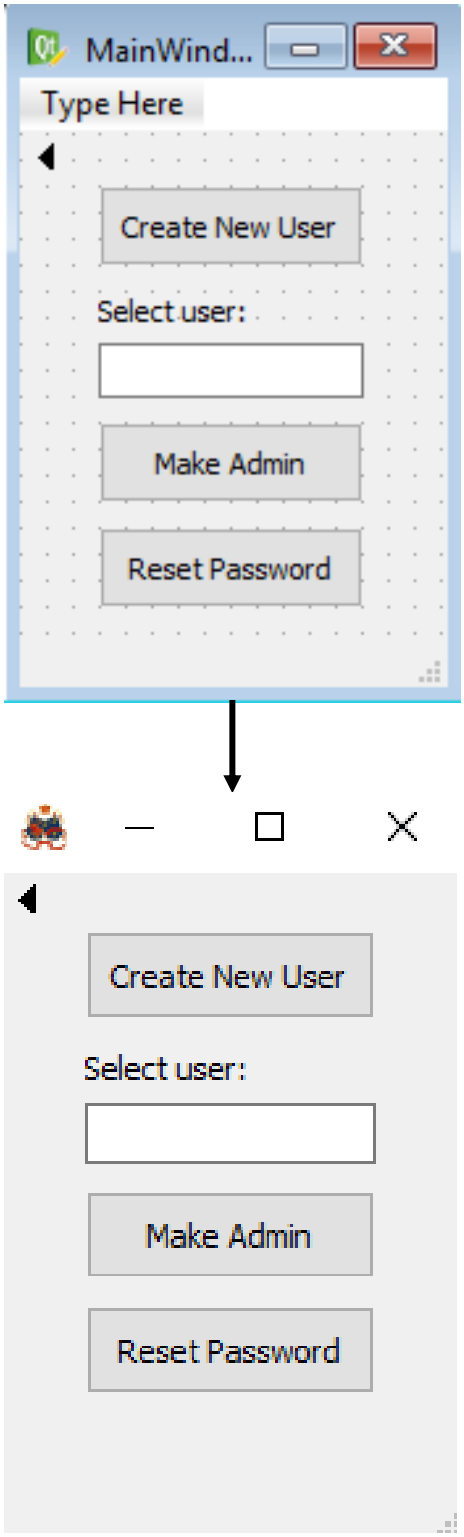
Similar to the change password screen, I was happy enough with the initial design of my create new users window that I changed nothing about it when developing the real version.

Manage users screen

Iterations of GUI development



Initially, I decided to alter the original design by having a combo box for the "select user" input. The reason I did this was like the reason I had a combo box for the username input on the login screen. I thought it would be faster for the user to choose from a list of existing users. To some extent this is true, as it does not allow for invalid usernames, however, if there are a lot of users, then it becomes far more efficient to have the user input the credentials directly.

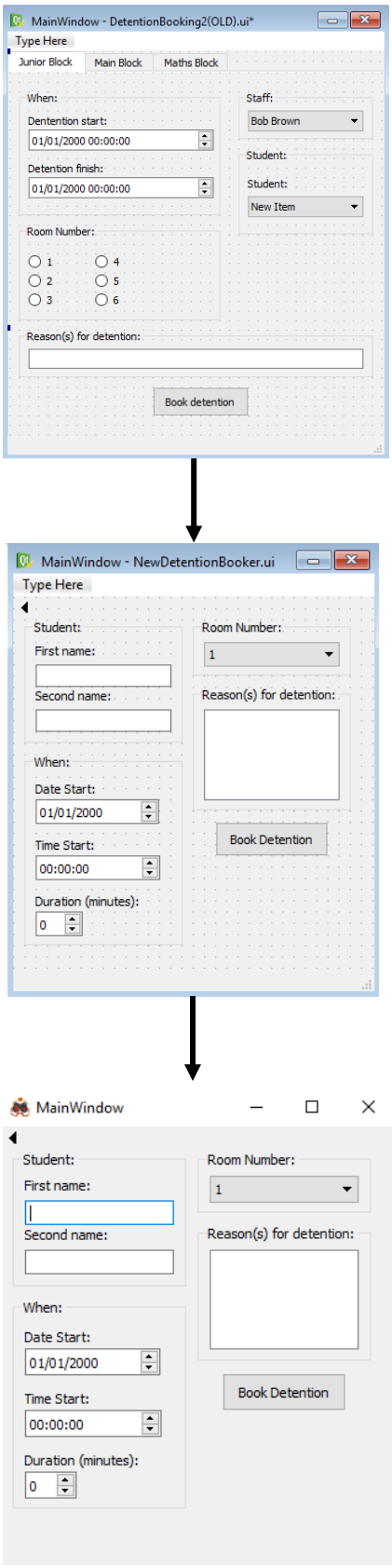


This window shows how I removed the combo box, and replaced it with a line edit. This simply means that I will have validation of the “select user” input outside of the GUI, and in my code. The best way for me to do this will most likely be to run a database query that checks if the input username exists.

Once again, this final screenshot simply shows how the window looks when the program is run.

Detention booking screen

Iterations of GUI development



The first version of the detention booking window that I created in QtDesigner is very different from the final version in many ways. Firstly, it uses combo boxes for both the staff input, and the student input. The combo boxes were removed for the reasons that have been mentioned for other windows, and the staff input was removed altogether. This was done because the issuer of the detention is already known since the program record who the current user is.

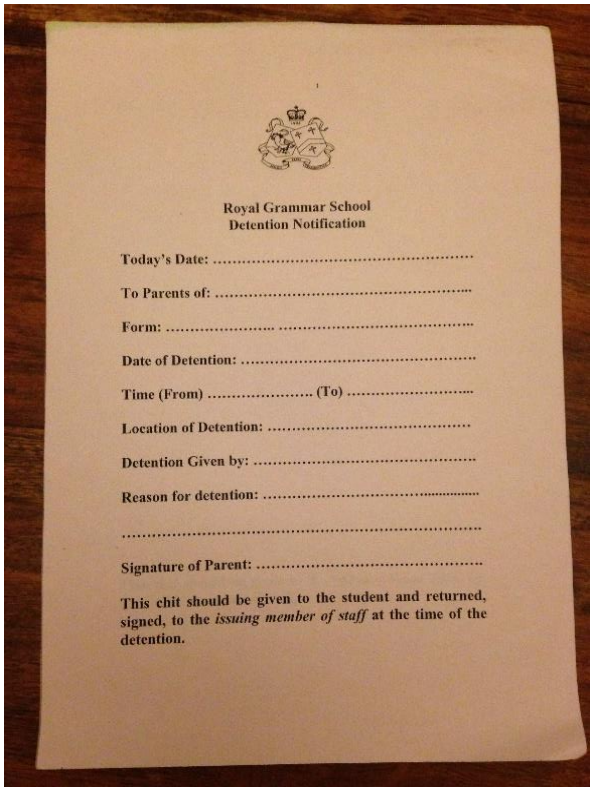
Other things to note about this iteration of the window is that the room number is given by radio buttons and tabs that determine the block. The reason I chose to avoid this, is by using a combo box for the room number, I can reduce the amount of clutter from radio buttons, and simultaneously make the tabs redundant. As seen in the second iteration, there is less clutter.

Finally, I also changed the method or inputting time and dates. Originally, a time date input was used to give the start time/date of the detention and the end time/date of the detention. From these inputs, the duration could be determined. However, I feel that users will not be as likely to consider an end time above the duration, and so it will save them time to consider what the duration would be if they can just input it themselves. The program can then determine what the end time would be after the fact. I have decided that putting the time and date inputs separate is beneficial as no detentions will extend over multiple dates. This means one date input can be taken, considering it will always be the same at the start and the end of the detention.

Detention form

I designed the detention form based upon the physical paper slip form that is used by the RGS currently. One of my objectives was to be able to effectively recreate this detention slip in a digital format, and so I will show a comparison to determine whether I have achieved this objective or not.

50. *"…which has been designed to replicate the physical form currently used by the RGS…"* (referring to the detention form).



Royal Grammar School
Detention Notification

Today's Date: .....

To Parents of: .....

Form: .....

Date of Detention: .....

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

Location of Detention: .....

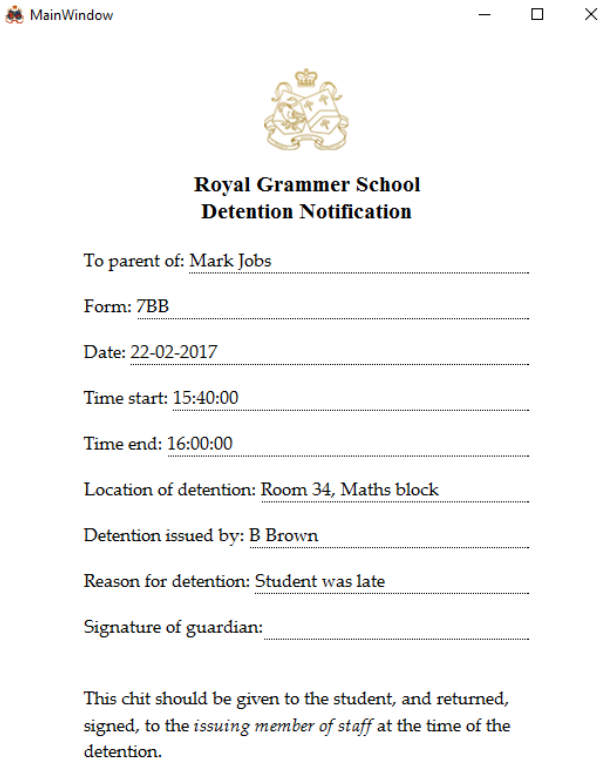
Detention Given by: .....

Reason for detention: .....

.....

Signature of Parent: .....

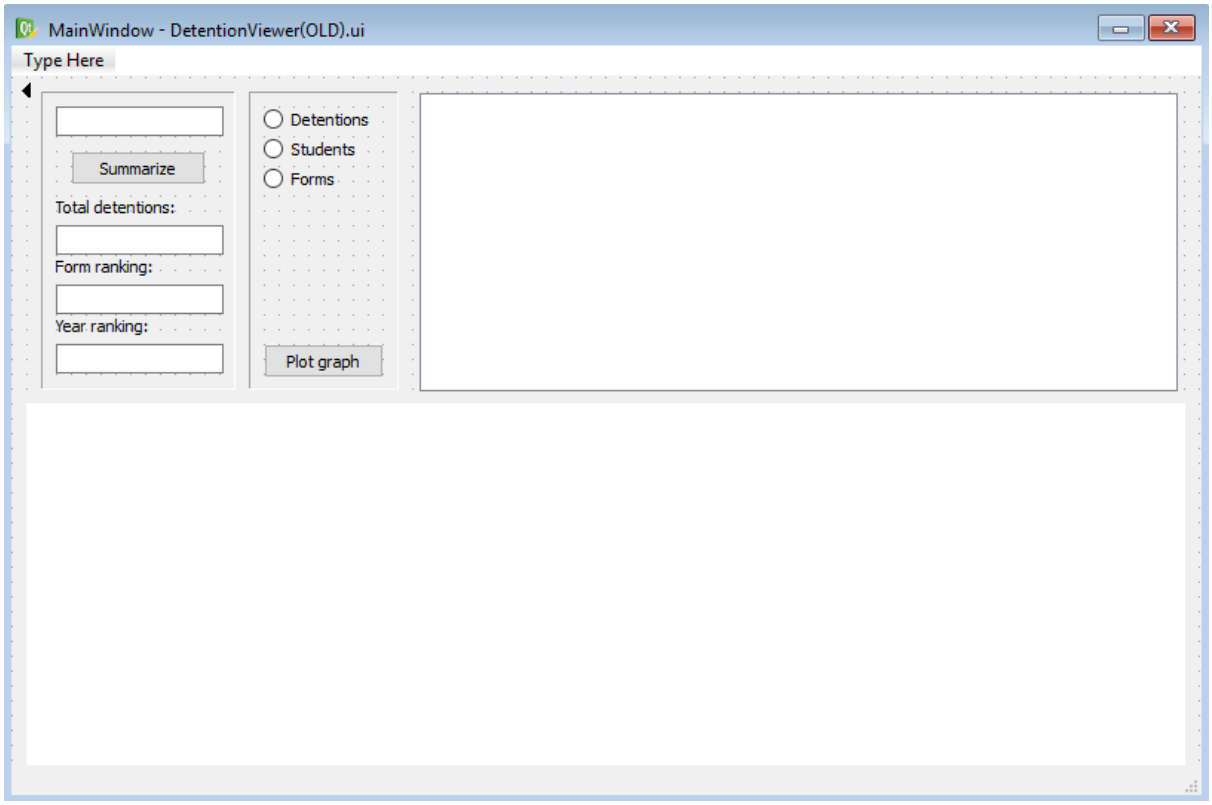
This chit should be given to the student and returned, signed, to the *issuing member of staff* at the time of the detention.



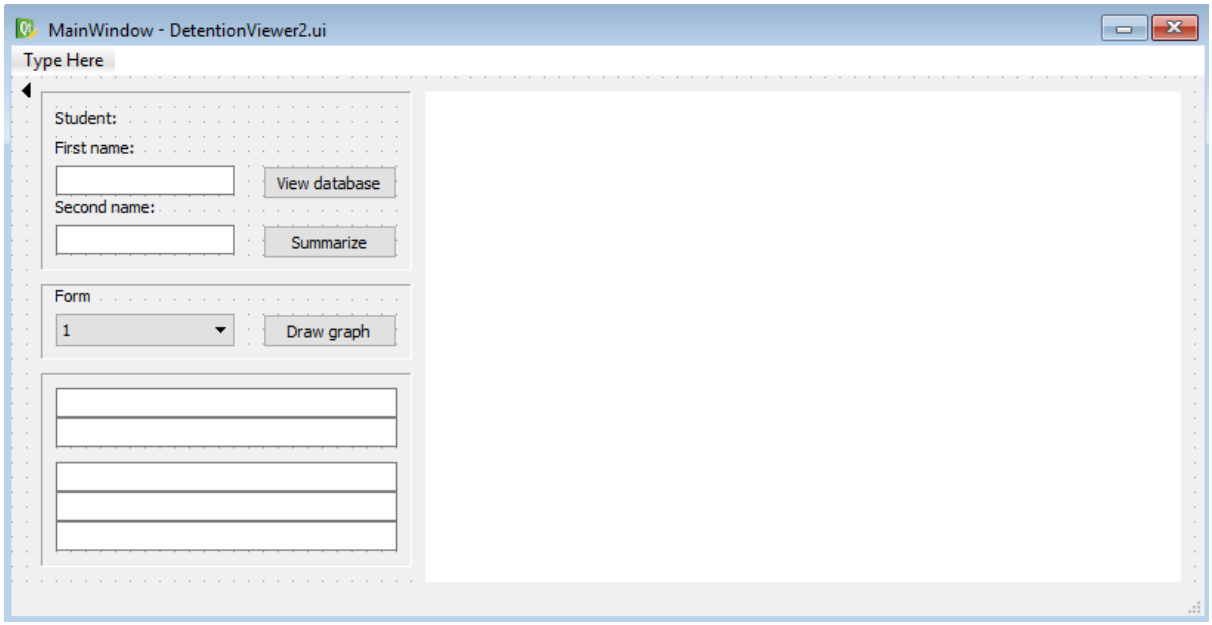
As shown in this comparison (the detention form created by the program using HTML, CSS and JavaScript is on the right), objective 50 has been achieved.

Detention viewer window

Iterations of GUI development



The above image shows the first version of my detention viewer window. It has a web viewer (bottom), and a table viewer (top right). The main reason it is different from the final version, is due to how the final version will open up a new window showing exclusively the table viewer, much like how the detention form is shown as its own window. The buttons have also been reworked, so that they have different functions. The new version has line edits to display statements to the user.

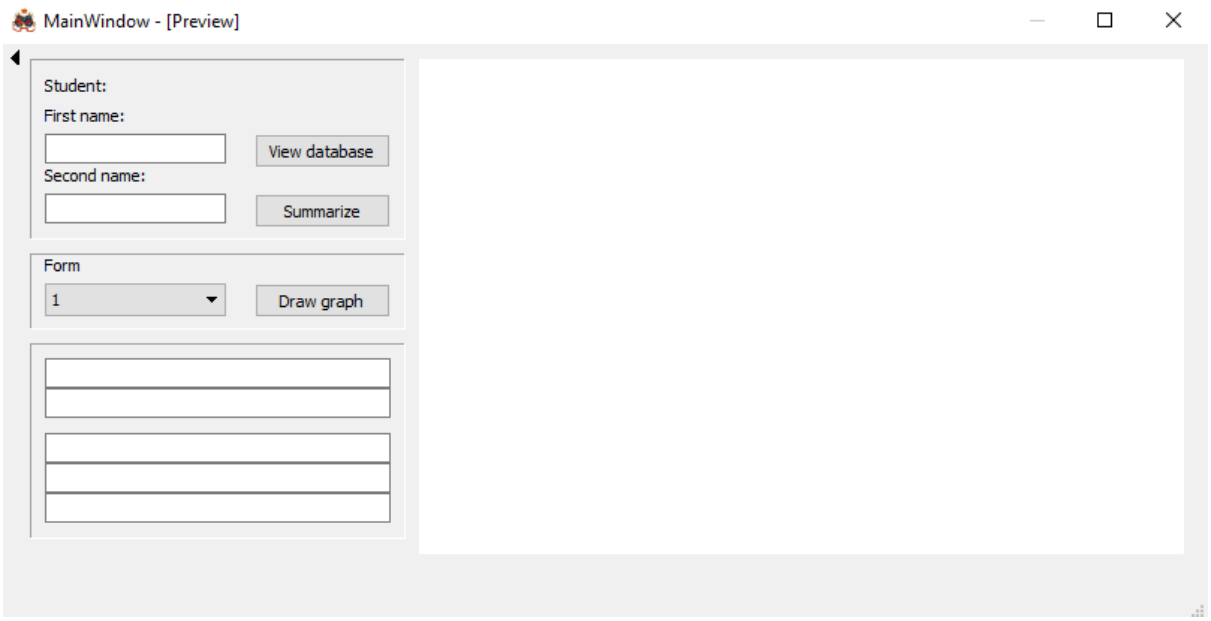


Testing: GUI

As all the different windows have been created, it is possible for the GUI to be reviewed to determine if the success criteria relating to it have been met.

20. *"Windows will be small, requiring no scrolling and never occupying more than half of the screens space, provided they have the recommended screen resolution of 1366x768 or higher."*

To test this above objective, I will use a monitor with a resolution of 1366x768 and take a screenshot of the largest window to determine that no scrolling is ever necessary.



As seen in this screenshot, scrolling is not possible, due to the whole window being visible with a recommended monitor size. Therefore, I can conclude this objective has been achieved.

21. *"Interface will use very little colours, limited to white, grey, black and light blue, outside of the RGS placeholder logos, to cater for colour-blind users"*

This objective may be reviewed by simply looking at all the above screen shots which show how the windows look when the program is run. From these, it is easily concluded that this objective has been achieved.

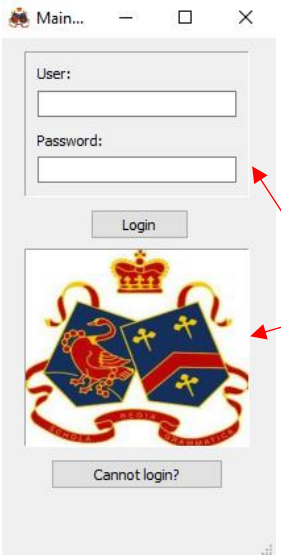
24. *"Minimal clutter will be enforced by limiting number of buttons per window to 5…"*

25. *"…and maximum number of widgets per window to 9"*

These are two more objectives which can be tested simply by looking at the above screenshots of all the windows. It may be concluded by counting that all the windows conform to these rules. Therefore, I can view both objectives as successfully achieved.

27. *"Frames will be used to create a "sunken" effect, grouping widgets together, further reducing clutter"*

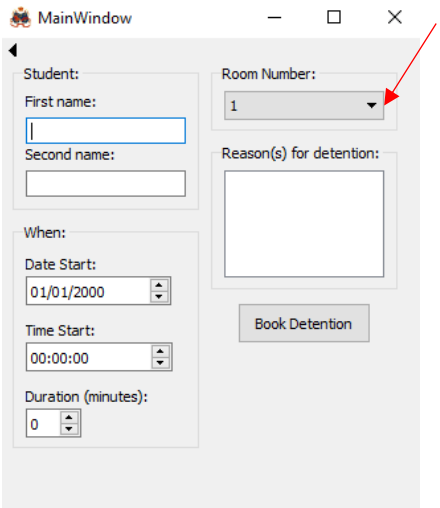
This is another objective which can be determined as successful or not by viewing screenshots of the windows as they appear in the program.



It can be seen in this window for example, that a frame has been used to create a sunken effect, which groups together the input boxes for the user's credentials. This in turn makes it appear far easier to see what is going on. Another frame is used in this window to box the RGS logo in as well. This makes the whole window look far neater.

29. *"Combo boxes will be used where possible, to reduce the possibility of user error…"*

This is yet another objective that may be tested simply by reviewing screenshots of the windows as they appear in the program. An example of one being used may be found in the detention booking window.



Here a combo box is used to take the input for a room number. As the inputs are pre-determined by me, it means that there is very little room for error. The only error could arise from putting an invalid input in the list of options from the combo box.

Code development

In this section I will provide annotated screen shots that will demonstrate the iterative development process that I carried out. I will also provide evidence of any tests that were carried out, with their result. To ensure these tests are valid, I will be using python's debugger built within IDLE. I will use it by creating breakpoints in my code, combined with variable watching to provide evidence of codes success or failure.

I have taken screenshots of my project folder, to evidence the iterative development process, as it includes many different versions of my program. However, it will not show all the versions, due to the nature of how many different versions there are.

The most logical division for sections of my program is by the different windows (which may be seen in the section above), therefore I will show the stages of development for the code relating to each window individually.

However, before I started to develop the code for my windows, I created the base skeleton of my system, which was the code that loaded the database, as well as other important imports that I knew I was going to be using. These may be seen here.

```
#### IMPORTS ####
import sys, os

from PyQt4 import QtCore, QtGui, uic
from datetime import timedelta, datetime
from dateutil import relativedelta

import sqlite3 as lite #SQL import
con = lite.connect('DetentionPlanner.db') #Importing database
cur = con.cursor()

import hashlib, binascii #Hashing imports
import re #Regex import
```

Login window

The first logical step for my development process was to create the login screen, as this would be the first window that the user will encounter when using the program. Since most of the features in this program are dependent on the current user as well, it is vital that I can ensure who the user is. The most efficient way of doing this, is to enforce authentication.

Login v1 code

The first version of the login code may be seen here.

```
LoginWindow = uic.loadUiType("Login(OLD).ui")[0]


password = ""

class LoginWindowClass(QtGui.QMainWindow, LoginWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        ##################
        self.BTN_Login.clicked.connect(self.Login)
        ##################
        self.password=""

    def Login(self):
        global password
        global user
        user=str(self.CB_User.currentText())
        print(user)
        fetchpass=("""SELECT Password FROM Teachers
        WHERE TeacherUser = ?""")
        cur.execute(fetchpass, (user,))
        password=cur.fetchone()[0]
        print(password)
        self.password=str(self.LE_Password.text())
        print(self.password)
        if hashlib.sha256(self.password.encode()).hexdigest() == password:
            print("window will open here")
        else:
            print("Password does not match with user.")


app = QtGui.QApplication(sys.argv)
LoginWindow = LoginWindowClass(None)
LoginWindow.show()
app.exec_()
```

As shown in the code on the left, I have used an object-orientated approach. Each window will be a class. However, as this was my first version of the program, it only contains one window and therefore one class at this point in time.

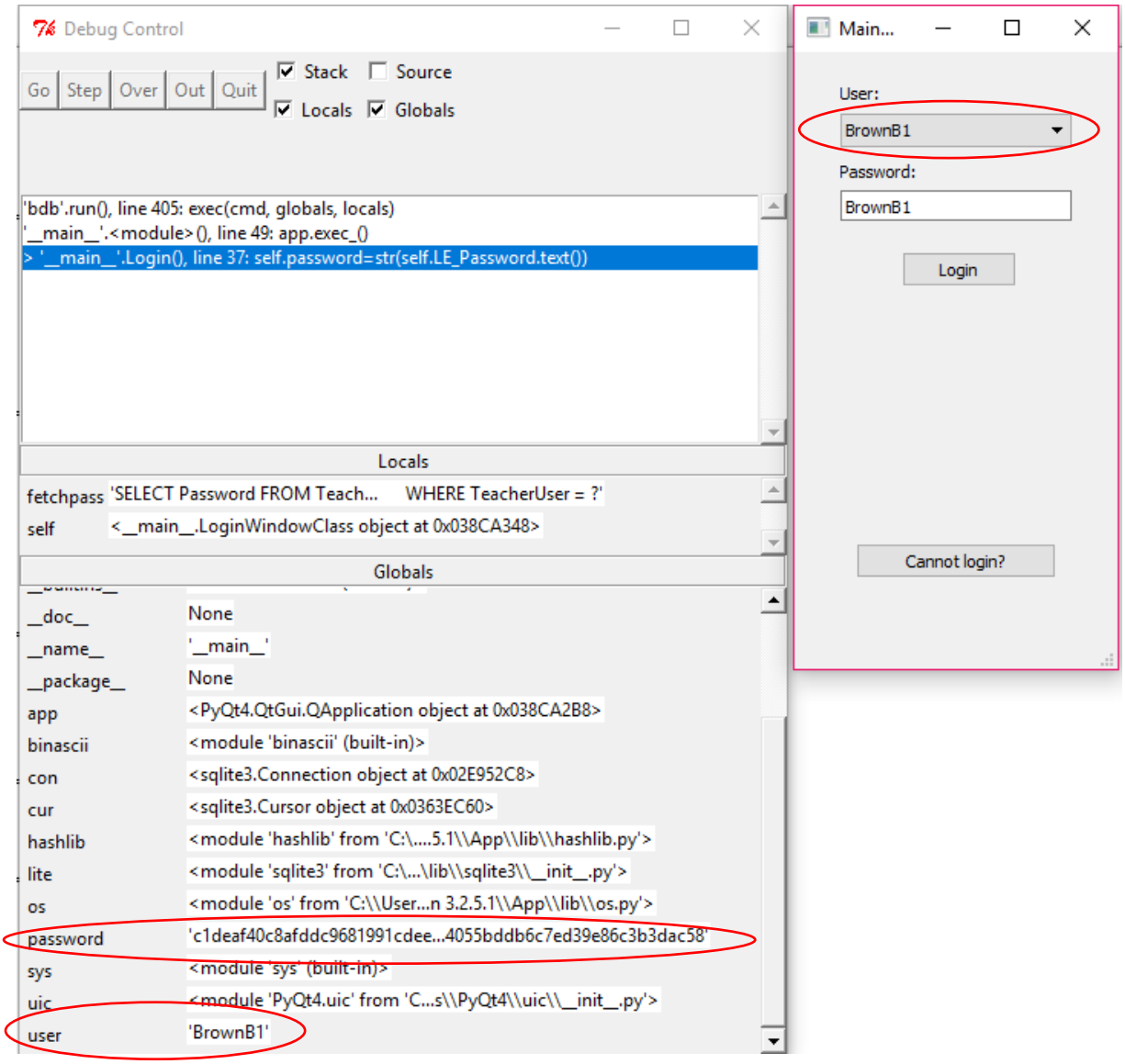
Although the login window was different back then, the code still created the GUI, and prompted the user to input their passwords. It also ran an SQL SELECT statement to retrieve the password of the user from the database.

If the user's credentials were valid, it would allow the user to login, and if not, it would report that the login was invalid. The user may then try to login again. However, it did not succeed in all of the tests provided by the test plan, therefore, I would proceed to amend this version.

Testing: login v1

The below screenshots will show evidence of tests being carried out, with their exact outcomes. I will show both the program output in the shell, the debugger and the GUI. After presenting evidence of the tests, I will summarise the results in a table and then show how I amended the code accordingly. Please note, output in these versions is not final, it only serves to help testing.

Attempting login with various credentials



The variable watch here clearly shows that the password is stored as a hash, and not in a raw form, thus successfully proving objective 8 which states the program shall do this. The program also successfully displaying the login screen serves to prove objective 3's success. By observing the database, it may also be concluded that objective 4 had been successfully achieved in this iteration of the program. Refer to screenshots below.

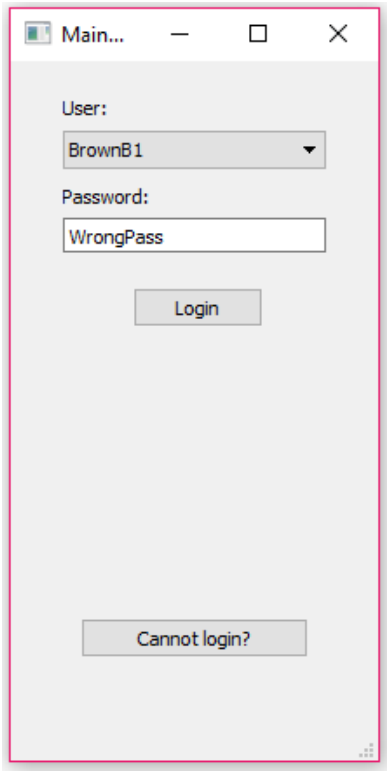
```
fetchpass=("""SELECT Password FROM Teachers
WHERE TeacherUser = ?""")
cur.execute(fetchpass, (user,))
password=cur.fetchone()[0]
```

TeacherID	SecondName	FirstName	TeacherUser	Administrator	Password	Email	FormID
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Brown	Bob	BrownB1	YES	c1deaf40c8af...	brownbob@m...	1
2	Smith	Lisa	SmithL2	YES	d4735e3a265...	smith88@mail...	2
3	Allen	Tim	AllenT3	NO	06ba0e411cfc...	allenallen@m...	3

The output produced by this test run may be seen here.

```
>>> ================================ RESTART ================================
[DEBUG ON]
>>>
BrownB1
c1deaf40c8afddc9681991cdee4d131f475ea4055bddb6c7ed39e86c3b3dac58
BrownB1
window will open here
```

"Window will open here" has been displayed to mark the success of the user's login attempt. Therefore, objective 2 has almost been achieved. All that must be done now is to attempt an invalid login to ensure the program will only allow access to a valid login attempt. I will attempt the next tests with an invalid password for "BrownB1", as this may simultaneously serve to test objective 6. Objective 5 cannot be tested in this version, due to the use of the combo box.




```
>>> ================================ RESTART ================================
[DEBUG ON]
>>>
BrownB1
c1deaf40c8afddc9681991cdee4d131f475ea4055bddb6c7ed39e86c3b3dac58
WrongPass
Password does not match with user.
```

"Password does not match with user" shows that the program can correctly identify when the login credentials are correct or not. Therefore, objective 2 has been successfully achieved in this iteration of the program.

However, the program did not successfully identify that it was the password that was wrong, and thus although the login was denied, the objective has only been partially achieved, and therefore I was forced to make amendments to this version of the program.

Another thing to note is that objective 39, specifying READ statements will be used for user authentication, has been achieved here.

Objective	How it was tested	Test data used	Expected outcome	Actual outcome	Success? (yes, no, partial)
2.	Attempting login with "BrownB1"	See "BrownB1" account in existing test data.	Program determined that login was successful.	Program determined that login was successful.	Yes
3.	Opening the program and observing.	n/a	Program successfully presented user with a login screen that allowed user inputs.	Program successfully presented user with a login screen that allowed user inputs.	Yes
4,39.	Variable watching and code analysis proved an SQL statement was successfully used to obtain a password for the user.	See "BrownB1" account in existing test data.	Program used an SQL SELECT query to obtain the password of BrownB1.	Program used an SQL SELECT query to obtain the password of BrownB1.	Yes
5.	Login attempted with a non-existent username.	FakeUser1, with password "BrownB1".	Program notified user that the account doesn't exist and doesn't allow login.	Program unable to achieve this as combo box does not allow for false inputs.	No
6.	Login attempted with a valid user, but an incorrect password.	"BrownB1" as username, and "WrongPass" as the password.	Program doesn't grant access, and specifically identifies the password was wrong.	Program didn't grant access, but didn't identify the password was wrong.	Partial
8.	Variable watching and raw view of database to confirm hashed passwords are used.	See "BrownB1" account in existing test data.	Variable watching will identify passwords as hashed in the database and program.	Variable watching will identify passwords as hashed in the database and program.	Yes

Login v2 code

After running the above tests, I identified that I would need to replace the combo box input for the username in the GUI window, as well as identify *which* user credential was incorrect in an invalid login attempt. This is how the next iteration of the login code appears. It is a function by itself, which may be used as its own module if necessary.

```
def TryLogin():
    global password #password = password of logged in user (grabbed from database)
    global username #username = logged in as permanently
    username=str(self.LE_User.text())
    try:
        fetchpass=("""SELECT Password FROM Teachers
        WHERE TeacherUser = ?""")
        cur.execute(fetchpass, (username, ))
        password=cur.fetchone() #Sets password variable to the chosen users
        password=password[0]
        print(password) #Prints unhashed password
        self.password=str(self.LE_Password.text()) #Saves user input password
        HashedUserInputPass=str(hashlib.sha256(self.password.encode()).hexdigest()) #Hashes user input
        print(HashedUserInputPass)
        if HashedUserInputPass == password:  #Checks user input to correct password
            print("Successful login")
            self.LE_User.setText("") #Removes uername from line edit
            self.LE_Password.setText("") #Removes username from line edit
            LoginWindow.hide()
            MenuWindow.show()
        else:
            print("Password does not match with user.")
            self.LE_Password.setText("") #Removes username from line edit
            QtGui.QMessageBox.information(self, "A login error has occured", "Password and username do not match.")
    except:
        self.LE_User.setText("") #Removes uername from line edit
        self.LE_Password.setText("") #Removes username from line edit
        QtGui.QMessageBox.information(self, "A login error has occured", "Username does not exist") #except for when SQL can't find username
TryLogin() #Call procedure again, resetting variables
```

I will show tests which I had previously carried out on this code, to ensure that it is able to identify which credentials are wrong. This code also uses pop-up windows, which is part of objective 23, meaning I can test this objective too.

In this version, I had also created a method within this class that is used when the “cannot login?” button is pressed. The purpose of this method is to return an administrative user’s email address to the user, so that they email them for assistance. I will therefore test this method after I have finished testing the login method. The “cannot login” method (PassForgot) may be seen here.

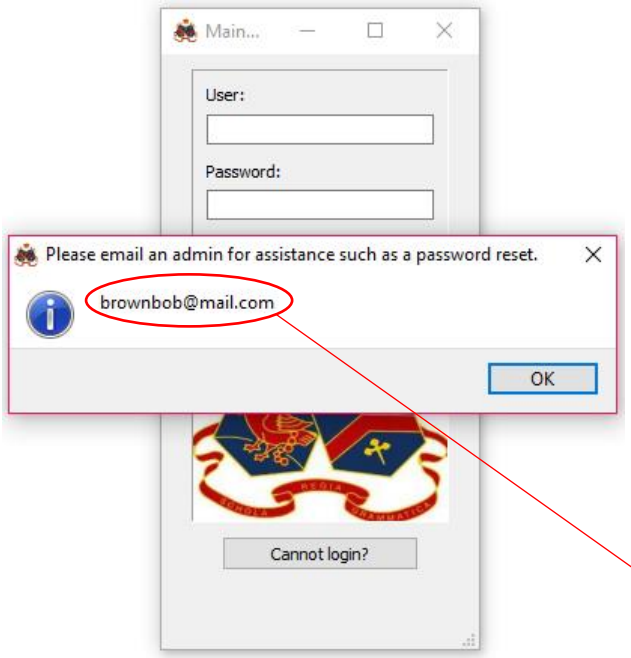
```
def PassForgot(self): #SQL grabs administrator email
    cur.execute("""SELECT Email FROM Teachers
    WHERE Administrator = 'YES'""")
    helpemail=cur.fetchone() #Fetches one admin email
    helpemail=str(helpemail[0])
    print(helpemail)
    QtGui.QMessageBox.information(self, "Please email an admin for assistance such as a password reset.",
    helpemail+"                                                                                    ")
```

Testing: login v2

I will not demonstrate the use of variable watching here, as it is not needed for any of the objectives that are going to be tested (5, 6, 12 and 23).

Clicking the "Cannot login?" button

In this test, the button was simply clicked, and the outcome was as recorded as follows.



As seen in the screenshot, a pop-up was created. This serves as evidence that objective 23, creating pop-ups to notify users of information, was achieved successfully.

Although an email has been displayed, it does not prove that this is the email of an administrator, only that an email has been produced. To prove that the output is valid, I will show the database, and how the email of an administrative user is being presented here.

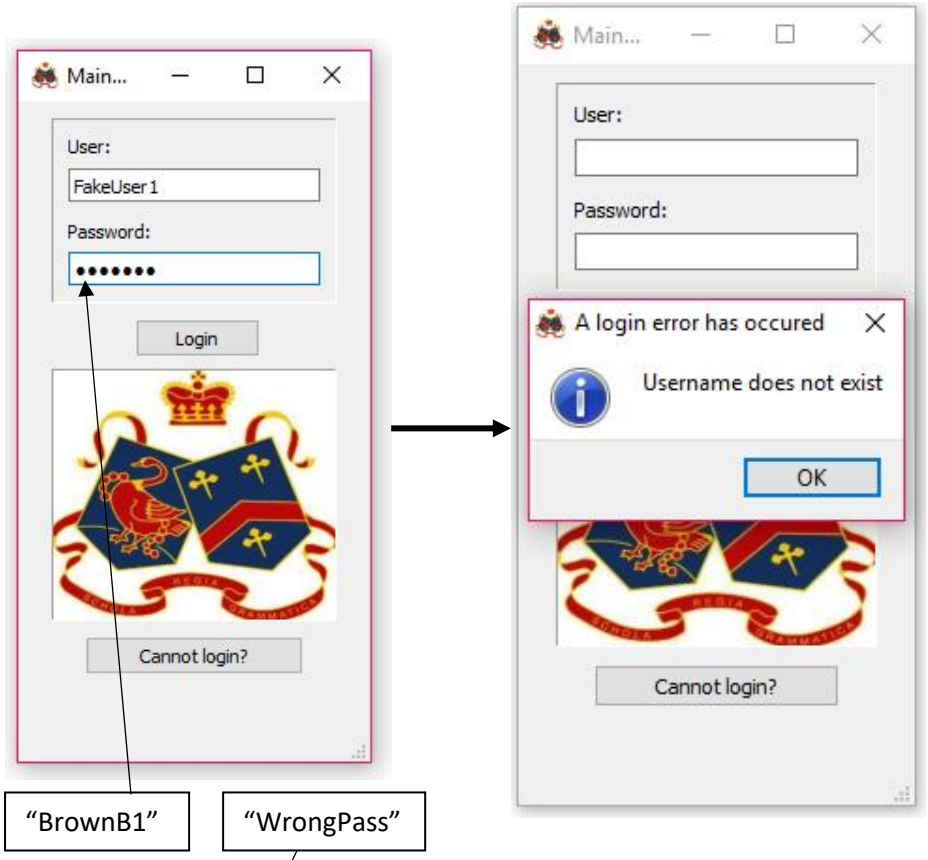
TeacherID	SecondName	FirstName	TeacherUser	Administrator	Password	Email	FormID
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Brown	Bob	BrownB1	YES	c1deaf40c8af...	brownbob@m...	1
2	Smith	Lisa	SmithL2	YES	d4735e3a265...	smith88@mail...	2
3	Allen	Tim	AllenT3	NO	06ba0e411cfc...	allenallen@m...	3

```
cur.execute("""SELECT Email FROM Teachers
WHERE Administrator = 'YES'""")
helpemail=cur.fetchone() #Fetches one admin email
helpemail=str(helpemail[0])
```

These screenshots should effectively serve as proof that the email of an administrator is being displayed here. Next, I will proceed to show the tests I carried out for the login method, as I did in Login v1.

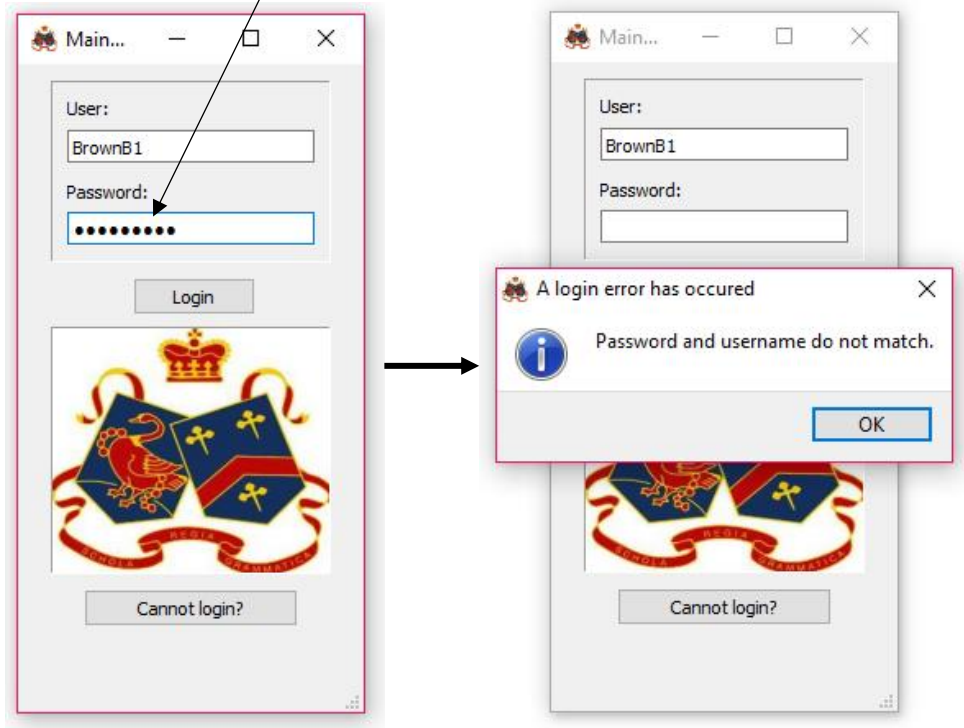
Attempting login with various credentials

One thing to note is that I have implemented hidden password inputs by replacing the characters with dots. I did this because it will increase the security of the program, as people will not be able to watch others input their details as easily. I will annotate what the password entered is however.



From these screenshots, it may be concluded that objective 5 has been achieved. This is due to how an invalid username was put in, and the program then identified this to the user using a pop-up window. Despite the password being a valid one, access was not granted.

The next test will use a valid username ("BrownB1") but an incorrect password (WrongPass).



This test shows how the program is able to identify if the password is incorrect, though the username is not. Also worth noting is how only the password input box is cleared in this case, as opposed to if the username is wrong then both credentials are cleared.

Objective	How it was tested	Test data used	Expected outcome	Actual outcome	Success? (yes, no, partial)
5.	Login attempted with a non-existent username.	FakeUser1	Program notified user that the account doesn't exist and doesn't allow login.	Program notified user that the account doesn't exist and didn't allow login.	Yes
6.	Login attempted with a valid user, but an incorrect password.	"BrownB1" as username, and "WrongPass" as the password.	Program doesn't grant access, and specifically identifies the password was wrong.	Program didn't grant access, and specifically identifies the password was wrong.	Yes
12.	The "cannot login?" button was pressed in the login window.	BrownB1 in "test users" table. Email: "BrownB@-mail.com"	Pop-up containing the email of an administrator popped up on screen.	Pop-up containing the email of an administrator popped up on screen.	Yes
23.	Use of program	n/a	Pop-up windows created, giving users information about the program.	Pop-up windows created, giving users information about the program.	Yes

As seen here, the objectives related to the login screen have all been achieved. Therefore, I may move on to show development of the menu window.

Menu window

The menu window was my next step in the iterative development process, as it was the first window that would be accessed once the user successfully logged in. It will also be used to navigate to all the other windows, so for testing purposes, it will be easier to develop the other windows after this one is available.

Menu v1 code

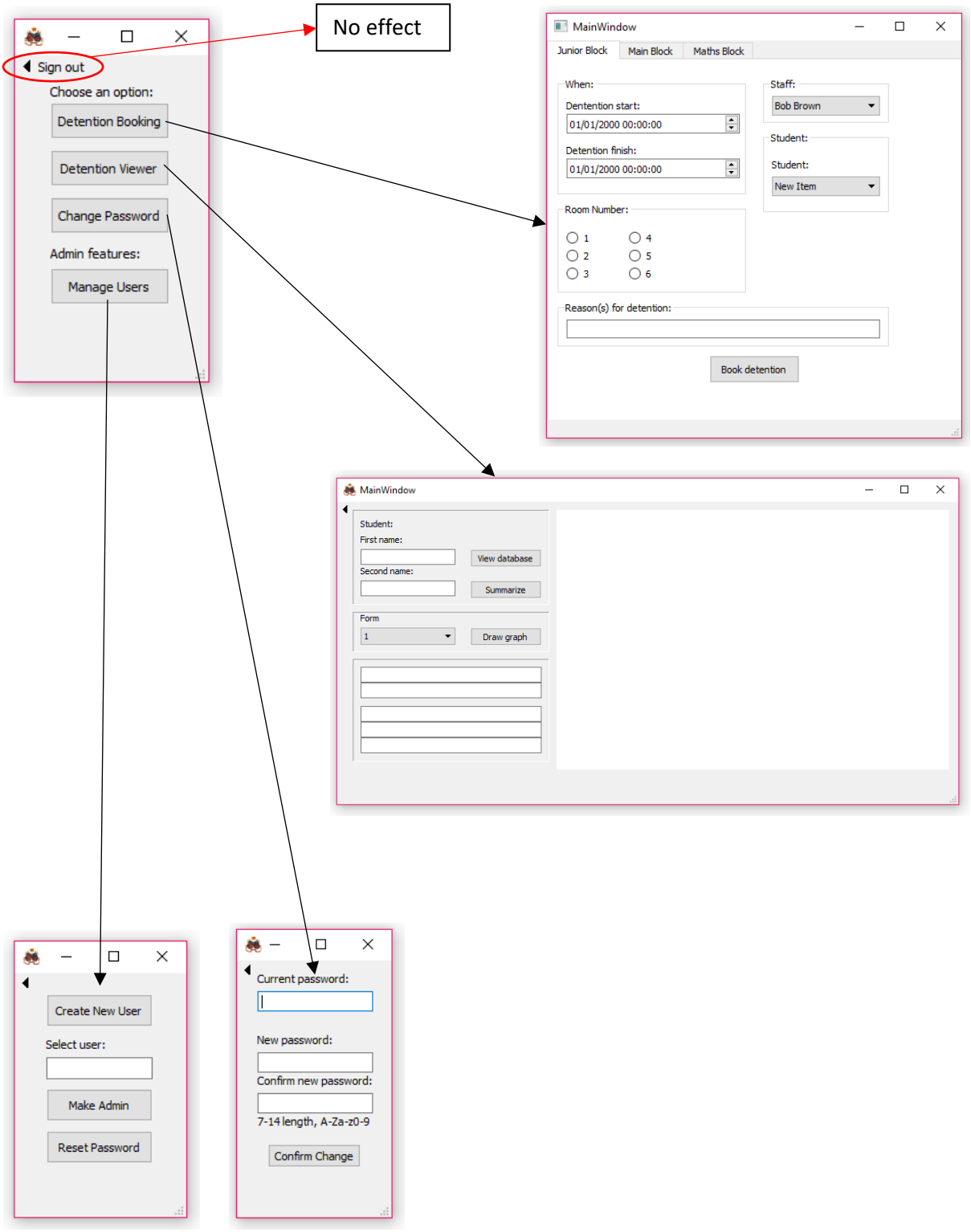
The following code shows the first iteration of my coding of the menu window. At this stage, I have simply made it so that the menu window will open when the user logs in, and its various buttons will open other windows. In this version, all of the buttons were successfully programmed, apart from the "sign out" button. This gets later addressed in version 2 of this code. Another missing feature in this version of the code includes how the "manage users" window can be opened by non-admins.

```
class MenuWindowClass(QtGui.QMainWindow, MenuWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        ###################
        self.BTN_DetentionPlanner.clicked.connect(self.DetentionPlanner)
        self.BTN_DetentionViewer.clicked.connect(self.DetentionViewer)
        self.BTN_ChangePass.clicked.connect(self.ChangePass)
        self.BTN_ManageUsers.clicked.connect(self.ManageUsers)
        ###################
    def DetentionPlanner(self):
        DetentionBookingWindow.show()
        MenuWindow.hide()
    def DetentionViewer(self): #Method opens window on click of button
        DetentionViewerWindow.show()
        MenuWindow.hide()
    def ChangePass(self):
        ChangePasswordWindow.show()
        MenuWindow.hide()
    def ManageUsers(self):
        ManageUsersWindow.show()
        MenuWindow.hide()

class DetentionBookingWindowClass(QtGui.QMainWindow, DetentionBookingWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        ###################
        #Define buttons here
        ###################

class ChangePasswordWindowClass(QtGui.QMainWindow, ChangePasswordWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        ###################
        #self.LE_CurrentPass.textChanged.connect(self.CurrentPass)
        ###################
    #def CurrentPass(self):

class ManageUsersWindowClass(QtGui.QMainWindow, ManageUsersWindow):
        self.setupUi(self)
        ###################
        #Define buttons here
        ###################

class DetentionViewerWindowClass(QtGui.QMainWindow, DetentionViewerWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        ###################
        self.TB_BACK.clicked.connect(self.BACK)
        ###################
    def BACK(self):
        DetentionViewerWindow.hide()
        MenuWindow.show()
```

Testing: menu v1

Testing buttons of menu window

Here I will demonstrate how the menu window will open different windows upon its various buttons being pressed. It can be seen that an old version of the detention booking window is in use.



As seen in the above tests, the “sign out” button has no effect whatsoever. This may determine the test as a partial failure, because the other 4 buttons serve the function of opening the other windows. Objective 31 states that only administrative users should be able to access the “manage users” window. However, if the method is reviewed, “def ManageUsers(self)”, it clearly carries out no check on whether the user is an admin or not.

Therefore, due to the tests not being a complete success, I made a second iteration of this code.

Menu v2 code

This code uses the above code; however, I have created a new method (“BACK”), and added to the original “ManageUsers” method. The intention was to use the “BACK” method to make the “sign out” button take the user back to the login screen, whilst resetting their credentials. The alterations to the “ManageUsers” method ensure that only administrators could access the manage users window.

```
def BACK(self):
    MenuWindow.hide()
    username = ""
    LoginWindow.show()
```

This first method is extremely short and simple, but serves the purpose of making the back-button work. One line is used to clear the username variable, so that the program can be reset effectively.

```
def ManageUsers(self):
    global usernameAdmin #use usernameAdmin to determine if admin or not
    CheckAdmin=("""SELECT Administrator FROM Teachers
    WHERE TeacherUser = ?""")
    cur.execute(CheckAdmin, (username, ))
    usernameAdmin=(cur.fetchone())
    usernameAdmin=usernameAdmin[0]
    print("Admin? "+usernameAdmin)
    if usernameAdmin == "YES":  #Checks if current user is admin and only grants access if they are
        ManageUsersWindow.show()
        MenuWindow.hide()
    else:
        QtGui.QMessageBox.critical(self, "Access denied:", username+" is not an administrative user")
```

This algorithm is more complex however. It immediately makes use of a global variable, “usernameAdmin”, so that other parts of the program, which need to know whether the user is an admin or not, will not have to run again if this has already been identified. The alternative would be to constantly use inheritance and passing this variable through various sub-routines. Using a global variable makes the code far easier to read.

Essentially, the code uses an SQL SELECT statement to identify if the current user is an admin or not (as recorded in the database). If the user is an admin, then it will grant access to the manage users window, and if not, a pop-up will notify the user that this window is only accessible to admins.

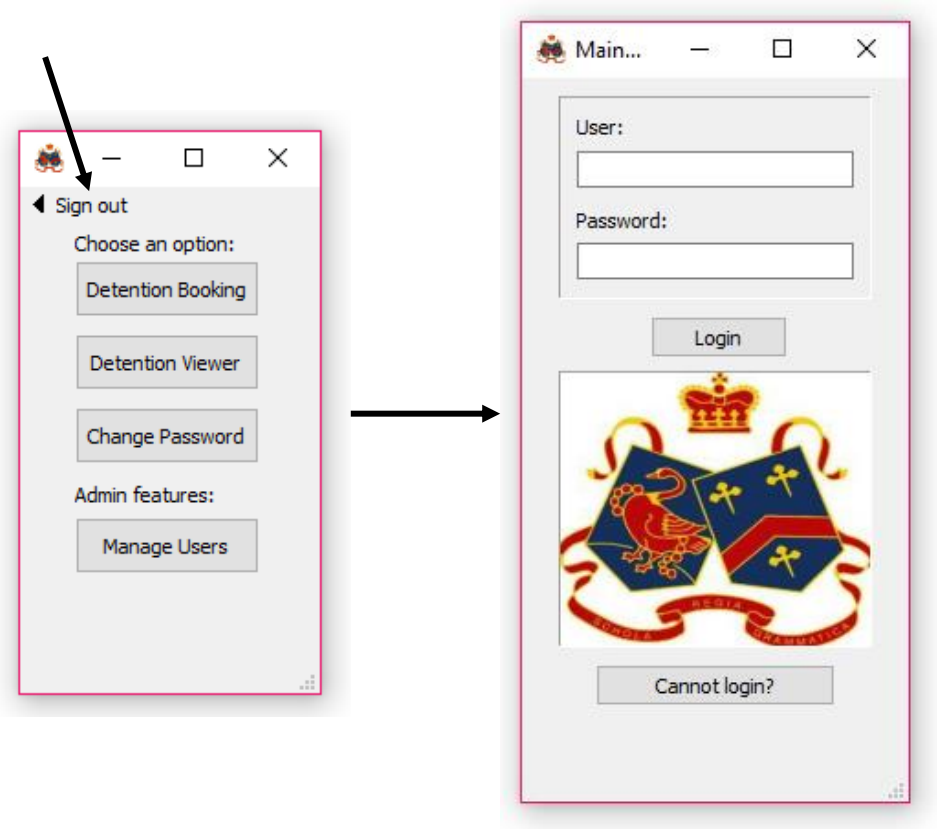
The changes to this code should be able to amend the issues that prevented the relevant success criteria from being achieved. As a result, I will not present the tests carried out that would identify if the following success criteria had been achieved: 7, 13, 22 and 31.

Testing: menu v2

As the buttons that take the users to the next window have already been tested, I did not test them again here, rather I will wait to test all these again during evaluation. The aspects that will be tested in this section are the back button and the manage users button.

Testing back button for menu window

This test will be extremely simple. I will simply show the outcome of the back button being pressed, as stated in the test plan found in the design section of this document. The aim of this test is to determine whether objective 22 has been achieved or not, as well as objective 7.

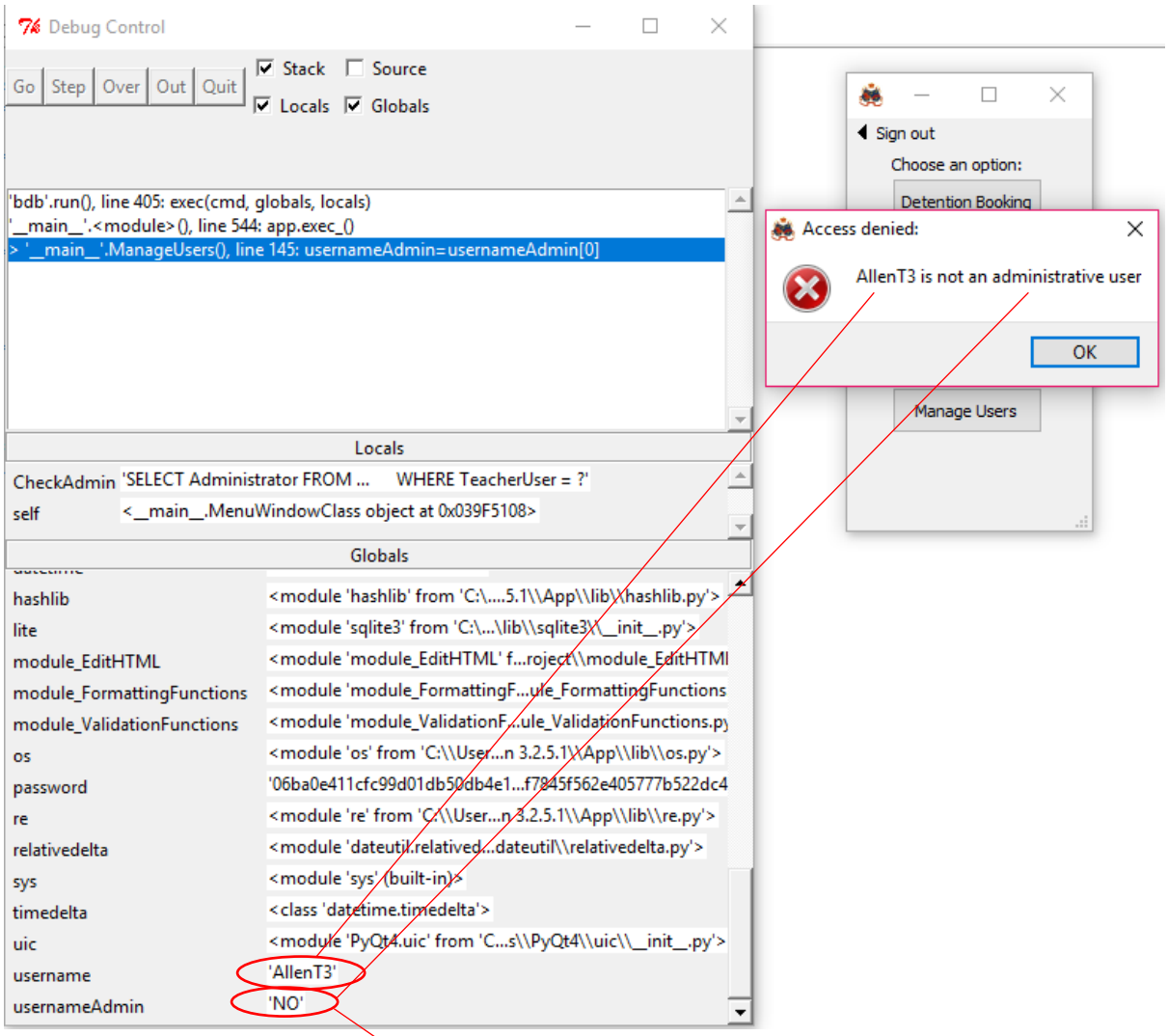


The above screenshots serve to show how the user successfully signed out by pressing the “sign out” button. Both showing how the back button, and more specifically, the sign out button is working as intended.

Testing “ManageUsers” method

As stated in objectives 13 and 31, administrative users should be able to access the manage users, while normal users should not be able to. The method of testing this will be to carry out 2 tests, one with a non-administrative user, and one with an administrative user. The test plan will be summarised after this test has been shown, identifying the test data that was used.

Non-administrative user



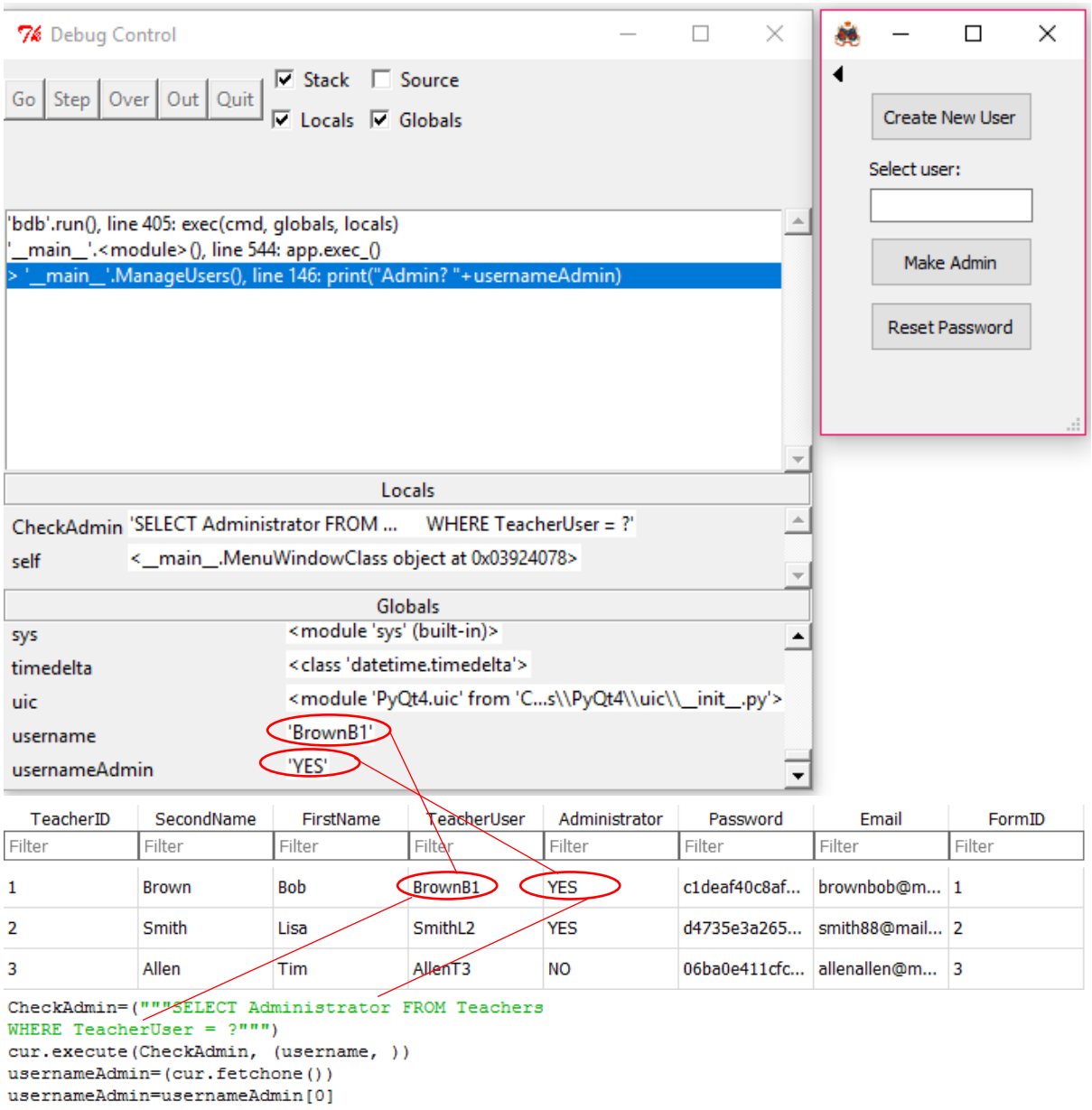
TeacherID	SecondName	FirstName	TeacherUser	Administrator	Password	Email	FormID
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Brown	Bob	BrownB1	YES	c1deaf40c8af...	brownbob@m...	1
2	Smith	Lisa	SmithL2	YES	d4735e3a265...	smith88@mail...	2
3	Allen	Tim	AllenT3	NO	06ba0e411cfc...	allenallen@m...	3

```
CheckAdmin=("""SELECT Administrator FROM Teachers
WHERE TeacherUser = ?""")
cur.execute(CheckAdmin, (username, ))
usernameAdmin=(cur.fetchone())
usernameAdmin=usernameAdmin[0]
```

As seen in the above screenshots, the program is able to identify if the user is not an admin, and thus reject their attempt to view the manage users window.

Administrative user

I will now carry out the same tests, but logged in with an administrative account.



As seen in the above screenshots, the manage user window was successfully opened. By looking at the variables within the debugger, it can be seen that the admin check would have proved successful due to the user being successfully identified as an administrator (usernameAdmin = 'YES').

Objective	How it was tested	Test data used	Expected outcome	Actual outcome	Success? (yes, no, partial)
7.	Sign out button pressed.	n/a, any username can be used for this.	Taken back into login screen, variables related to login credentials also reset.	Taken back into login screen, variables related to login credentials also reset.	Yes
13,31.	Test users “BrownB1” and “AllenT3” attempted to access the “manage users” window.	See test users table under “existing test data”. “BrownB1” and “AllenT3”.	Program will deny access to “AllenT3”, but allow “BrownB1”. Pop-up informs AllenT3 that they must be an admin.	Program denied access to “AllenT3”, but allowed “BrownB1”. Pop-up informs AllenT3 that they must be an admin.	Yes
22.	Use of program, specifically pressing back buttons.	n/a, any username can be used for this.	Users were able to navigate back to the previous window, from any other window once logged in.	Users were able to navigate back to the previous window, from any other window once logged in.	Yes

Due to all the objectives being achieved that relate to the menu window, I will now show the various iterations of development for the next window.

Change password window

Change password v1 code

The following code shows the first iteration of the change password window. In this version, I only implemented the code that would result in the program updating the password of the currently logged in user. However, there was no validation. As a result, any password would be accepted, despite enforcing of specific passwords being part of objectives 9 and 10. That being said, since functionality for objectives 11 and 42 had been addressed at this stage (being able to change the user's password), I will show the relevant tests I carried out for it.

```
class ChangePasswordWindowClass(QtGui.QMainWindow, ChangePasswordWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        ##################
        #self.LE_CurrentPass.textChanged.connect(self.CurrentPass)
        self.BTN_ConfirmChange.clicked.connect(self.ChangePass)
        ##################
    def ChangePass(self): #There is no validation in this version
        self.CurrentPass=str(self.LE_CurrentPass.text())
        self.NewPass=str(self.LE_NewPass.text())
        self.ConfirmNewPass=str(self.LE_ConfirmNewPass.text()) #Takes 3 inputs
        if self.CurrentPass != password[0]:                 #Checks if current password is correct
            QtGui.QMessageBox.information(self, "Error changing password:", "The current password is incorrect.")
        else:
            if self.NewPass == self.ConfirmNewPass: #checks if new passwords match
                UpdatedPass=self.NewPass
                print("username:"+username)
                print("New password:"+UpdatedPass)
                UpdatePass=("""UPDATE Teachers
                Set Password = ?
                WHERE TeacherUser = ?""")  #Query sets new password to relevant user
                cur.execute(UpdatePass, (UpdatedPass, username, ))
                QtGui.QMessageBox.information(self, "Password change successful:", username+"'s password has been updated.")
                FetchPassAgain=("""SELECT Password FROM Teachers
                WHERE TeacherUser = ?""") #Query grabs password to check if updated
                cur.execute(FetchPassAgain, (username, ))
                password2=cur.fetchone()
                password2=str(password2)
                print("Current password:"+password2)
                ChangePasswordWindow.hide()
                MenuWindow.show()    #Note: password changes do not appear to be saved. SQL reverts automatically?
            else:                    # ^ this was due to missing con.commit()
                QtGui.QMessageBox.information(self, "Error changing password:", "The new passwords do not match.")
```

The "ChangePass" method is the only method that was created in this iteration of the code. The second iteration would also add another method which makes the back-button work.

As shown in the pseudocode that can be found in the design section of this document, this algorithm will check if the current password is correct, and if the two new passwords are identical. If they are, then the program will carry out an SQL UPDATE statement, changing the password of the user to the new password. The SQL query can be seen more clearly here.

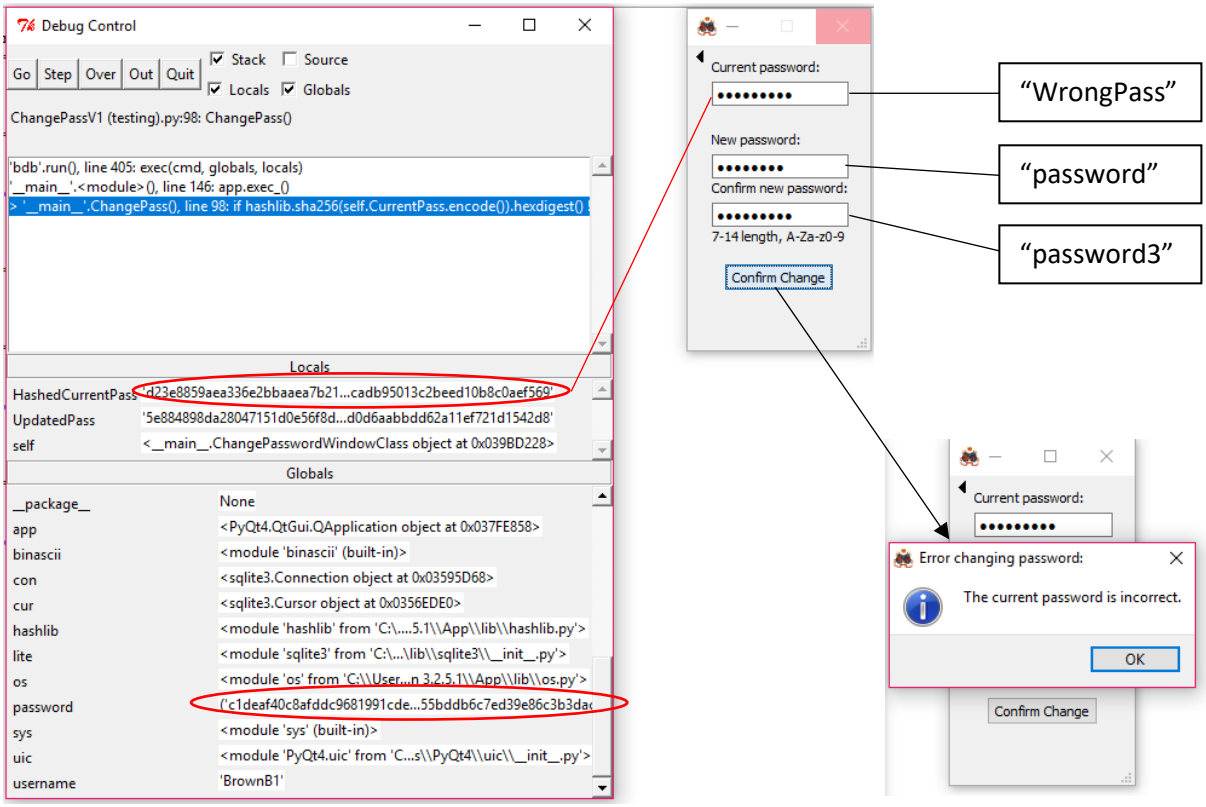
```
UpdatePass=("""UPDATE Teachers
Set Password = ?
WHERE TeacherUser = ?""")  #Query sets new password to relevant user
cur.execute(UpdatePass, (UpdatedPass, username, ))
```

The question marks in this query are used to pass variables into the query. As seen on the last line in this segment, "UpdatePass" shows the query, and "UpdatedPass" and "username" are the variables that are being passed into the query.

Testing: change password v1

As shown in the above code, the functionality for updating the user's password has been created. Since this the requirements of objectives 11 and 42, I will show the tests that evidence whether it was successful or not.

Testing password change with wrong "current password"

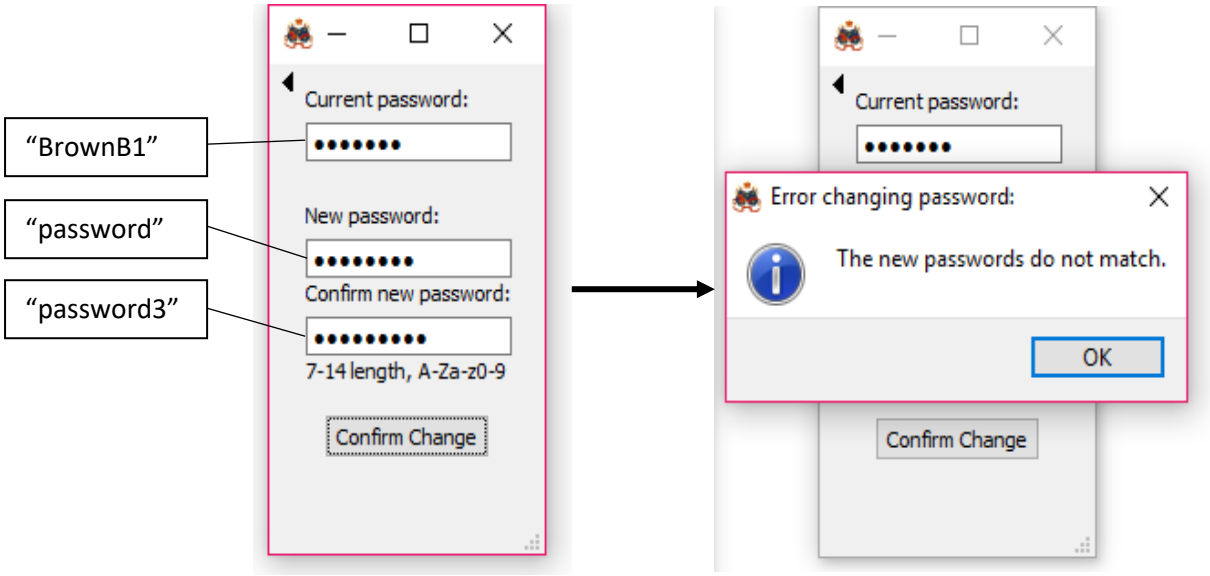


The above screenshots show the output of the program when those specific inputs were used for passwords. This demonstrates that the program is able to clarify whether or not the current password is correct, as it compares it with the global variable, "password". If they were to match, then "password" and "HashedCurrentPass" would be equal.

Line of code responsible for determining if the "current password" and the actual password are the same may be seen here.

```
if self.CurrentPass != password[0]:                    #Checks if current password is correct
```

Testing password change when "confirm new password" is incorrect

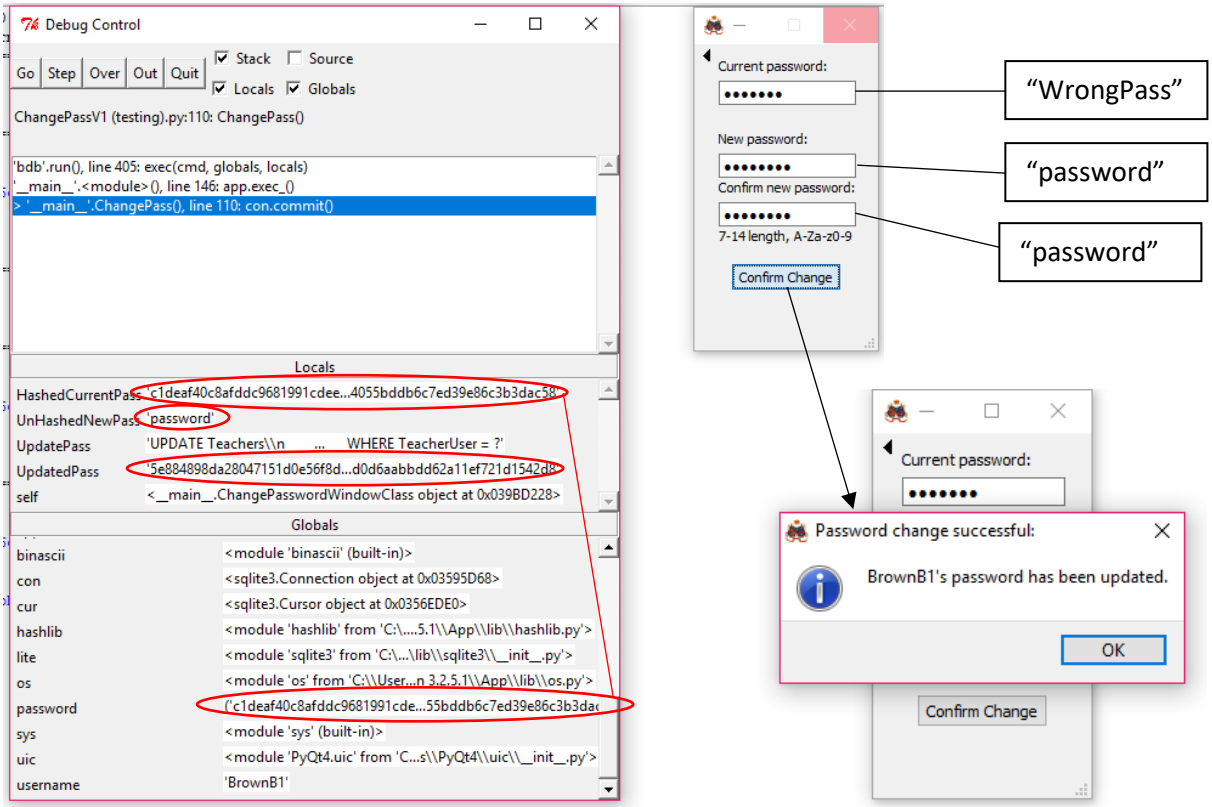


As evidenced in the above screenshots, the program can successfully identify when the two new passwords are not identical. This is done by the following line of code.

```
if self.NewPass == self.ConfirmNewPass: #checks if new passwords match
```

The simple if statement determines whether the passwords match or not.

Testing password change when all passwords are correct



Although the above screenshots show how the program is able to identify that all of the input passwords are valid, these alone don't prove that the database is updated with the new password. Therefore, I will now show the before and after screenshots of the database.

Before the password was changed

TeacherID	SecondName	FirstName	TeacherUser	Administrator	Password	Email	FormID
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Brown	Bob	BrownB1	YES	c1deaf40c8af...	brownbob@m...	1
2	Smith	Lisa	SmithL2	YES	d4735e3a265...	smith88@mail...	2
3	Allen	Tim	AllenT3	NO	06ba0e411cfc...	allenallen@m...	3

As seen the password here begins "c1deaf40…". This is the hash of "BrownB1"

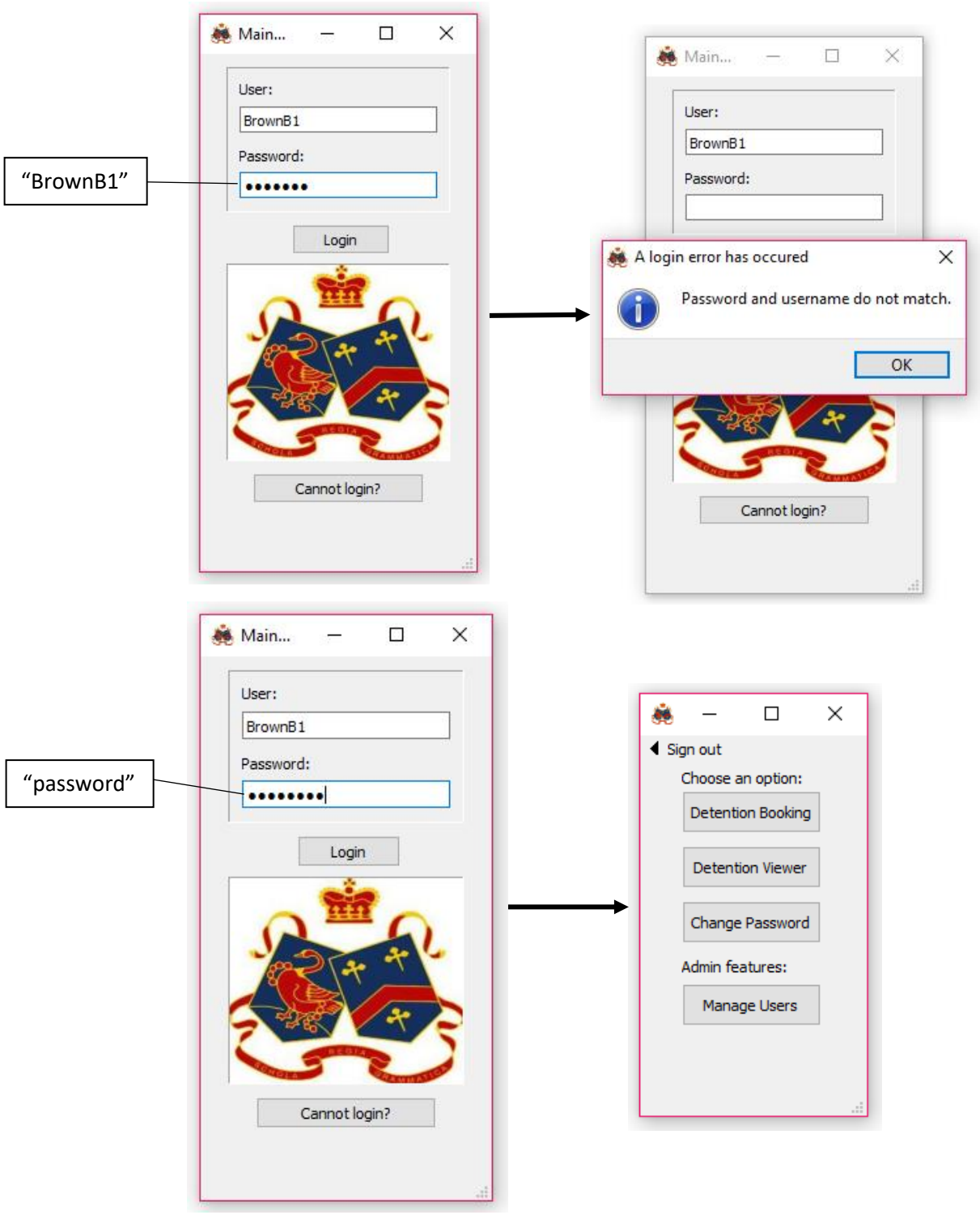
After the password was changed

TeacherID	SecondName	FirstName	TeacherUser	Administrator	Password	Email	FormID
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Brown	Bob	BrownB1	YES	5e884898da2...	brownbob@m...	1
2	Smith	Lisa	SmithL2	YES	d4735e3a265...	smith88@mail...	2
3	Allen	Tim	AllenT3	NO	06ba0e411cfc...	allenallen@m...	3

The password here, however, beings "5e88589…" which is different. This is the hash of "password". To ensure these tests are valid, I then attempted to login with both the old and new passwords.

Attempting logins after a successful password change

In order to prove the password was successfully changed, I will show how the login was unsuccessful when using the previous password, and successful when using the new password.



It may very easily be concluded from these screenshots that the password update was successful, thus proving that this iteration of the program had achieved objectives 11 and 42.

Objective	How it was tested	Test data used	Expected outcome	Actual outcome	Success? (yes, no, partial)
9,10.	Test data used to attempt creating various passwords.	See password examples under “input test data”.	“Password123” to be the only accepted passwords. All others will create a pop-up, notifying user of incorrect password.	No validation for passwords was used, and therefore any passwords were accepted.	No
11,42.	Change password feature was used.	“password” with user “BrownB1”.	Another login attempt using the new password was carried out to confirm If the password was changed.	The next login attempt only accepted the updated version of the password, proving it had changed.	Yes

As this table shows that I had not achieved objectives 9 and 10 with this iteration of the program, I would develop the code and re-test it.

NOTE: For testing purposes, I changed the password back to “BrownB1” after these tests had been carried out.

Regular expressions for password validation

I made an external function with no link to the main program, for the purpose of testing regular expressions. Once I had tested this function to work successfully, I would attempt to integrate it into the whole program. The reason I chose this approach, was due to how I had never used regular expressions before, so I had to learn before I could attempt to make it work. I would make it a function on its own so that I could import it as a separate module. This would improve readability of the code, therefore making it easier to follow what the program is doing.

For each iteration of my validating passwords function, I will test it using the "password examples" that I stated under my "input test data" within the design section of this document. For the validation to work as intended, only "Password123" should be accepted, and the user should be notified with at least one specific problem that their input password has, unless there are none.

The test data specified above may be seen here:

Password example	Intended error of password
"hello"	No capitals or digits. Insufficient length.
"password123"	No capitals
"Password123"	No errors
"Password"	No digits
"Pass12"	Insufficient length
"Pass$word123"	Special characters used
"Pass%1"	Special characters used, insufficient length
"Password123Password"	Too long

Regular expressions v1 code

```
import re

def TWOstepREGEX():
    TestString = input("password 7-14 length, 1 capital, 1 number: ")
    regex = r"^(\w{7,14})$" #7-14 length, no special characters
    CaptureGroup = re.findall(regex, TestString)
    if CaptureGroup == []:
        print("password invalid: does not satisfy length \n")
        TWOstepREGEX()
    else:
        CaptureGroup = CaptureGroup[0]
        print("password valid: satisfies length")
    regex2 = r"^([^0-9]|[^A-Z]*)$" #atleast 1 capital and 1 number
    CaptureGroup2 = re.findall(regex2, CaptureGroup)
    if CaptureGroup2 == []:
        print("password valid: satisfies regex \n")
        for i in CaptureGroup2:
            print(CaptureGroup2[i])
        TWOstepREGEX()
    else:
        CaptureGroup2 = CaptureGroup2[0]
        print("password invalid: does not satisfy regex \n")
        TWOstepREGEX()

TWOstepREGEX()
```

The above code was my first attempt at creating a function which would enforce the password validation. As stated in my objectives, the password must be 7-14 in length, contain no special characters and contain both at least 1 capital letter and 1 digit. There is very little to comment about the code since it is very self-explanatory. The code is essentially checking for a capture group with the regular expressions, and whether there are capture groups or not determines if the string is valid or not.

In this iteration of the code, the problem present is that it both doesn't work for all types of invalid passwords, and it doesn't specify what the issue with the password always is, despite it being one of my objectives.

Testing: Regular expressions v1

I will now present how this function handled the various password inputs.

```
password 7-14 length, 1 capital, 1 number: hello
password invalid: does not satisfy length

password 7-14 length, 1 capital, 1 number: password123
password valid: satisfies length
password invalid: does not satisfy regex

password 7-14 length, 1 capital, 1 number: Password123
password valid: satisfies length
password valid: satisfies regex

password 7-14 length, 1 capital, 1 number: Password
password valid: satisfies length
password valid: satisfies regex

password 7-14 length, 1 capital, 1 number: Pass123
password valid: satisfies length
password valid: satisfies regex

password 7-14 length, 1 capital, 1 number: Pass$word123
password invalid: does not satisfy length

password 7-14 length, 1 capital, 1 number: Pass%1
password invalid: does not satisfy length

password 7-14 length, 1 capital, 1 number: Password123Password
password invalid: does not satisfy length
```

Incorrect validation results can be seen for "Password" and "Pass$word123". This is because it should have rejected "Password" on the grounds of it not containing any digits, and it should have rejected "Pass$word123" on the grounds that it contained a special character. However, it rejected it because it was an invalid length, despite it being a fine length. It also didn't differentiate between errors of missing a capital letter or missing a digit.

I therefore concluded that this iteration was a failure, and so I proceeded to create another version.

Regular expressions v2 code

After identifying the problem with the last code, which was that it could not successfully differentiate between all different types of errors, I created this iteration.

However, as the tests will identify, I still hadn't addressed the problem of identifying whether special characters were present or not. One thing to note about this solution, was at this stage I had identified that QtDesigner's line edits would allow validation of their own. As a result, I could limit the length of characters being put in, making the idea of ensuring it was less than 14 in length redundant.

```
import re

TestString = input("Password 7-14 length, 1 cap, 1 digit: ")

LengthRegex = r"(\w{7,14})$"
C1 = re.findall(LengthRegex, TestString)

if len(C1) < 1:
    print("String length invalid.")
else:
    pass

OneDigitRegex = r"\d"
C2 = re.findall(OneDigitRegex, TestString)
#print("\n")
print(C2)

if len(C2) < 1:
    print("No digit present.")
else:
    pass

OneCapRegex = r"[A-Z]"
C3 = re.findall(OneCapRegex, TestString)
#print("\n")
print(C3)

if len(C3) < 1:
    print("No caps present.")
else:
    pass
```

Testing: Regular expressions v2

Carrying out the same test as shown before, the program gave the following outputs.

```
Password 7-14 length, 1 cap, 1 digit: hello
String length invalid.
[]
No digit present.
[]
No caps present.

Password 7-14 length, 1 cap, 1 digit: password123
['1', '2', '3']
[]
No caps present.

Password 7-14 length, 1 cap, 1 digit: Password123
['1', '2', '3']
['P']

Password 7-14 length, 1 cap, 1 digit: Password
[]
No digit present.
['P']

Password 7-14 length, 1 cap, 1 digit: Pass12
String length invalid.
['1', '2']
['P']

Password 7-14 length, 1 cap, 1 digit: Pass$word123
['1', '2', '3']
['P']

Password 7-14 length, 1 cap, 1 digit: Pass%1
String length invalid.
['1']
['P']

Password 7-14 length, 1 cap, 1 digit: Password123Password
['1', '2', '3']
['P', 'P']
```

As seen above, it was able to determine whether a capital was missing, or whether a digit was missing. It could even determine both at the same time. However, it was still not identifying whether there were special characters present or not, as seen by it accepting “Pass$word123”.

Because of this test, I knew where I could improve it, to get it in a position where I could implement it into the main program, as its own module.

Regular expressions v3 code

```
import re

def Regex():
    print("")
    TestString = input("Password 7-14 length, 1 cap, 1 digit: ")
    if len(re.findall(r"\W+", TestString)) > 0:
        print("no special characters.")
        Regex()
    else:
        if len(re.findall(r"(\w{7,14})$", TestString)) < 1:
            print("invalid length.")
            Regex()
        else:
            if len(re.findall(r"\d", TestString)) < 1:
                print("no digits.")
                Regex()
            else:
                if len(re.findall(r"[A-Z]", TestString)) < 1:
                    print("no caps.")
                    Regex()
                else:
                    print("PASSWORD VALID.")
                    Regex()

Regex()
```

The first line of the above code shows how special characters will be rejected. It searches for any special characters, and if any returned capture groups exist, it will notify the user that the password is invalid, and the reason it is invalid.

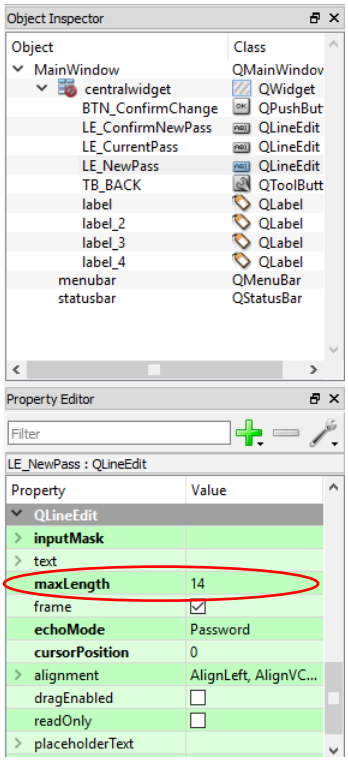
Due to how this function will call itself again as soon as an error in the password has been determined, it will not identify multiple different errors in one go. I do not see this as a problem however, and identifying multiple errors at once was not in my objectives so if it tests successfully, I will still consider it successful.

Testing: Regular expressions v3

The following screenshot shows how the code responded to the various password inputs.

```
Password 7-14 length, 1 cap, 1 digit: hello
invalid length.

Password 7-14 length, 1 cap, 1 digit: password123
no caps.

Password 7-14 length, 1 cap, 1 digit: Password123
PASSWORD VALID.

Password 7-14 length, 1 cap, 1 digit: Password
no digits.

Password 7-14 length, 1 cap, 1 digit: Pass12
invalid length.

Password 7-14 length, 1 cap, 1 digit: Pass$word123
no special characters.

Password 7-14 length, 1 cap, 1 digit: Pass%1
no special characters.

Password 7-14 length, 1 cap, 1 digit: Password123Password
PASSWORD VALID.
```

As seen, the only error in this function is that “Password123Password” is accepted, despite it being too long. I have already addressed this issue, as I found out how to enforce input lengths for line edits within QtDesigner. Therefore, it is a redundant issue in this code, as the function will never receive strings that are of greater length than 14.



This screenshot shows where I have specified the maximum input size of a string for the line edit in the GUI.

Change password (final version) code

This code builds upon the “change password v1” code, using the same working functionality that uses an SQL UPDATE statement to change the password of the user if the passwords are valid. It also uses the same validation checks to ensure that the current password is correct, and the new passwords are matching. As these have already been tested successfully (objectives 11 and 42), I will not be testing them in the same way again.

```
    def ChangePass(self):
        self.CurrentPass=str(self.LE_CurrentPass.text())
        HashCurrentPass=str(hashlib.sha256(self.CurrentPass.encode()).hexdigest())
        self.NewPass=str(self.LE_NewPass.text())
        self.ConfirmNewPass=str(self.LE_ConfirmNewPass.text()) #Takes 3 inputs
        if HashCurrentPass != password:                #Checks if current password is correct
            QtGui.QMessageBox.information(self, "Error changing password:", "The current password is incorrect.")
        else:
            if self.NewPass == self.ConfirmNewPass: #Checks if new passwords match
                #print(PasswordRegex(self.NewPass))
                PasswordRegexCheck = module_ValidationFunctions.PasswordRegex(self.NewPass)
                if PasswordRegexCheck == "VALID":#See PasswordRegex function within "ValidationFunctions" module
                    UpdatedPass=self.NewPass
                    HashedUpdatedPass=str(hashlib.sha256(UpdatedPass.encode()).hexdigest())
                    print("username:"+username)
                    print("New password:"+UpdatedPass)
                    print("Hashed = "+HashedUpdatedPass)
                    UpdatePass=("""UPDATE Teachers
                    SET Password = ?
                    WHERE TeacherUser = ?""")  #Query sets new password to relevant user
                    cur.execute(UpdatePass, (HashedUpdatedPass, username, ))
                    con.commit()
                    QtGui.QMessageBox.information(self, "Password change successful:", username+"'s password has been updated.")
                    FetchPassAgain=("""SELECT Password FROM Teachers
                    WHERE TeacherUser = ?""") #Query grabs password to check if updated
                    cur.execute(FetchPassAgain, (username, ))
                    password2=cur.fetchone()
                    password2=password2[0]
                    password2=str(password2)
                    print("Current password:"+password2)
                    ChangePasswordWindow.hide()
                    MenuWindow.show()
                else:
                    QtGui.QMessageBox.information(self, "Error changing password: ",PasswordRegexCheck)
            else:  #If passwords don't match, error message created
                QtGui.QMessageBox.information(self, "Error changing password:", "The new passwords do not match.")
```

The above code shows what the final version of the “ChangePass” method looks like. As seen it uses a module called “ValidationFunctions”. This module contains a function called “PasswordRegex”, hence why the function is called by “ValidationFunctions.PasswordRegex(self.NewPass)”.

This function returns either “VALID”, if there are no issues with the password, or it returns what the error with the new password. This error can then be used in the 3rd from bottom line, to create a pop-up which notifies the user of the issue. The PasswordRegex function can be seen here.

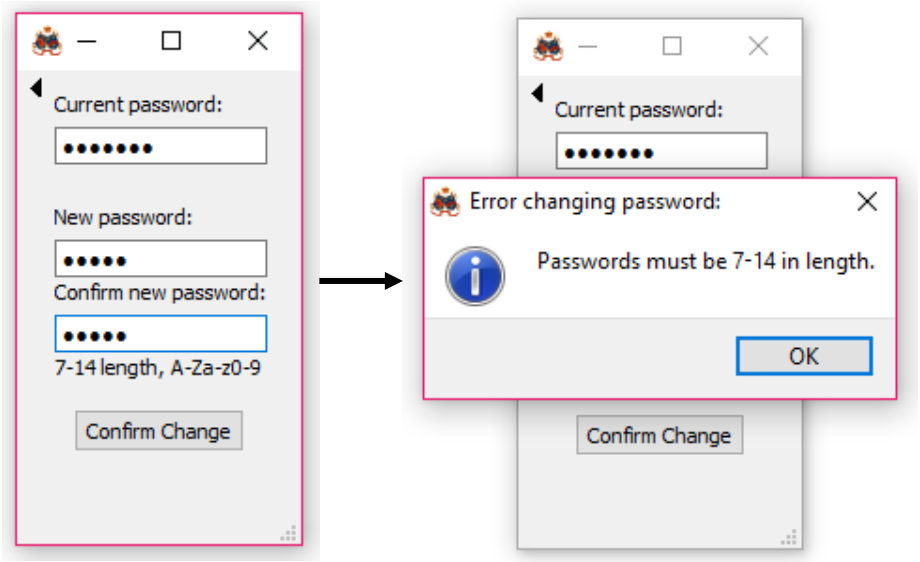
```
import re

def PasswordRegex(str_Password): #for each statement, checks if any capture groups are returned
    if len(re.findall(r"\W+", str_Password)) > 0: #password contain a special character?
        str_Error = "Passwords must not contain any special characters."
    else:
        if len(re.findall(r"(\w{7,14})$", str_Password)) < 1: #password not 7-14 in length?
            str_Error = "Passwords must be 7-14 in length."
        else:
            if len(re.findall(r"\d", str_Password)) < 1: #password not contain a digit?
                str_Error = "Passwords must contain a digit."
            else:
                if len(re.findall(r"[A-Z]", str_Password)) < 1: #password not contain a capital?
                    str_Error = "Passwords must contain a capital."
                else:
                    print("PASSWORD VALID.")
                    return "VALID"
    return str_Error
```

“Str_Error” is a variable that is used to show the user what the error with the password was.

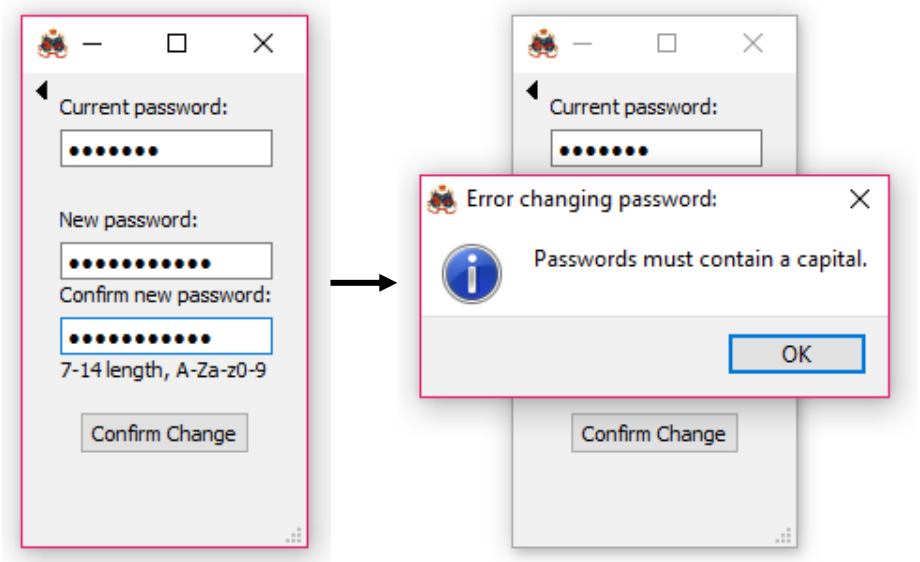
Testing: Regular expressions (final version)

Due to the “change password v1” code showing that the program can successfully update the passwords, I will only be carrying out these tests to show how the regex function enforces valid passwords, and not showing again that they are updated to the database. Therefore, I will only need to use screenshots to prove how the program reacts to the various passwords.

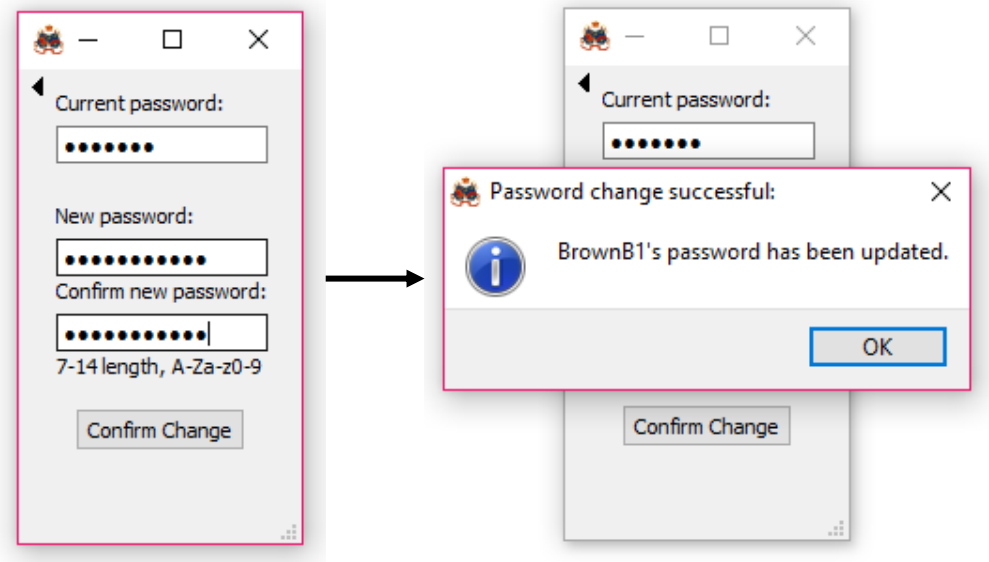
Password change: “hello”



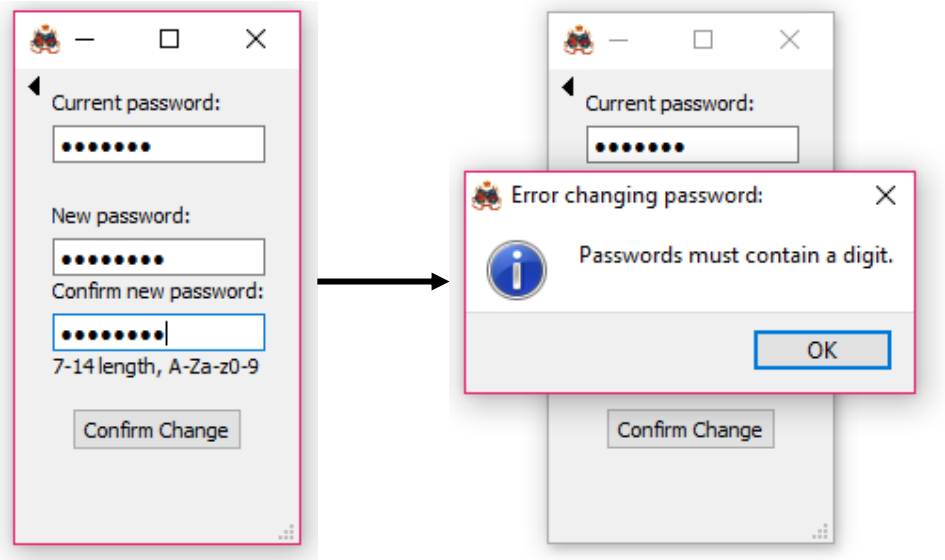
Password change: “password123”



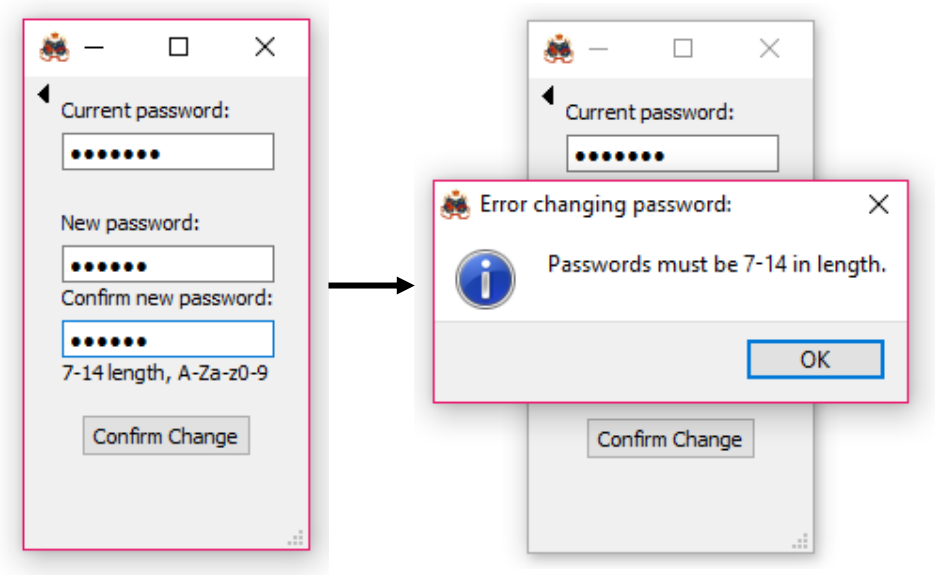
Password change: “Password123”



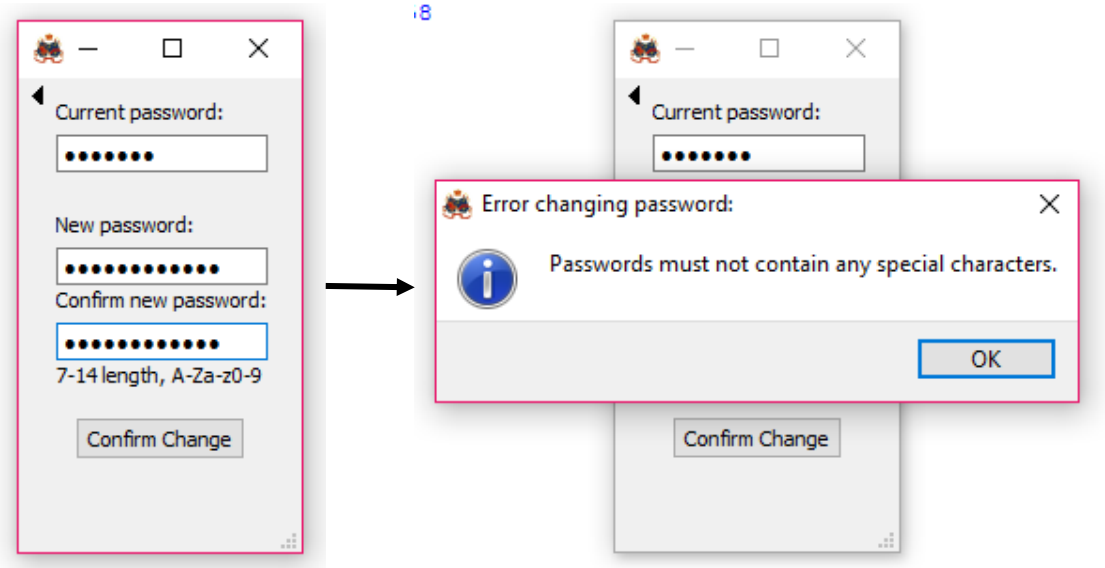
Password change: “Password”



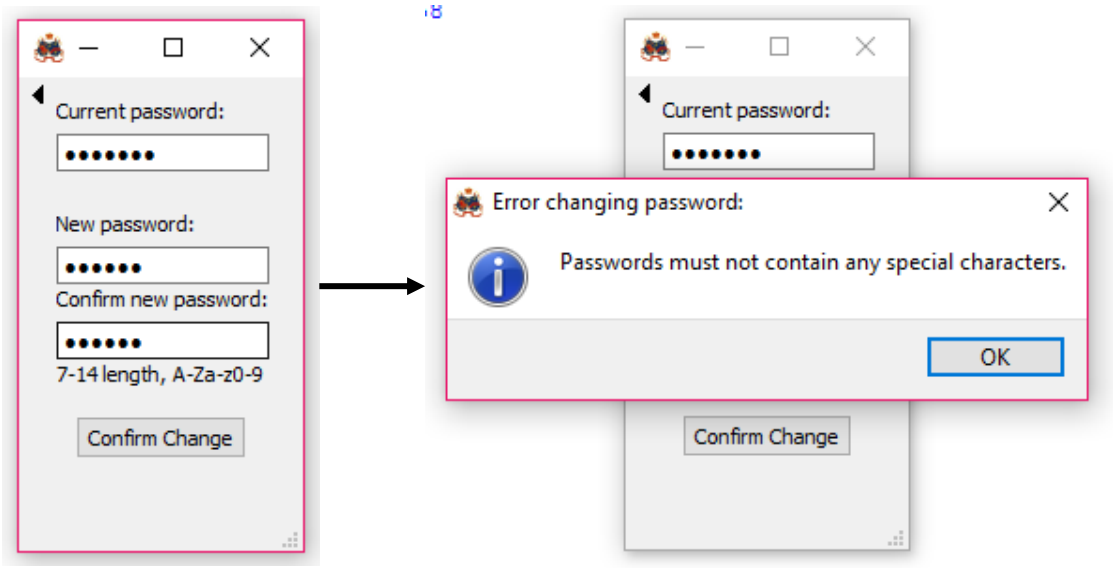
Password change: “Pass12”



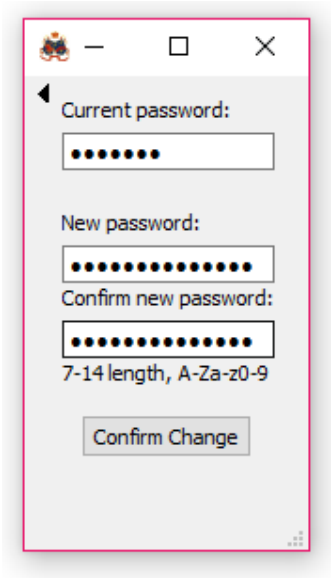
Password change: “Pass$word123”



Password change: “Pass%1”



Password change: “Password123Password”



This final test was unique in that I could not actually put in the whole password due to it exceeding the maximum length that I had enforced on the actual input widget in QtDesigner. As a result, the validation was still successful, simply by not allowing the input to be used.

Testing: overview of testing for change password window

This below table will summarise how the different password inputs were dealt with by the program.

Password example	Intended error(s) of password	Error successfully identified?
“hello”	No capitals or digits, Insufficient length	Yes
“password123”	No capitals	Yes
“Password123”	No errors	Yes
“Password”	No digits	Yes
“Pass12”	Insufficient length	Yes
“Pass$word123”	Special characters used	Yes
“Pass%1”	Special characters used, insufficient length	Yes
“Password123Password”	Too long	Program prevented invalid length being entered (partial success)

As seen, it was able to validate the inputs successfully. The next table will show how this window has achieved all of the objectives related to it.

Objective	**How it was tested**	**Test data used**	**Expected outcome**	**Actual outcome**	**Success? (yes, no, partial)**
9,10.	Test data used to attempt creating various passwords.	See password examples under “input test data”.	“Password123” to be the only accepted passwords. All others will create a pop-up, notifying user of incorrect password.	“Password123” was the only accepted password. All others caused a pop-up to be created, notifying the user of the error.	Yes
11,42.	Change password feature was used.	“password” with user “BrownB1”.	Another login attempt using the new password was carried out to confirm If the password was changed.	The next login attempt only accepted the updated version of the password, proving it had changed.	Yes
30.	Invalid inputs used for line edits.	Shared test with 9,10.	Shared test with 9,10.	Shared test with 9,10.	Yes

As shown, all objectives have been successfully achieved for this window. Therefore, I will now show development for the next relevant section of the program.

Manage users window

This window shall allow administrative users to do the follow things:

- Access the “create user” window
- Make other existing users into administrative users
- Reset other user’s passwords to 123

To add to this, it should also be able to validate if the username that the current user specifies exists or not, and if the specified user is an admin or not. Checking if the existing user exists or not is a type of validation, as it is ensuring that the input is valid.

Validate TeacherUser code

Much like with the regex functions, I have created this function outside of the main program for the purpose of testing. They will use hardcoded inputs at this point, and in later iterations I will implement user input like in the other windows.

```
import sqlite3 as lite #SQL import
con = lite.connect('C:/Users/ryanb/Desktop/Coursework/Project/DetentionPlanner.db')
cur = con.cursor()

def validate(SelectedUser):
    CheckSelectedUser=("""SELECT TeacherUser FROM Teachers
    WHERE TeacherUser = ?""")
    cur.execute(CheckSelectedUser, (SelectedUser, ))
    SelectedUserTest=cur.fetchone()
    #print(SelectedUserTest)
    if SelectedUserTest == None:
        print("TeacherUser doesn't exist.")
    else:
        print("TeacherUser exists.")

test1 = "BrownB1"
print(test1+"\n")
validate(test1)
```

This function is very simple due to how the task it has is also very simple. All it does is attempt to return the TeacherUser from the database, where the TeacherUser = the input. Therefore, nothing will be returned if there is no TeacherUser in the database as specified by the input. This function will be necessary when attempting to make a user into an administrative user or to reset another user’s password, because if it didn’t notify the user that their specified user doesn’t exist, then the user would be lead to think they had successfully altered someone else’s account, when in reality they hadn’t.

Testing: validate TeacherUser

Searching for "BrownB1" (existing user)

```
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\ryanb\Desktop\Coursework\Examples\Random testing\validates Te
acherUser.pyw
BrownB1
TeacherUser exists.
```

Searching for "FakeUser1" (non-existent user)

```
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\ryanb\Desktop\Coursework\Examples\Random testin
acherUser.pyw


FakeUser1
TeacherUser doesn't exist.
>>>
```

As seen in these tests, the function is able to determine if a user exists or not.

Check if admin code

This next function is part of the same test script used for validating users. I simply added another SQL statement to return whether the specified user is an admin or not.

```
def CheckAdmin(SelectedUser):
    CheckSelectedUser=("""SELECT Administrator FROM Teachers
    WHERE TeacherUser = ?""")
    cur.execute(CheckSelectedUser, (SelectedUser, ))
    SelectedUserTest=cur.fetchone()
    if SelectedUserTest == "('YES')":
        print("TeacherUser is an admin.")
    else:
        print("TeacherUser isn't an admin.")

print("\n")
test1 = "FakeUser1"
print(test1)
validate(test1)
CheckAdmin(test1)
```

Testing: check if admin

Checking "BrownB1" (existing user)

```
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\ryanb\Desktop\Coursework\Examples\Random testing\validates Te
acherUser.pyw


BrownB1
TeacherUser exists.
TeacherUser is an admin.
>>>
```

Checking "BrownB1" (existing user)

```
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\ryanb\Desktop\Coursework\Examples\Random testing
acherUser.pyw


FakeUser1
TeacherUser doesn't exist.
TeacherUser isn't an admin.
>>>
```

As seen in these very simple tests, the function can now also identify if a user is an admin or not, on top of checking if the user exists.

Because these functions work successfully, I can easily implement them into the final version of this windows code.

Open “create user” window and back button

This first method is extremely simple, and only serves to open the window which allows admins to create new users.

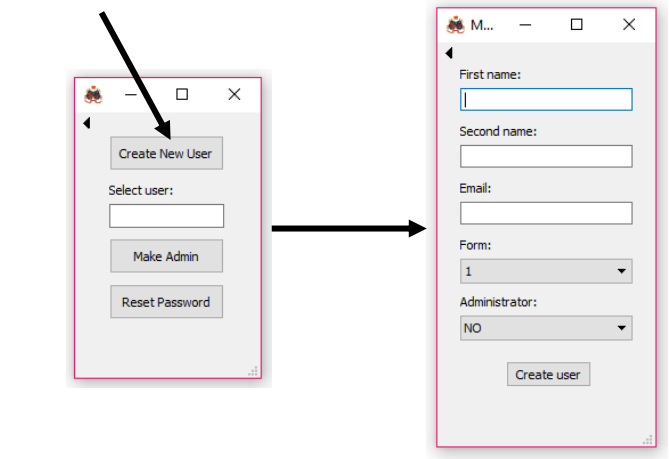
```
def CreateUser(self):
    ManageUsersWindow.hide()
    CreateUserWindow.show()
```

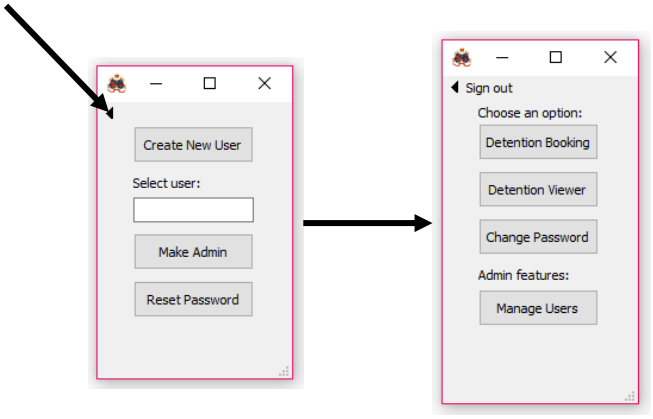
The second method is also very simple, and only serves to open the previous window.

```
def BACK(self):
    ManageUsersWindow.hide()
    MenuWindow.show()
```

Testing: open “create user” window and back button

These tests are very simple and self-explanatory, as I simply needed to click the relevant buttons to see if they were working as intended or not.





These tests show that the buttons are working as intended in this iteration of the code.

MakeAdmin() method

The next stage of developing the code for this window to work was to create the “MakeAdmin” method. This method would utilise the previous functions that check if the user exists. It would then simply use an SQL UPDATE statement to commit the change to the database.

The objectives relevant here are 15 and 44. They are as follows:

15. “*Admins will be able to upgrade other accounts to administrative users*”

44. “*(Update statements (UPDATE, INSERT) will be used when) creating administrative users*”

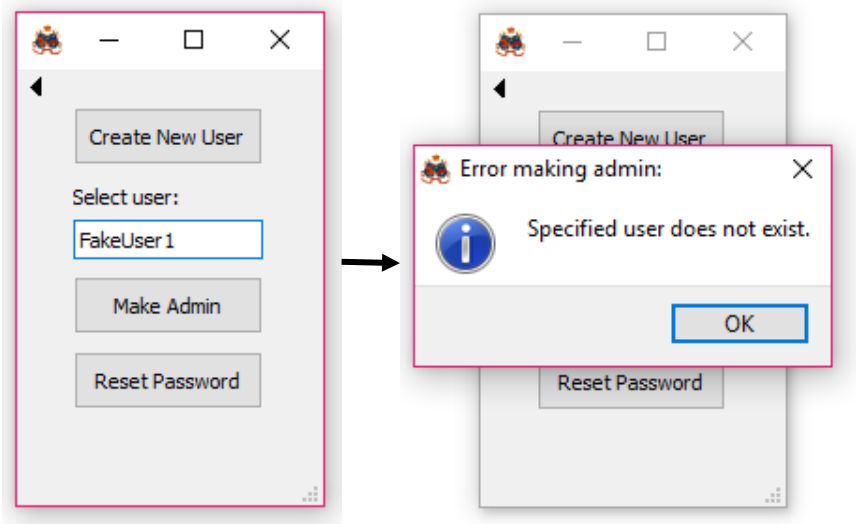
The code for the method is as follows:

```
def MakeAdmin(self):
    print("Logged in as "+username)
    SelectedUser=str(self.LE_SelectUser.text())
    print(SelectedUser) #User that Admin has selected is printed
    MakeAdmin=("""UPDATE Teachers
    SET Administrator = 'YES'
    WHERE TeacherUser = ?""")
    CheckSelectedUser=("""SELECT TeacherUser FROM Teachers
    WHERE TeacherUser = ?""")
    cur.execute(CheckSelectedUser, (SelectedUser, )) #checks if record of specified user exists
    SelectedUserTest=cur.fetchone()
    if SelectedUserTest != None:  #statement determines if user exists
        cur.execute(MakeAdmin, (SelectedUser, )) #Selected User is made an admin
        con.commit()
        QtGui.QMessageBox.information(self, "User made admin:", SelectedUser+" has been made an admin.")
        CheckAdmin=("""SELECT Administrator FROM Teachers
        WHERE TeacherUser = ?""")
        cur.execute(CheckAdmin, (SelectedUser, ))
        AdminCheck=str(cur.fetchone()) #SQL query proves they are an Admin now
        print(AdminCheck)
    else:
        QtGui.QMessageBox.information(self, "Error making admin:", "Specified user does not exist.")
```

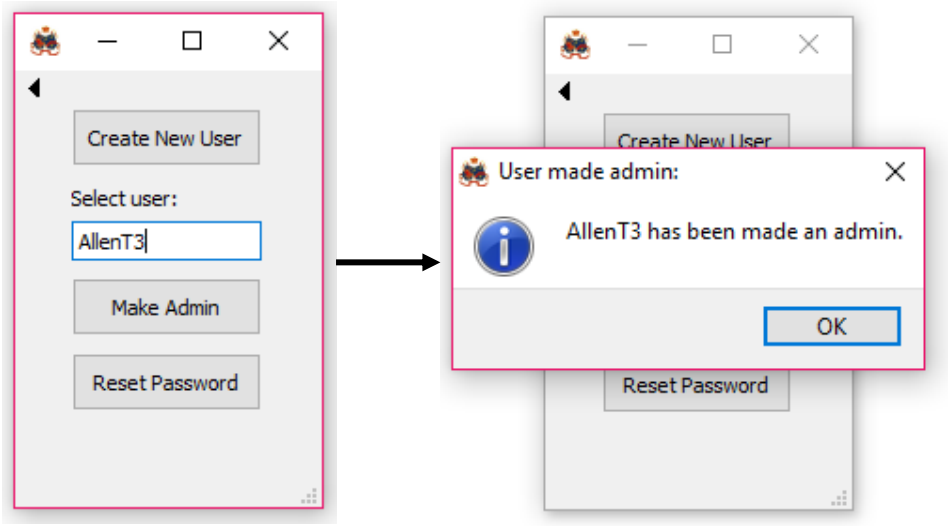
It uses the previous function to determine whether a user exists or not, and if they do, it uses an SQL UPDATE statement to make the user into an admin. I will now carry out a test to show how administrative users can be made, with reference to the database.

Testing: MakeAdmin() method

Making "FakeUser1" an admin (non-existent user)



Making "AllenT3" an admin (existing user)



These screenshots show that the program has been able to identify that the user exists, however, I will need to show the database changes to prove that the actual UPDATE statement has been carried out successfully.

Before:

TeacherID	SecondName	FirstName	TeacherUser	Administrator	Password	Email	FormID
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Brown	Bob	BrownB1	YES	c1deaf40c8af...	brownbob@m...	1
2	Smith	Lisa	SmithL2	YES	d4735e3a265...	smith88@mail...	2
3	Allen	Tim	AllenT3	NO	06ba0e411cfc...	allenallen@m...	3

After:

TeacherID	SecondName	FirstName	TeacherUser	Administrator	Password	Email	FormID
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Brown	Bob	BrownB1	YES	c1deaf40c8af...	brownbob@m...	1
2	Smith	Lisa	SmithL2	YES	d4735e3a265...	smith88@mail...	2
3	Allen	Tim	AllenT3	YES	06ba0e411cfc...	allenallen@m...	3

These database screenshots serve as evidence that the UPDATE statement was carried out successfully. It is therefore fair to conclude that both objectives 15 and 44 have been achieved successfully.

NOTE: after I ran this test, I reset AllenT3's administrative status to "NO".

ResetPass() method

```
def ResetPass(self):
    HashedBasePass = hashlib.sha256("123".encode()).hexdigest()
    SelectedUser=str(self.LE_SelectUser.text())
    print(SelectedUser) #User that Admin has selected is printed
    ResetPassword=("""UPDATE Teachers
    SET Password = ?
    WHERE TeacherUser = ?""")
    CheckSelectedUser=("""SELECT TeacherUser FROM Teachers
    WHERE TeacherUser = ?""")
    cur.execute(CheckSelectedUser, (SelectedUser, )) #checks if record of specified user exists
    SelectedUserTest=cur.fetchone()
    if SelectedUserTest != None:  #if statement determines if user exists
        cur.execute(ResetPassword, (HashedBasePass, SelectedUser, )) #sets password (to hashed 123) of chosen user
        con.commit()
        QtGui.QMessageBox.information(self, "Password reset successful:", SelectedUser+"'s password has been reset to 123.")
        PasswordChecking=("""SELECT Password FROM Teachers
        WHERE TeacherUser = ?""") #Query grabs password to check if updated
        cur.execute(PasswordChecking, (SelectedUser, ))
        PasswordChecker=cur.fetchone()
        PasswordChecker=str(PasswordChecker)
        print("Current password:"+PasswordChecker)
    else:
        QtGui.QMessageBox.information(self, "Error resetting password:", "Specified user does not exist.")
```

This code is very similar to previous method, as its function is very similar. It again checks if the specified user exists, and if they do, it uses an SQL UPDATE statement. However, instead of changing the administrator field, it changes the password field. It Updates the password to a hashed version of 123.

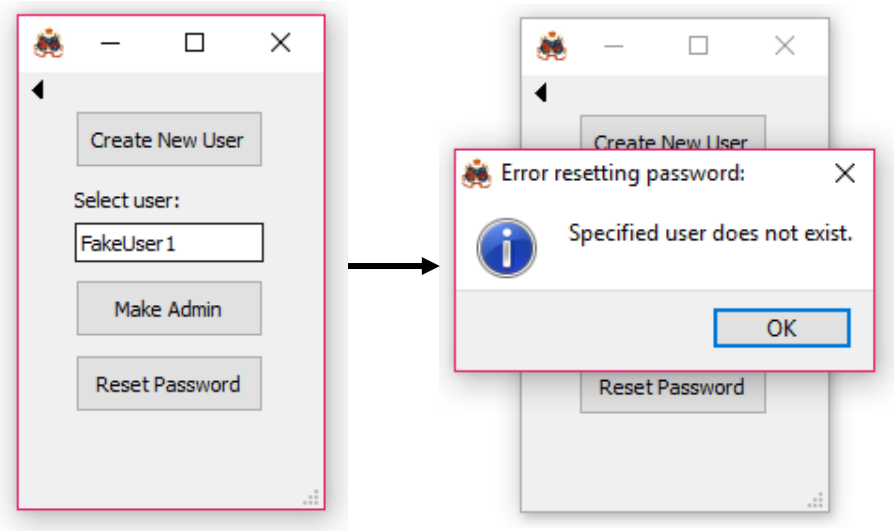
This method's functionality is relevant to the following objectives:

14. "*Admins will be able to reset other user's passwords "123", for if they forget theirs*"

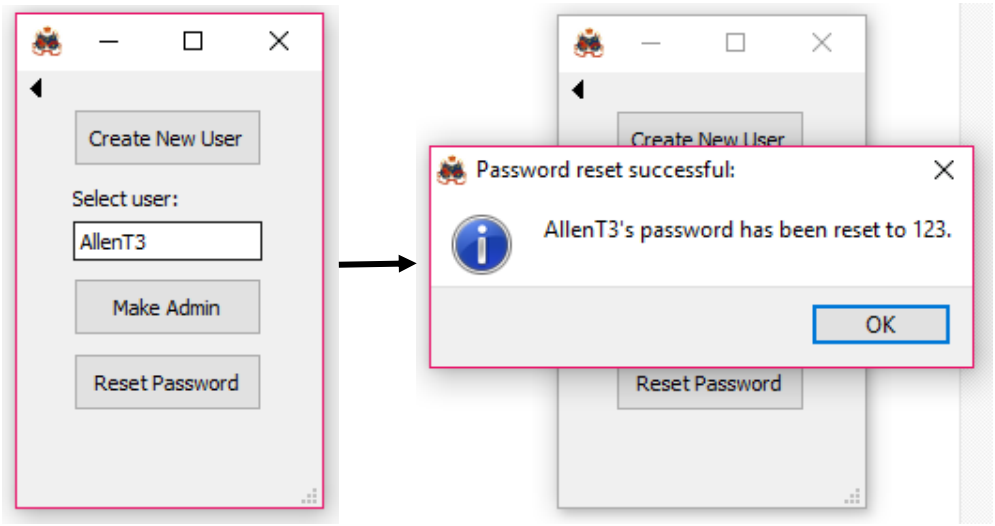
43. "*(Update statements (UPDATE, INSERT) will be used when) resetting passwords*"

Testing: ResetPass() method

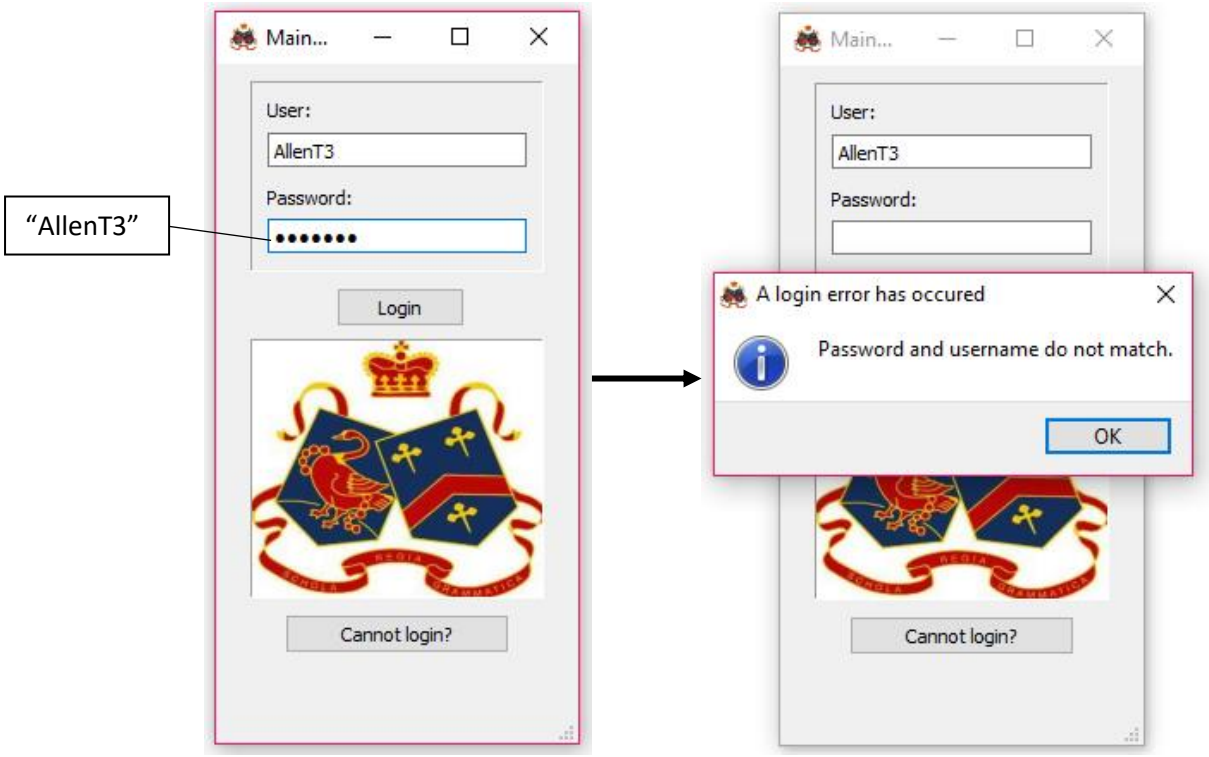
Resetting password of "FakeUser1" (non-existent user)

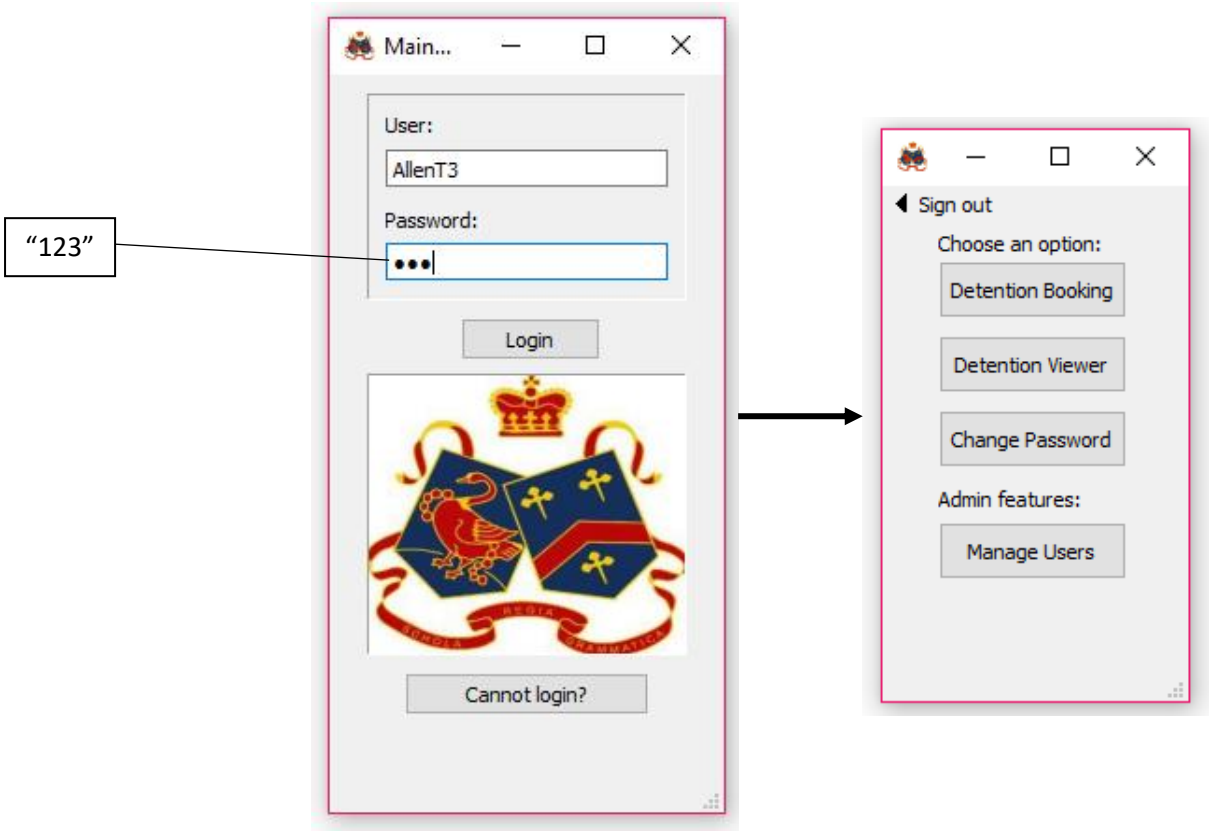


Resetting password of "AllenT3" (existing user)



I will not attempt logins with AllenT3, using both the previous password ("AllenT3") and the new password ("123"), to confirm that the password has been successfully changed.





These screenshots successfully demonstrate that the password change worked as intended.

NOTE: I reset AllenT3's password back to "AllenT3" after these tests were completed.

Objective	How it was tested	Test data used	Expected outcome	Actual outcome	Success? (yes, no, partial)
15,44.	Test user 1 will attempt to make test user 2 an admin.	Test user 1 = "BrownB1". Test user 2 = "AllenT3".	Database will show test user 2 to be an administrator, where they were previously not.	Database will show test user 2 to be an administrator, where they were previously not.	Yes
14,43.	Test user 1 will attempt to reset password of test user 2.	Test user 1 = "BrownB1". Test user 2 = "AllenT3".	Password of test user 1 was successfully changed to "123".	Password of test user 1 was successfully changed to "123".	Yes

I can conclude that the success criteria related to this window have been achieved.

Create new user window

The following objectives should be achieved if this window is functioning as intended:

16. "*Admins will be able to create new users…*"

17. "*…specifying their account type and adding their details to the database…*"

18. "*…with a completely unique user ID…*"

19. "*…and a hashed version of "123" as their password*"

41. "*Update statements (UPDATE, INSERT) will be used when creating new users…*"

As a result, I ensured that all of these objectives were complete by the time I was finished with development of this window.

As the only functionality for this window was taking inputs, creating a new TeacherID and a new TeacherUser, and finally carrying out an SQL INSERT statement, there was only one version required to get the class working as intended.

Class

```
class CreateUserWindowClass(QtGui.QMainWindow, CreateUserWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        ##################
        self.BTN_CreateNewUser.clicked.connect(self.CreateNewUser)
        self.TB_BACK.clicked.connect(self.BACK)
        ##################
    def BACK(self):
        CreateUserWindow.hide()
        ManageUsersWindow.show()
    def CreateNewUser(self):
```

As seen, there are only 2 buttons associated with this class, and each of them cause a method to be called when they are clicked. The more complex method is the CreateNewUser method however, and so I will go into more depth on it here.

CreateNewUser()

The whole method may be seen here:

```
    def CreateNewUser(self):
        FirstName=str(self.LE_FirstName.text())
        SecondName=str(self.LE_SecondName.text())
        Email=str(self.LE_Email.text())
        FormID=str(self.CB_Form.currentText())
        Administrator=str(self.CB_Administrator.currentText()) #makes a variable for each user input
        HashedBasePass = hashlib.sha256("123".encode()).hexdigest()
        cur.execute("""SELECT TeacherID FROM Teachers
        ORDER BY TeacherID DESC""") #Finds largest and therefore most recent TeacherID
        HighestTeacherID1=cur.fetchone()
        HighestTeacherID1=HighestTeacherID1[0]
        HighestTeacherID=int(HighestTeacherID1)
        NextTeacherID=HighestTeacherID+1 #Works out next TeacherID by incrementing
        NewTeacherIDstr=str(NextTeacherID)
        FirstNameList=list(FirstName)
        FirstLetter=FirstNameList[0]
        TeacherUsername=SecondName+FirstLetter+NewTeacherIDstr #Creates a TeacherUser
        print(TeacherUsername)
        InsertUser=("""INSERT INTO Teachers (TeacherID, SecondName, FirstName, TeacherUser, Administrator, Password, Email, FormID)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?)""") #inserts data into table
        cur.execute(InsertUser, (NextTeacherID, SecondName, FirstName, TeacherUsername, Administrator, HashedBasePass, Email, FormID, ))
        con.commit()
        QtGui.QMessageBox.information(self, "User created successfully.", TeacherUsername+"'s account created. Password: 123.")
        CreateUserWindow.hide()
        MenuWindow.show()
```

The first part of this method is essentially just taking inputs from the various widgets that are on the "create user" window. It does nothing with them at this point however:

```
FirstName=str(self.LE_FirstName.text())
SecondName=str(self.LE_SecondName.text())
Email=str(self.LE_Email.text())
FormID=str(self.CB_Form.currentText())
Administrator=str(self.CB_Administrator.currentText()) #makes a variable for each user input
HashedBasePass = hashlib.sha256("123".encode()).hexdigest()
```

The next part is used to find the **latest** record in the teachers table by using an SQL SELECT query. It then increments the TeacherID of this record, to find the new TeacherID. This is used in its own right as one of the fields that needs to be present in the record. However, it is also used to create a unique TeacherUser. It does this by taking the whole of the second name that has been input, then adding the first letter of the first name and finally the TeacherID after that. This can be seen here:

```
cur.execute("""SELECT TeacherID FROM Teachers
ORDER BY TeacherID DESC""") #Finds largest and therefore most recent TeacherID
HighestTeacherID1=cur.fetchone()
HighestTeacherID1=HighestTeacherID1[0]
HighestTeacherID=int(HighestTeacherID1)
NextTeacherID=HighestTeacherID+1 #Works out next TeacherID by incrementing
NewTeacherIDstr=str(NextTeacherID)
FirstNameList=list(FirstName)
FirstLetter=FirstNameList[0]
TeacherUsername=SecondName+FirstLetter+NewTeacherIDstr #Creates a TeacherUser
```

Now that all the inputs are prepared for creating the new record, all that is left is the SQL query:

```
InsertUser=("""INSERT INTO Teachers (TeacherID, SecondName, FirstName, TeacherUser, Administrator, Password, Email, FormID)
VALUES (?, ?, ?, ?, ?, ?, ?, ?)""") #inserts data into table
cur.execute(InsertUser, (NextTeacherID, SecondName, FirstName, TeacherUsername, Administrator, HashedBasePass, Email, FormID, ))
con.commit()
```

The question marks are used to pass variables into the query, and the variables used are shown on the following line.

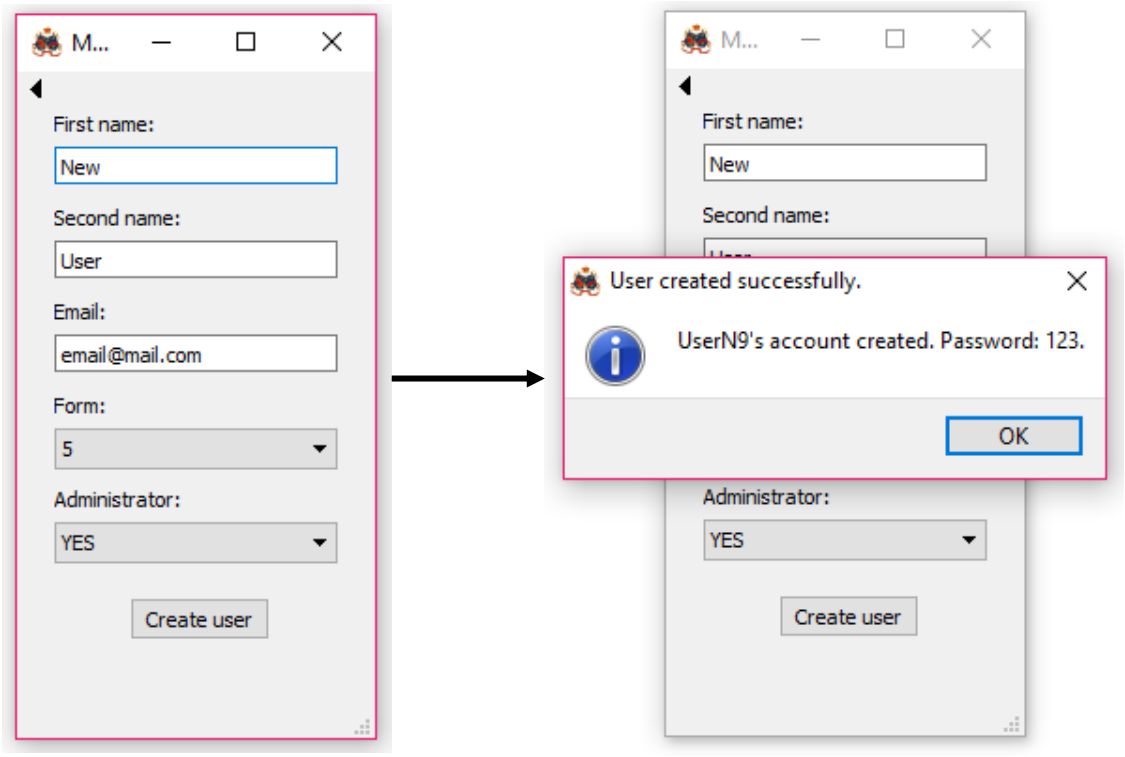
Testing: CreateNewUser()

In order for the prior mentioned objectives to be successfully achieved, I must show the test that I carried out which will create a new user.

The credentials of the new user can be seen here:

TeacherID	Surname	FirstName	Administrator?	Email	FormID
(dependent on number of existing records)	User	New	YES	email@mail.com	5

Attempting to create the new user



I will now present the before and after database screenshots to prove that the new record was successfully created. Once I have done that, I will show how the program allowed the login of this new user with its credentials, providing further proof that the account was created successfully. It will also be able to access the "mange users" window, due it being as an administrative account.

Before:

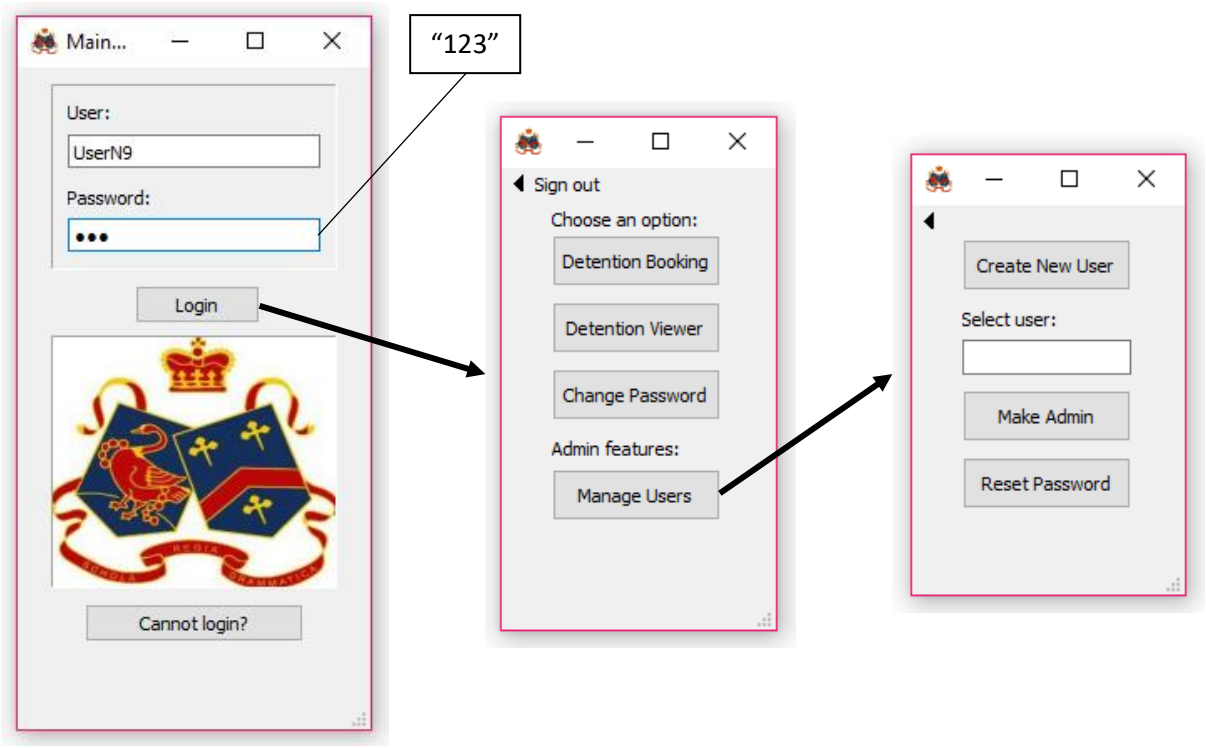
TeacherID	SecondName	FirstName	TeacherUser	Administrator	Password	Email	FormID
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Brown	Bob	BrownB1	YES	c1deaf40c8af...	brownbob@m...	1
2	Smith	Lisa	SmithL2	YES	d4735e3a265...	smith88@mail...	2
3	Allen	Tim	AllenT3	NO	06ba0e411cfc...	allenallen@m...	3
4	Hunt	Sam	HuntS4	NO	4b227777d4d...	hunt87@mail....	4
5	Cook	Megan	CookM5	NO	ef2d127de37b...	meganc@mail...	5
6	Scott	Holly	ScottH6	NO	e7f6c011776e...	hs2@mail.com	6
7	Adams	Madison	AdamsM7	NO	7902699be42...	madams@ma...	7
8	Edwards	Tom	EdwardsT8	YES	2c624232cdd...	tomedwards...	8

After:

TeacherID	SecondName	FirstName	TeacherUser	Administrator	Password	Email	FormID
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Brown	Bob	BrownB1	YES	c1deaf40c8af...	brownbob@m...	1
2	Smith	Lisa	SmithL2	YES	d4735e3a265...	smith88@mail...	2
3	Allen	Tim	AllenT3	NO	06ba0e411cfc...	allenallen@m...	3
4	Hunt	Sam	HuntS4	NO	4b227777d4d...	hunt87@mail....	4
5	Cook	Megan	CookM5	NO	ef2d127de37b...	meganc@mail...	5
6	Scott	Holly	ScottH6	NO	e7f6c011776e...	hs2@mail.com	6
7	Adams	Madison	AdamsM7	NO	7902699be42...	madams@ma...	7
8	Edwards	Tom	EdwardsT8	YES	2c624232cdd...	tomedwards...	8
9	User	New	UserN9	YES	a665a459204...	email@mail.c...	5

As seen in the above screenshots, the SQL UPDATE statement was clearly successful, thus proving the prior mentioned objectives were achieved. However, I will still show the new account being successful logged in with, whilst also proving it has administrative rights.

Attempting login with the new account



Due to this final test, I can conclude that this iteration of my program had successfully achieved the prior mentioned objectives.

Objective	How it was tested	Test data used	Expected outcome	Actual outcome	Success? (yes, no, partial)
16,18,41.	Using "create new user" functionality of the program.	See creating new users table, under "input test data".	Database screenshots show that the new user was created successfully.	Database screenshots show that the new user was created successfully.	Yes
17.	Attempting to use an administrative user feature with the new account.	See creating new users table, under "input test data".	Account successfully performed action that requires an administrator status.	Account successfully performed action that requires an administrator status.	Yes
19.	Attempting to log in using the new account.	^ Using the password "123".	Account successfully logged in with use of "123" as a password.	Account successfully logged in with use of "123" as a password.	Yes

Detention booking window

The following objectives should be achieved if this window is functioning as intended:

40. "*(Update statements (UPDATE, INSERT) will be used when) creating a new detention record…*"

41. "*…which will also increment the offending student's detention total*"

As a result, I ensured that both of these objectives were complete by the time I was finished with development of this window.

Time formatting code

Due to the how the input taken from the detention booking window being in the form "HH:MM:SS", I will have to reformat it, converting it to a form that I can work with.

Since I must be able to ensure that detentions don't overlap, I decided to store detention times as an integer of minutes after midnight. This was a justifiable decision since the user will not typically be looking at the database in a raw view, and it will make working with times a lot easier. This is because I can compare the time periods that detentions extend, to determine whether they will overlap or not. Therefore, my first iteration for making this section involved creating the necessary functions that would be able to convert these time strings into workable formats. Once I got these functions to work, I could then create a module with them, which would allow me to import the functions as necessary.

My first step was to initialise the variables that I would be converting.

```
Date = "2000-01-01"
RoomID = 1
StudentID = 1
TeacherID = 1

TimeStart = "15:40:00"
Duration = "20"
print("Date: "+Date)
print("TimeStart: "+TimeStart)
print("Duration:"+Duration)
print("")
```

The first function required will take a parameter of a time string, in this case, TimeStart.

```
def ConvertStartToMins(TimeStart):
    Times=TimeStart.split(":")
    Times.pop() #Creates list, removing seconds
    HoursConverted = (int(Times[0])*60) #converts hours to mins
    Mins = int(Times[1])
    TimeStartMins = HoursConverted + Mins #finds minutes relative to 00:00
    print(TimeStartMins)
    return TimeStartMins
```

The function here will remove seconds from the list, due to it being unnecessary to consider seconds when considering the times of detentions.

This intention of this function was to return a converted form of the input time, to an integer of minutes after midnight. The commenting with the code shows how it does this.

The next function simply uses the duration, and the time start (which has been converted to an integer of minutes after midnight, and adds them together to find the time end as an integer of minutes after midnight.

```
def EndInMins(TimeStartMins, Duration):
    Duration = int(Duration)
    TimeEndMins = TimeStartMins + Duration
    print(TimeEndMins)
    return TimeEndMins #calculates end in minutes relative to 00:00
```

Although working with this integer format of times, it is not very user-friendly, as the user would have to convert the number of minutes in their head to a 24-hour equivalent. Therefore, I created a function that would be able to take a time relative to midnight in minutes, and convert it to the original format, which is a string in the form of “HH:MM:SS”.

```
def ConvertToString(TimeInMins):
    Mins = TimeInMins%60 #MOD gives minutes
    Hours = TimeInMins//60 #integer division returns hours
    if Mins < 10: #string formatting
        Mins = "0"+str(Mins)
    else:
        Mins = str(Mins)

    if Hours < 10: #string formatting
        Hours = "0"+str(Hours)
    else:
        Hours = str(Hours)

    TimeInMins = Hours+":"+Mins+":00" #string formatting
    print(TimeInMins)
    return TimeInMins
```

I use integer division to return the number of whole hours, and “MOD” to return the number of left over minutes after the hours have been found. The “if” statements only serve to ensure that the string appears as intended. They do this by checking if the number of a variable is less than 10, and if it is, it adds a “0” before it. This means times such as 01:30:00 will not appear as 1:30:00.

It is then simply a case of adding the hours, minutes and then “00” for seconds together in a single string. This will be used for output, as opposed to actual processing.

Finally, a simple function was required to convert the dates as they are given in PyQt, to a more universal standard. As dates are received in the form: YY:MM:DD, but I want them in the form: DD:MM:YY.

```
def DateReformat(Date):
    DateList = Date.split("-")
    Date = DateList[2]+"-"+DateList[1]+"-"+DateList[0]
    return Date
```

Testing: Time formatting code

Here I am simply going to show the outputs that are created with the following hardcoded inputs.

```
Date = "2000-01-01"
RoomID = 1
StudentID = 1
TeacherID = 1

TimeStart = "15:40:00"
Duration = "20"
print("Date: "+Date)
print("TimeStart: "+TimeStart)
print("Duration:"+Duration)
print("")
```

When the prior mentioned functions are called using the above inputs, the following output is created.

```
>>>
 RESTART: C:\Users\ryanb\Desktop\Coursework\Exampl
+reformats dates).py
Date: 2000-01-01
TimeStart: 15:40:00
Duration:20

940
960
16:00:00
01-01-2000
>>>
```

940 represents the number of minutes after midnight, which should be equal to “15:40:00”. I will show this now: 15*60 = 900. 900+40 = 940.

940 + 20 (the duration of the detention) = 960. This converted to a time string will then end up 16:00:00.

Finally, the date can clearly be seen to have simply been reversed, as intended.

I can therefore conclude that these functions are working intended, and so they were made into their own module, to be later used in the main program.

module_FormattingFunctions.py - C:\Users\ryanb\Desktop\Coursework\Project\module_FormattingFunctions.py

File Edit Format Run Options Windows Help

```
def FetchTreater(oldtuple): #Function used to format result from an SQL SELECT query
    newstring = str(oldtuple)
    newstring = newstring.replace("(","")
    newstring = newstring.replace(")","")
    newstring = newstring.replace(",","")
    return newstring

def ConvertTimeToMins(TimeString): #Function converts "time-strings" into integers of minutes after 00:00:00
    Times=TimeString.split(":")
    Times.pop() #Creates list, removing seconds
    HoursConverted = (int(Times[0])*60) #Converts hours to mins
    Mins = int(Times[1])
    TimeStartMins = HoursConverted + Mins #Finds minutes relative to 00:00
    #print(TimeStartMins)
    return TimeStartMins

def EndInMins(TimeStartMins, Duration):
    Duration = int(Duration)
    TimeEndMins = TimeStartMins + Duration
    #print(TimeEndMins)
    return TimeEndMins #Calculates end in minutes relative to 00:00

def ConvertToTimeString(TimeInMins): #Function converts integers of minutes after 00:00:00 into a "time-string"
    Mins = TimeInMins%60 #MOD gives minutes
    Hours = TimeInMins//60 #Integer division returns hours
    if Mins < 10: #String formatting
        Mins = "0"+str(Mins)
    else:
        Mins = str(Mins)
    if Hours < 10: #String formatting
        Hours = "0"+str(Hours)
    else:
        Hours = str(Hours)
    TimeInMins = Hours+":"+Mins+":00" #String formatting
    #print(TimeInMins)
    return TimeInMins

def DateReformat(Date): #String formatting to make dates from YYYY/MM/DD to DD/MM/YYYY
    DateList = Date.split("-")
    Date = DateList[2]+"-"+DateList[1]+"-"+DateList[0]
    return Date
```

This is how all the different functions appear in the “module_FormattingFunctions.py” file. I will alter demonstrate how these will be utilised in the main program.

Validate detention records code

The next step was to write an algorithm which would be capable of determining if a new detention record would be colliding with any existing detention records. Either due to a student being in multiple detentions at the same time, or a room being used for multiple detentions at the same time. It would also be required to determine which of these two circumstances was creating the issue, as discussed in the design section of this document.

Initially, I will demonstrate the algorithm I created, based off the pseudocode idea in my design section, and then I will test whether it worked successfully or not when it was implemented into the main program.

```
cmd = ("""SELECT DetentionID FROM Detentions
WHERE StudentID = ? AND Date = ? AND (TimeStart BETWEEN ? AND ? OR TimeEnd BETWEEN ? AND ?)""")
cur.execute(cmd, (StudentID, Date, TimeStartMins, TimeEndMins, TimeStartMins, TimeEndMins, ))
result = cur.fetchall()
print(result)
if len (result) > 0:
    print("Student is already in detention during the desired time period.")
else:
    print("Student is free.")
    cmd = ("""SELECT DetentionID FROM Detentions
    WHERE RoomID = ? AND Date = ? AND (TimeStart BETWEEN ? AND ? OR TimeEnd BETWEEN ? AND ?)""")
    cur.execute(cmd, (RoomID, Date, TimeStartMins, TimeEndMins, TimeStartMins, TimeEndMins, ))
    result = cur.fetchall()
    print(result)
    if len (result) > 0:
        print("Room is already in use during the desired time period.")
    else:
        print("Detention successfully booked.")
        pass
```

There is no present commenting for this algorithm in this iteration, due to it not being implemented into the main program yet.

Essentially, the first SQL SELECT statement:

```
cmd = ("""SELECT DetentionID FROM Detentions
WHERE StudentID = ? AND Date = ? AND (TimeStart BETWEEN ? AND ? OR TimeEnd BETWEEN ? AND ?)""")
cur.execute(cmd, (StudentID, Date, TimeStartMins, TimeEndMins, TimeStartMins, TimeEndMins, ))
```

Is used to return ANY DetentionID where the StudentID and the date are the same as the new detention, and if either their start or end time are located within the time period of the new detention record.

The algorithm then alerts the user if there are any such detention records, which are using the same student, at the same time.

Then, another SQL statement is used, which is almost identical:

```
    cmd = ("""SELECT DetentionID FROM Detentions
    WHERE RoomID = ? AND Date = ? AND (TimeStart BETWEEN ? AND ? OR TimeEnd BETWEEN ? AND ?)""")
    cur.execute(cmd, (RoomID, Date, TimeStartMins, TimeEndMins, TimeStartMins, TimeEndMins, ))
```

However, it is checking if the same room is in use, as opposed to the same student.

Although both of these statements could have been made into one, the purpose of having two separate statements is so that the program can identify which issue is causing the new detention record to be invalid. The reason this is important, is so that the teacher knows if they have to try to book the same student in a different room, or if they must move the detention to a different time or date.

CreateDetention() method

The following method is very long, due to it requiring the formatting of many inputs, as given by the user.

```
class DetentionBookingWindowClass(QtGui.QMainWindow, DetentionBookingWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        ##################
        self.TB_BACK.clicked.connect(self.BACK)
        self.BTN_BookDetention.clicked.connect(self.CreateDetention)
        ##################
        global username
        self.StudentID = None
        self.Student = None
        self.Date = None
        self.TimeStart = None
        self.Duration = None
        self.Description = None
        self.Room = None
        self.FirstName = None
        self.SecondName = None    #all variables required for creating a detention record
        self.TimeStartMins = None
        self.TimeEndMins = None
```

Firstly, here is the class, without its methods. As seen by the attributes in the constructor method, there are a fair number of variables that need to be considered when booking a detention.

The first step of the CreateDetention() method, is to simply take the user inputs for most of these attributes, from the GUI.

```
    def CreateDetention(self):  #module used to create a new detention record at user's request
        #self.DateRAW = self.DateInput.date()
        #self.TimeStartRAW = self.TimeStartInput.time()
        self.FirstName = str((self.LE_FirstName.text())) #variables obtained from GUI
        self.SecondName = str((self.LE_SecondName.text()))
        self.Date = str((self.DateInput.date()).toPyDate()) #YYYY/MM/DD
        self.TimeStart = str((self.TimeStartInput.time()).toPyTime())
        self.Duration = int(self.DurationInput.text())
        self.Offense = str(self.Desc_Box.toPlainText())
        self.Room = (self.CB_RoomNo.currentText())
        self.TimeEnd = None
```

Once it does this, some of these given variables are reformatted, for varying reasons.

```
#Variable for detention form:
self.StudentName = self.FirstName+" "+self.SecondName
#print(self.StudentName)
self.SlipDate = module_FormattingFunctions.DateReformat(self.Date) #Uses my function to for
#TimeStart
self.TimeStartMins = module_FormattingFunctions.ConvertTimeToMins(self.TimeStart) #Uses my
print("TIME START:"+str(self.TimeStartMins))

#TimeEnd
self.TimeEndMins = module_FormattingFunctions.EndInMins(self.TimeStartMins, self.Duration)
print("TIME END:"+str(self.TimeEndMins))
```

Unfortunately, I had to crop out some comments, but it is clear how I have used the previously mentioned formatting modules to re-arrange the data and convert the times to minutes.

The next part of the method is used to create some of the variables which are required for the detention slip.

```
#RoomID
FindRoomID = ("""SELECT RoomID FROM Rooms
WHERE RoomNumber = ?""")
cur.execute(FindRoomID, (self.Room, )) #RoomID worked out by using the RoomNumber (as given by user's input)
self.RoomID = str(cur.fetchone()[0]) #Should identify block here
print("ROOM ID: "+self.RoomID)
#Variable for detention form
FindBlock = ("""SELECT Block FROM Rooms
WHERE RoomNumber = ?""")
cur.execute(FindBlock, (self.Room, )) #Find the block for the DetentionForm
self.Block = cur.fetchone()[0]
self.Location = "Room "+self.Room+", "+self.Block+" block" #For DetentionForm
```

First, the RoomNumber is used (as given by the teacher when they booked the detention), to find the RoomID. This is essential, as my detentions table in the database stores the RoomID, as opposed to the RoomNumber.

It also uses this RoomNumber to work out the Block of the room, from the rooms table. This is important because the “Location” attribute is comprised of the room number and the block. Location will be used to display where the detention is in the detention form.

Another important part of this method is how the detention record will need a new DetentionID. This very simple SQL SELECT query is used to achieve this, by finding the DetentionID of the latest detention record, and then incrementing it.

```
#DetentionID
cur.execute("""SELECT DetentionID FROM Detentions
ORDER BY DetentionID DESC""") #finds largest and therefore most recent DetentionID
HighestDetentionID = cur.fetchone()
HighestDetentionID = int(HighestDetentionID[0])
NextDetentionID = str(HighestDetentionID+1) #works out next DetentionID by incrementing
print("DETENTION ID: "+NextDetentionID)
```

Finally, the TeacherID must be found, for the sake of creating the new detention record. This is relatively simple, as the global “username” variable can easily be used to find the TeacherID from the Teachers table. However, 2 other statements are also used to return the first and second name of the teacher, due to this being necessary for displaying the name of the teacher on the detention slip. It uses the first letter of the teacher’s first name, and their second name. For example, a teacher called “Tim Allen”, would have their name displayed as T Allen on the detention slip.

```
#TeacherID
#print("TeacherUser: "+username)
FindTeacherID = ("""SELECT TeacherID FROM Teachers
WHERE TeacherUser = ?""")
cur.execute(FindTeacherID, (username, )) #uses global "username", to find the TeacherID (i.e. who is logged in)
self.TeacherID = str(cur.fetchone()[0])
print("TeacherID: "+self.TeacherID)
#TeacherName for DetentionForm
FindTeacherFirstName = ("""SELECT FirstName FROM Teachers
WHERE TeacherID = ?""")
cur.execute(FindTeacherFirstName, (self.TeacherID, ))
TeacherFirstName = cur.fetchone()[0]
FindTeacherSecondName = ("""SELECT SecondName FROM Teachers
WHERE TeacherID = ?""")
cur.execute(FindTeacherSecondName, (self.TeacherID, ))
TeacherSecondName = cur.fetchone()[0]
self.TeacherName = TeacherFirstName[0]+" "+TeacherSecondName #Creates TeacherName for use in DetentionForm
```

The final parts of this algorithm are sued to verify whether the input student exists or not, and then to determine whether there are any other detention records which would collide with this new record.

```
#StudentID
CheckSelectedStudent = ("""SELECT StudentID FROM Students
WHERE (FirstName = ?) AND (SecondName = ?)""")   #SQL statement finds StudentID where first name
cur.execute(CheckSelectedStudent, (self.FirstName, self.SecondName, ))
self.StudentID = cur.fetchall()
if len(self.StudentID) == 0: #if no StudentID's are returned, the student does not exist
    QtGui.QMessageBox.information(self, "Booking error", "Student does not exist.              ")
else:
    self.StudentID = self.StudentID[0] #else, there is only 1 student with the name, and so a det
    self.StudentID = module_FormattingFunctions.FetchTreater(self.StudentID) #module_FormattingFu
```

This algorithm shows how the program will use the first and second name, as input by the user, to try and find a StudentID. If it cannot return a StudentID, it means the credentials used by the teacher were invalid, and thus it will notify the user.

Once the student has been verified to exist, the program can use another SELECT query, with use of an inner join to determine the form of the student. It does this due to how the detention slip requires the form group of the student.

```
FindForm = ("""SELECT Forms.FormName FROM Forms
INNER JOIN Students ON Students.FormID = Forms.FormID
WHERE Students.StudentID = ?""")
cur.execute(FindForm, (self.StudentID, )) #Used to get FormName for use in creating DetentionForm
self.Form = cur.fetchone()[0]
```

Now that all the variables can be obtained for use in the detention slip, the only steps left for creating the new detention record are ensuring there are no clashes, and then carrying out the actual “INSERT” statement.

```
self.TimeEnd = module_FormattingFunctions.ConvertToTimeString(self.TimeEndMins)
print("TIME ENDS AT: "+self.TimeEnd)
CheckStudent = ("""SELECT DetentionID FROM Detentions
WHERE StudentID = ? AND EventDate = ? AND (TimeStart BETWEEN ? AND ? OR TimeEnd BETWEEN ? AND ?)""")
cur.execute(CheckStudent, (self.StudentID, self.Date, self.TimeStartMins, self.TimeEndMins, self.TimeStartMins, self.TimeEndMins, ))
result = cur.fetchall() #Statement returns any detentions that are on the same date, with the same student and during the same time period
print(result)
if len (result) > 0:
    QtGui.QMessageBox.information(self, "Booking error", "This student is already in detention during this time period.")
else:
    print("Student is free.")
    CheckRoom = ("""SELECT DetentionID FROM Detentions
    WHERE RoomID = ? AND EventDate = ? AND (TimeStart BETWEEN ? AND ? OR TimeEnd BETWEEN ? AND ?)""")
    cur.execute(CheckRoom, (self.RoomID, self.Date, self.TimeStartMins, self.TimeEndMins, self.TimeStartMins, self.TimeEndMins, ))
    result = cur.fetchall() #Statement returns any detentions that are on the same date, in the same room and during the same time period
    print(result)
    if len (result) > 0:
        QtGui.QMessageBox.information(self, "Booking error", "This room is occupied during this time period.")
    else:
        print("Detention successfully booked.")
```

The above code shows how the previous “validate detention record” algorithm has been integrated into the main program. I will not explain how this part works again as it is essentially the same code that was described in-depth earlier.

If the detention record is valid (meaning the last line of code will be executed), then this extra segment of code will be run.

The following code is used to carry out an SQL INSERT statement, passing in the various variables that have been determined in this algorithm (the variables being passed in are noted by a "?" symbol in the statement).

```
BookDetentionStatement = ("""INSERT INTO Detentions (DetentionID, EventDate, TimeStart, TimeEnd, Duration, RoomID, TeacherID, StudentID, Offense)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)""") #INSERT statement used to create the new record with all of the variables given by the user
cur.execute(BookDetentionStatement, (NextDetentionID, self.Date, self.TimeStartMins, self.TimeEndMins, self.Duration, self.RoomID, self.TeacherID,
#con.commit()
self.UpdateStudentTable() #module increments student's number of detentions within databse
self.BACK() #module closes detention planner and opens previous window
DetentionFormWindow.GenerateHTML(self)
```

Due to each detention record having 9 fields, the insert statement is very long.

The code after the statement shows 3 different methods being called. "self.BACK()" is simply the same method that has been present in the other windows, which takes the user back to the previous screen. "DetentionFormWindow.GenereateHTML(self)" is a method which has not been discussed yet, and it exists in a separate class which has yet to be mentioned. I will go into more detail on this later.

Finally, the "self.UpdateStudentTable()" method is a method which I will now discuss. Essentially, it's purpose is to increment the student's total number of detentions once they have been given a detention.

UpdateStudentTable() method

This method is relatively simple.

```
def UpdateStudentTable(self):  #this module is ran when a detention is booked. It increments detention count of students in the data
    FindStudentDetentions = ("""SELECT DetentionCount FROM Students
    WHERE StudentID = ?""")  #statement finds the detention count of the specified student
    cur.execute(FindStudentDetentions, (self.StudentID, ))
    self.DetentionCount = cur.fetchone()
    self.DetentionCount = module_FormattingFunctions.FetchTreater(self.DetentionCount) #module_FormattingFunctions.FetchTreater usec
    print("Prior detentions for "+self.FirstName+" "+self.SecondName+": "+self.DetentionCount)
    self.DetentionCount = int(self.DetentionCount)
    self.DetentionCount = self.DetentionCount + 1 #increments DetentionCount
    IncrementStudentsDetentions = ("""UPDATE Students
    SET DetentionCount = ?
    WHERE StudentID = ?""")
    cur.execute(IncrementStudentsDetentions, (self.DetentionCount, self.StudentID, )) #statement updates detention count for student
    #con.commit()
    QtGui.QMessageBox.information(self, "Booking success", "Detention for "+self.FirstName+" "+self.SecondName+" has been booked.")
```

All that is done in this method is the DetentionCount is retrieved from the Students table, by using the StudentID, it is then incremented and finally the DetentionCount in the table is updated with the new, incremented value.

I will now proceed to test the functionality of this class.

Testing: detention booking functionality

For these tests, I will attempt to create the following detention records, and then present the output produced by the program. The following detention records that I will attempt to input can be seen here:

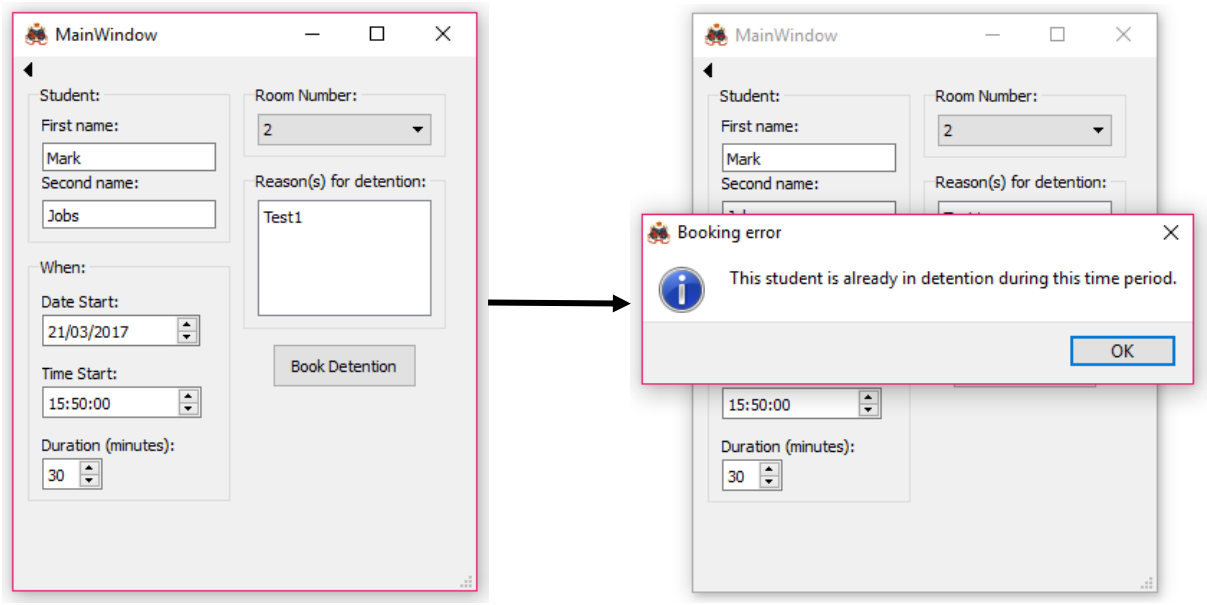
DetentionID	EventDate	TimeStart	TimeEnd	Duration	RoomID	TeacherID	StudentID	Offense
3	2017-03-21	950	980	30	2	1	1	Test1
4	2017-03-21	930	950	20	1	1	2	Test2
5	2017-03-22	950	970	20	1	1	1	Test3
6	2017-03-22	945	965	20	8	3	2	Test4

This is the database prior to any new detention records being booked. This is relevant as the program will be checking the existing records to see if there are going to be any clashes.

If all works as intended, detention record 6 should be the only successfully booked detention.

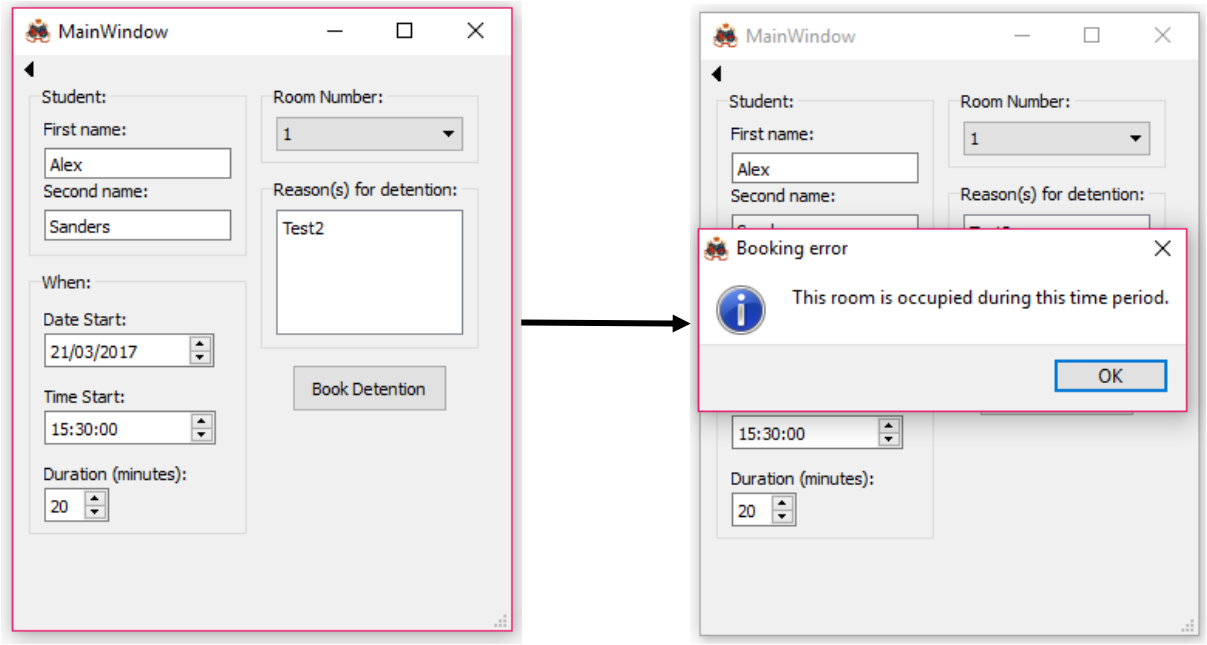
DetentionID	EventDate	TimeStart	TimeEnd	Duration	RoomID	TeacherID	StudentID	Offense
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	2017-03-21	940	960	20	1	1	1	Late
2	2017-03-22	960	990	30	7	1	1	Rude

Creating DetentionID 3



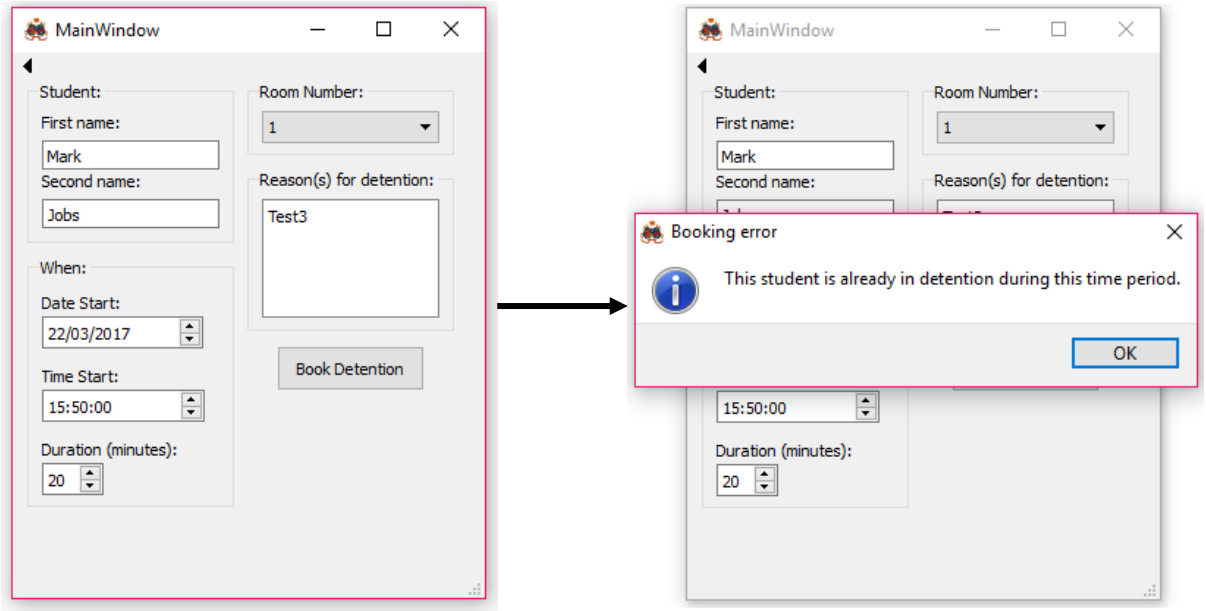
As seen, the program is able to identify that the student is currently in a detention during this time period. It clashes with DetentionID = 1.

Creating DetentionID 4



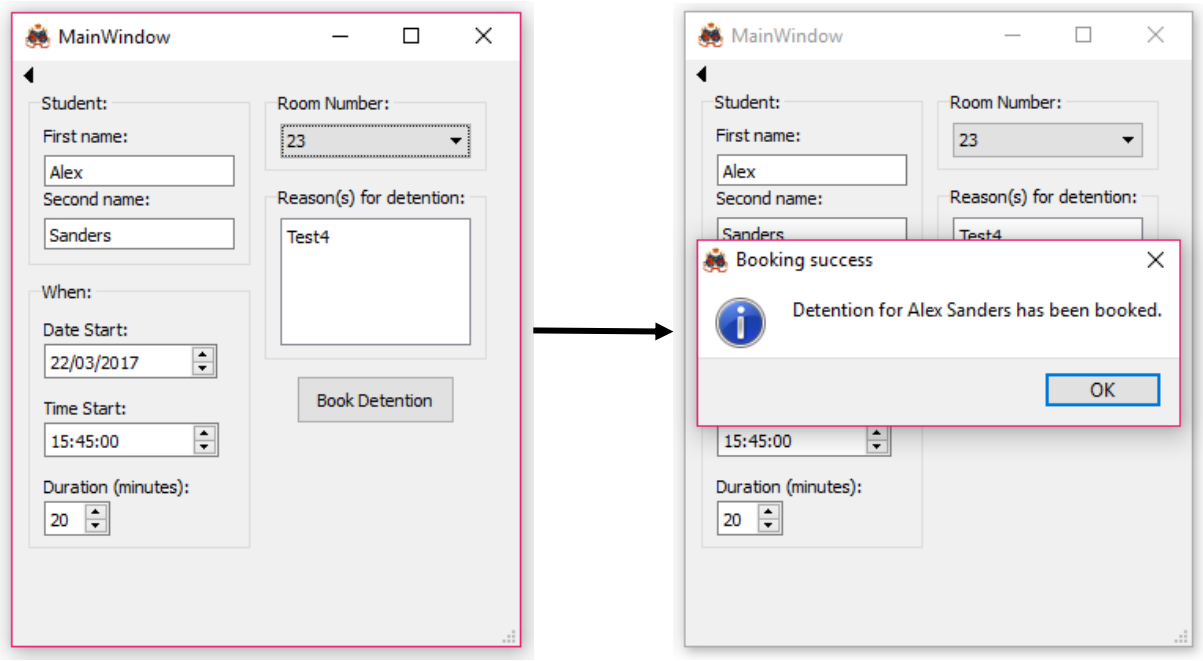
The program is able to identify here that the room is in use already during this time period (see existing detention record (DetentionID = 1). It sees they collide, despite it being with a different student.

Creating DetentionID 6



As seen here, the detention is rejected due to the student already being in detention in the existing detention record 2.

Creating DetentionID 6



This detention record can be seen to be accepted, due to how it doesn't collide with any existing detention records.

Testing if the detention record was successfully created

Objective 45 is being tested here.

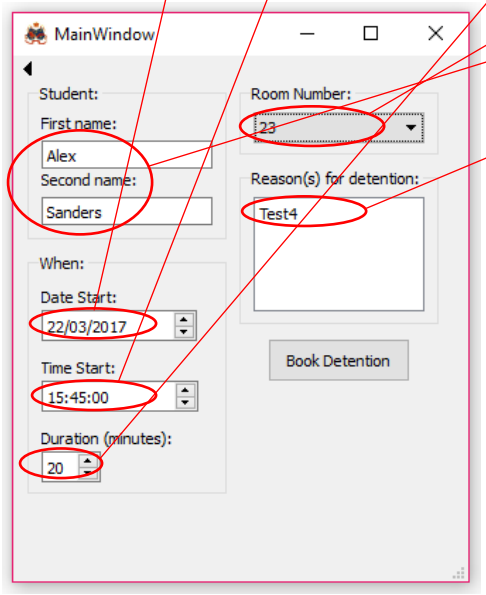
As seen in the below screenshots, the new detention record has been successfully booked, using the credentials given by the user.

Before:

DetentionID	EventDate	TimeStart	TimeEnd	Duration	RoomID	TeacherID	StudentID	Offense
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	2017-03-21	940	960	20	1	1	1	Late
2	2017-03-22	960	990	30	7	1	1	Rude

After:

DetentionID	EventDate	TimeStart	TimeEnd	Duration	RoomID	TeacherID	StudentID	Offense
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	2017-03-21	940	960	20	1	1	1	Late
2	2017-03-22	960	990	30	7	1	1	Rude
3	2017-03-22	945	965	20	8	3	2	Test4



This diagram shows how the different variables for the new record have been obtained. Some of the fields, which have not been circled, were obtained by using other methods than taking inputs from the GUI.

The DetentionID was obtained by incrementing the most recent detention record's DetentionID, the TimeEnd was obtained by adding the TimeStart (in minutes) and the duration and finally the TeacherID was obtained by using the details of the user was currently logged in when the detention was being booked.

These screenshots should serve as proof the objective 45 was achieved successfully.

Validating that the student's detention count was incremented

As stated in objective 46, my program should increment the student's detention count when a detention is successfully booked. I will show before and after screenshots of the students table in the database to show how the count has been incremented.

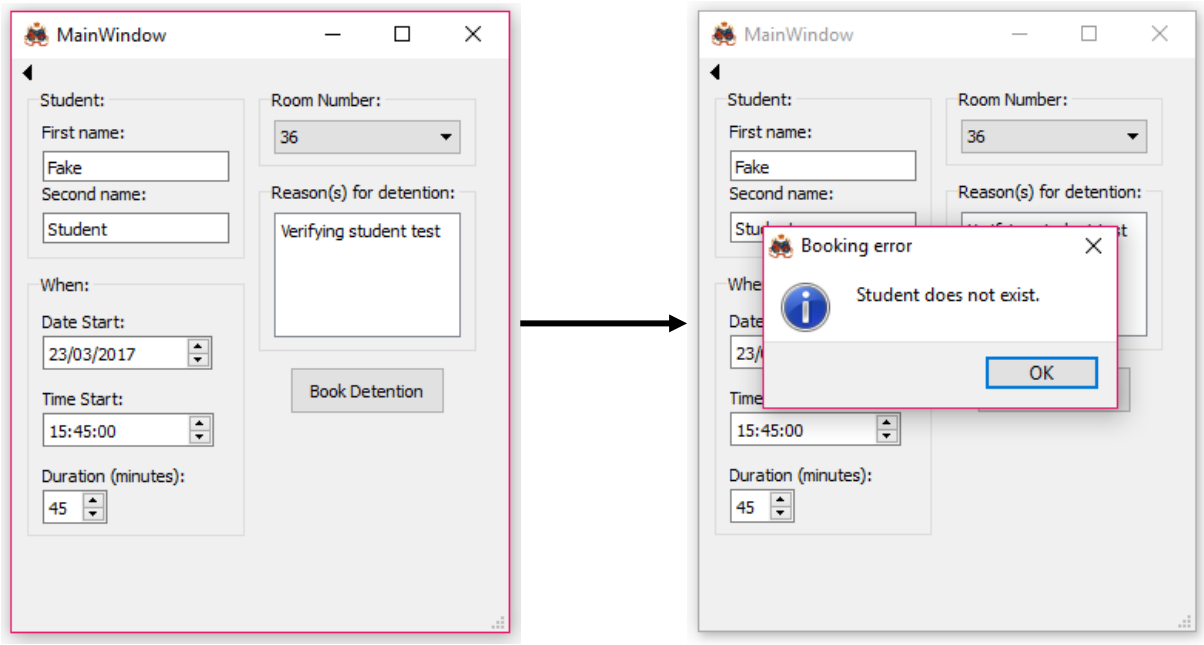
StudentID	SecondName	FirstName	CurrentYear	DetentionCount	FormID
Filter	Filter	Filter	Filter	Filter	Filter
1	Jobs	Mark	7	2	1
2	Sanders	Alex	8	0	2

StudentID	SecondName	FirstName	CurrentYear	DetentionCount	FormID
Filter	Filter	Filter	Filter	Filter	Filter
1	Jobs	Mark	7	2	1
2	Sanders	Alex	8	1	2

The DetentionCount can clearly be seen to be incremented, thus proving objective 46 has been achieved.

Testing student validation for detention booking

Although verification of the specified student's existence in the database has been considered in and catered for in my program, it is not a specific objective in my success criteria. However, I will test it here nonetheless, by using a non-existent student's name.



As seen in this screenshot, the program successfully identified that the student doesn't exist.

Objective	How it was tested	Test data used	Expected outcome	Actual outcome	Success? (yes, no, partial)
45.	Functionality to create new detention records was used.	See detention examples under “input test data” section.	Valid detentions should be booked, invalid detentions records should be rejected.	Valid detentions were booked, invalid detentions records were rejected.	Yes
46.	Functionality to create new detention records was used.	See detention examples under “input test data” section.	After a successfully booked detention, the students total detention count should be incremented.	After a successfully booked detention, the students total detention count was incremented.	Yes

This table shows the objectives which have been achieved from this section.

NOTE: the database was reset to its previous state after this record was booked.

Detention form and print windows

This window is very simple as far as the methods involved. The purpose of the class is so that a separate window may be opened, displaying the detention slip that the program will create after a detention record has been created.

Because it is a separate form, I have had to use inheritance to ensure that it obtains all of the previously stated variables from the detention booking window. This form is then displayed to the user, giving them an option to print the form, saving it as an XPS file.

However, due to the XPS format providing difficulties when I attempted to create a printout of the window, I had to create a new class, the “DetentionPrintWindow” class, to create a similar form, that would be easier to print.

As this is another separate class, I had to have inheritance of the previous “DetentionFormWindow” class. Therefore, multiple inheritance was utilised.

The reason I had to make a new class for printing the detention form will become apparent later in the tests I carried out.

Detention form class code

```
class DetentionFormWindowClass(QtGui.QMainWindow, DetentionFormWindow, DetentionBookingWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.BTN_Print.clicked.connect(self.Print1) #Button calls printing method
    def GenerateHTML(self, parent):
        DetentionFormWindow.show()
        file = open("Detention form HTML.txt", "r") #HTML file locally loaded
        HTMLfile = file.read()
        HTMLfile1 = JavaScriptInputs(HTMLfile) #Function used to give inputs into HTML file
        self.webView.setHtml(HTMLfile1) #webView used to view HTML in GUI
    def Print1(self):
        DetentionPrintWindow.Print(self) #Calls Print method from DetentionPrintWindow class
        DetentionFormWindow.hide() #Closes
```

This class is very limited due to how it’s attributes are all going to be inherited from the detention booking window. This is simply because the variables required to create the detention slip have all been identified from that window to create the detention record.

Although another approach could be to simply use various SELECT statements to obtain the relevant details from the detention record in the database. However, this would be wasteful, running various SQL queries would require processing whereas obtaining variables, by using inheritance, that have already been identified is very efficient.

Methods

Print1()

The Print1() method is called by the following line of code.

```
        self.BTN_Print.clicked.connect(self.Print1) #Button calls printing method
```

The “BTN_Print” widget will be available under the detention form that has been created, and once clicked, it will call the method. This method simply calls the “Print()” method from the detention print window class.

GenerateHTML()

The more complicated method, however, is the GenerateHTML() method.

```
        file = open("Detention form HTML.txt", "r") #HTML file locally loaded
        HTMLfile = file.read()
        HTMLfile1 = JavaScriptInputs(HTMLfile) #Function used to give inputs into HTML file
```

As seen, a file called “Detention form HTML.txt” is being read. The purpose of the HTML file is to generate a detention slip, by using HTML, CSS and JavaScript. The text file is as follows:

Detention form HTML.txt

Detention form HTML - Notepad

File Edit Format View Help

```
<style>
body{
  width:475px;
  margin:0 auto;
  margin-top: 30px;
}

p{
  font-family: "Palatino Linotype", "Book Antiqua",  Palatino, serif;
  color:black;
  text-align: left
}

.Labels{
  border-bottom-style: dotted;
  border-width: 1px;
  border-color: white;
}

.Main{
  border-bottom-style: dotted;
  border-width: 1px;
  width: 80%;
  margin-left: 10%
}

h3{
 //font-family: "palatino Linotype", "Book Antiqua", Palatino, serif;
  text-align: center
}

img{
  display: block;
  margin-left: auto;
  margin-right: auto
}

#Clarification{
  margin-left: 10%;
  margin-right: 10%;
}
</style>

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>RGS Detention Form</title>
</head>
<body>
  <img src="C:\Users\ryanb\Desktop\Coursework\Project\RGS outline logo cropped.png" style="align:center;">
  <h3>Royal Grammer School<br /> Detention Notification</h3>

  <p class="Main"id="StudentName"><span class="Labels">To parent of: </span></P>
  <p class="Main"id="Form"><span class="Labels">Form: </span></P>
  <p class="Main"id="EventDate"><span class="Labels">Date: </span></P>
  <p class="Main"id="TimeStart"><span class="Labels">Time start: </span></P>
  <p class="Main"id="TimeEnd"><span class="Labels">Time end: </span></P>
  <p class="Main"id="Location"><span class="Labels">Location of detention: </span></P>
  <p class="Main"id="Teacher"><span class="Labels">Detention issued by: </span></P>
  <p class="Main"id="Reason"><span class="Labels">Reason for detention: </span></P>
  <p class="Main"><span class="Labels">Signature of guardian: </span></P>

  <p id="Clarification"><br />This chit should be given to the student, and returned, signed, to the <i>issuing member of staff</i> at the time of the detention.</p>

</body>
</html>

<script>
var StudentName = "#StudentName#";
document.getElementById("StudentName").innerHTML += StudentName;

var Form = "#Form#";
document.getElementById("Form").innerHTML += Form;

var EventDate = "#EventDate#";
document.getElementById("EventDate").innerHTML += EventDate;

var TimeStart = "#TimeStart#";
document.getElementById("TimeStart").innerHTML += TimeStart;

var TimeEnd = "#TimeEnd#";
document.getElementById("TimeEnd").innerHTML += TimeEnd;

var Location = "#Location#";
document.getElementById("Location").innerHTML += Location;

var Teacher = "#Teacher#";
document.getElementById("Teacher").innerHTML += Teacher;

var Reason = "#Reason#";
document.getElementById("Reason").innerHTML += Reason;
</script>
```

After this text file has been read, it is assigned to a variable “HTMLfile”. This is where the JavaScriptInputs() function is to be utilised.

JavaScriptInputs()

```
def JavaScriptInputs(HTMLfile): #Function used to give inputs to JavaScript in a locally saved HTML file
    #file = open("Detention form HTML.txt", "r")
    #HTMLfile = file.read()
    HTMLfile = HTMLfile.replace("#StudentName#", DetentionBookingWindow.StudentName)
    HTMLfile = HTMLfile.replace("#Form#", DetentionBookingWindow.Form)
    HTMLfile = HTMLfile.replace("#EventDate#", DetentionBookingWindow.SlipDate)
    HTMLfile = HTMLfile.replace("#TimeStart#", DetentionBookingWindow.TimeStart)
    HTMLfile = HTMLfile.replace("#TimeEnd#", DetentionBookingWindow.TimeEnd)
    HTMLfile = HTMLfile.replace("#Location#", DetentionBookingWindow.Location)
    HTMLfile = HTMLfile.replace("#Teacher#", DetentionBookingWindow.TeacherName)
    HTMLfile = HTMLfile.replace("#Reason#", DetentionBookingWindow.Offense) #Inputs variables into script for HTML
    return HTMLfile
```

The purpose of this function is to take a parameter, which is a string, and then to replace certain keywords with various items. The keywords can be seen in this function by the green text.

The keywords can be seen here in the script tags of this HTML file.

```
<script>
var StudentName = "#StudentName#";
document.getElementById("StudentName").innerHTML += StudentName;

var Form = "#Form#";
document.getElementById("Form").innerHTML += Form;

var EventDate = "#EventDate#";
document.getElementById("EventDate").innerHTML += EventDate;

var TimeStart = "#TimeStart#";
document.getElementById("TimeStart").innerHTML += TimeStart;

var TimeEnd = "#TimeEnd#";
document.getElementById("TimeEnd").innerHTML += TimeEnd;

var Location = "#Location#";
document.getElementById("Location").innerHTML += Location;

var Teacher = "#Teacher#";
document.getElementById("Teacher").innerHTML += Teacher;

var Reason = "#Reason#";
document.getElementById("Reason").innerHTML += Reason;
</script>
```

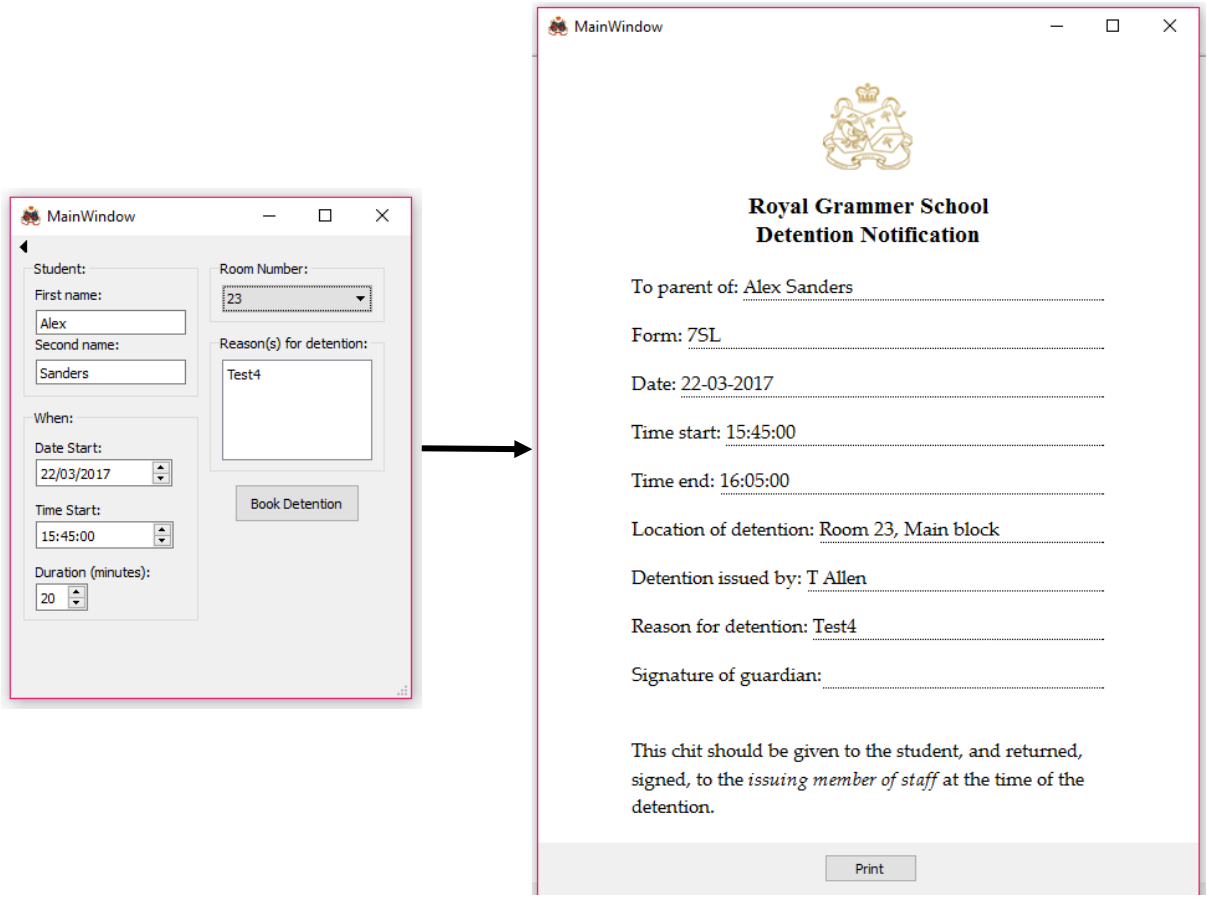
These items will be replaced by the relevant inherited attributes from the detention booking class, as denoted by DetentionBookingWindow.(attribute)

Once this has been carried out, essentially putting inputs into the HTML file. Another variable “HTMLfile1”, is the assigned to the new string that was returned by the JavaScriptInputs() function.

Then, by using the WebView widget within the window, I am able to load the HTML locally, meaning no internet connection is required. The window will then display the generated HTML.

Testing: detention form generation

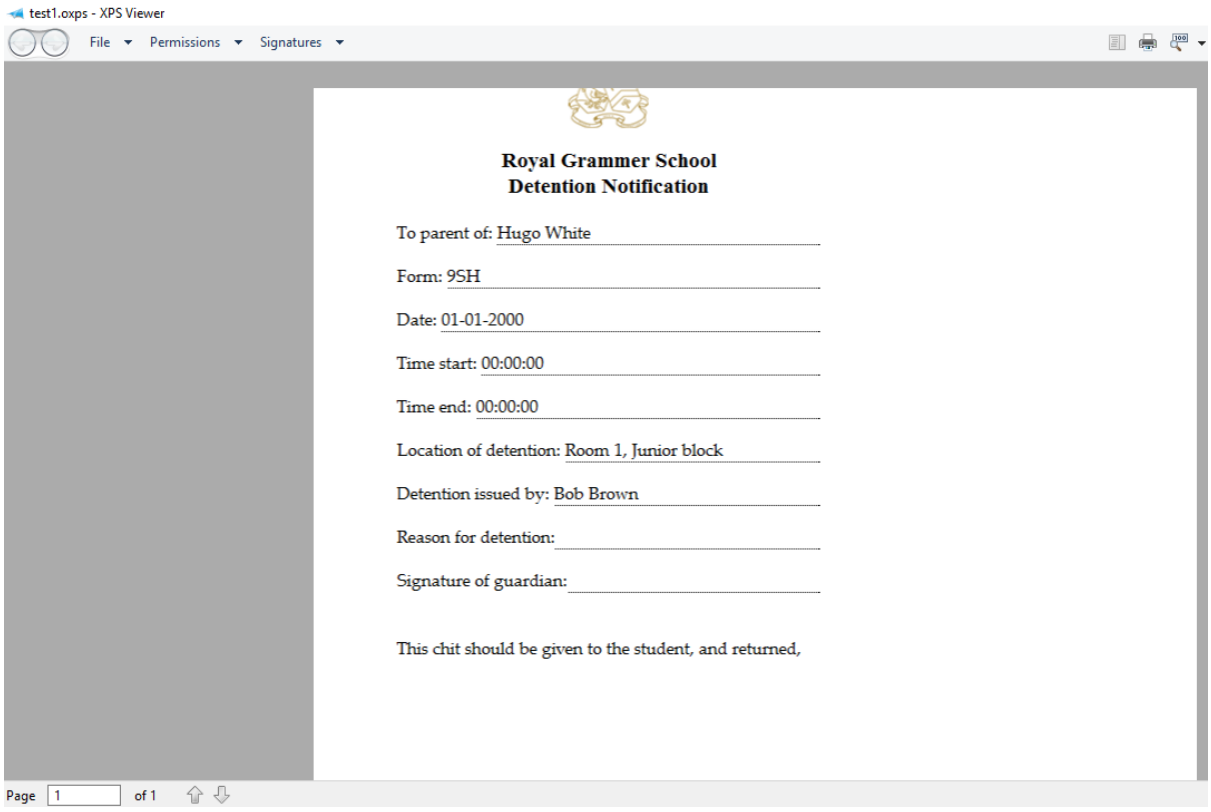
Creating a detention form

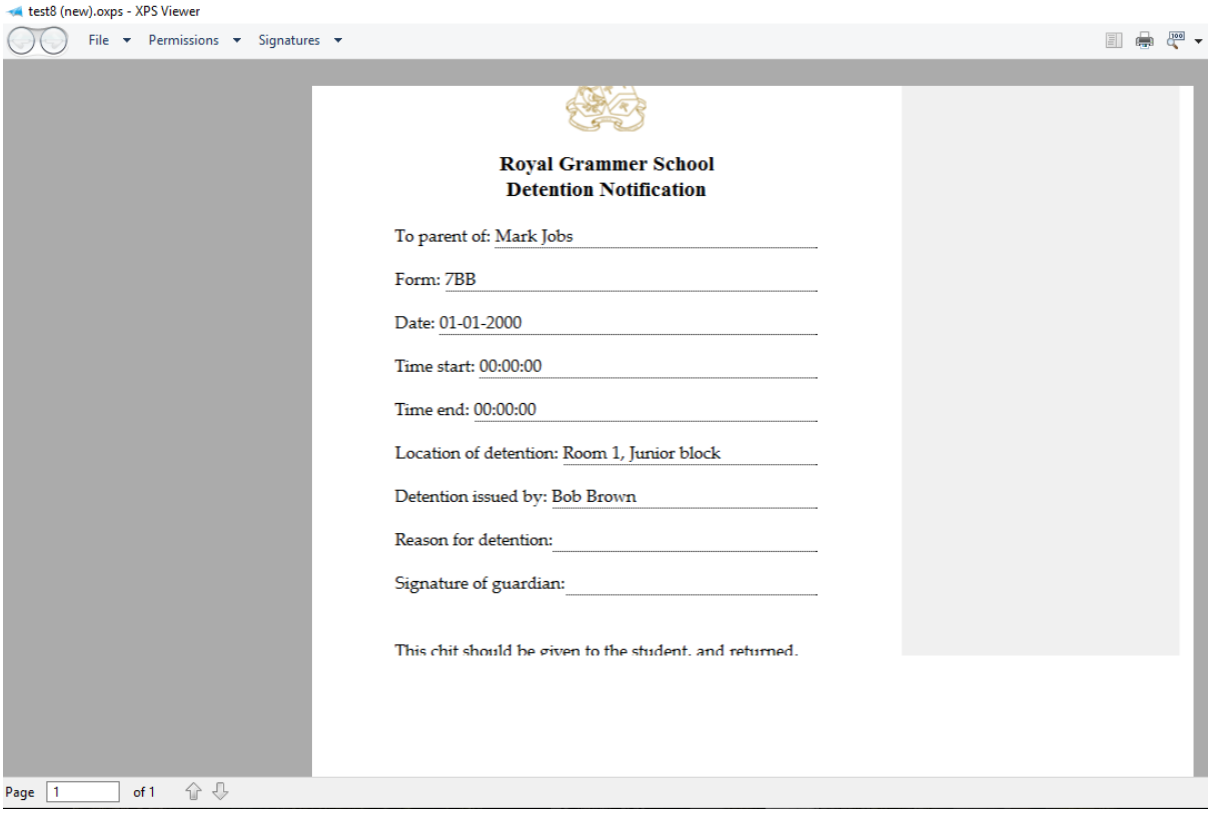


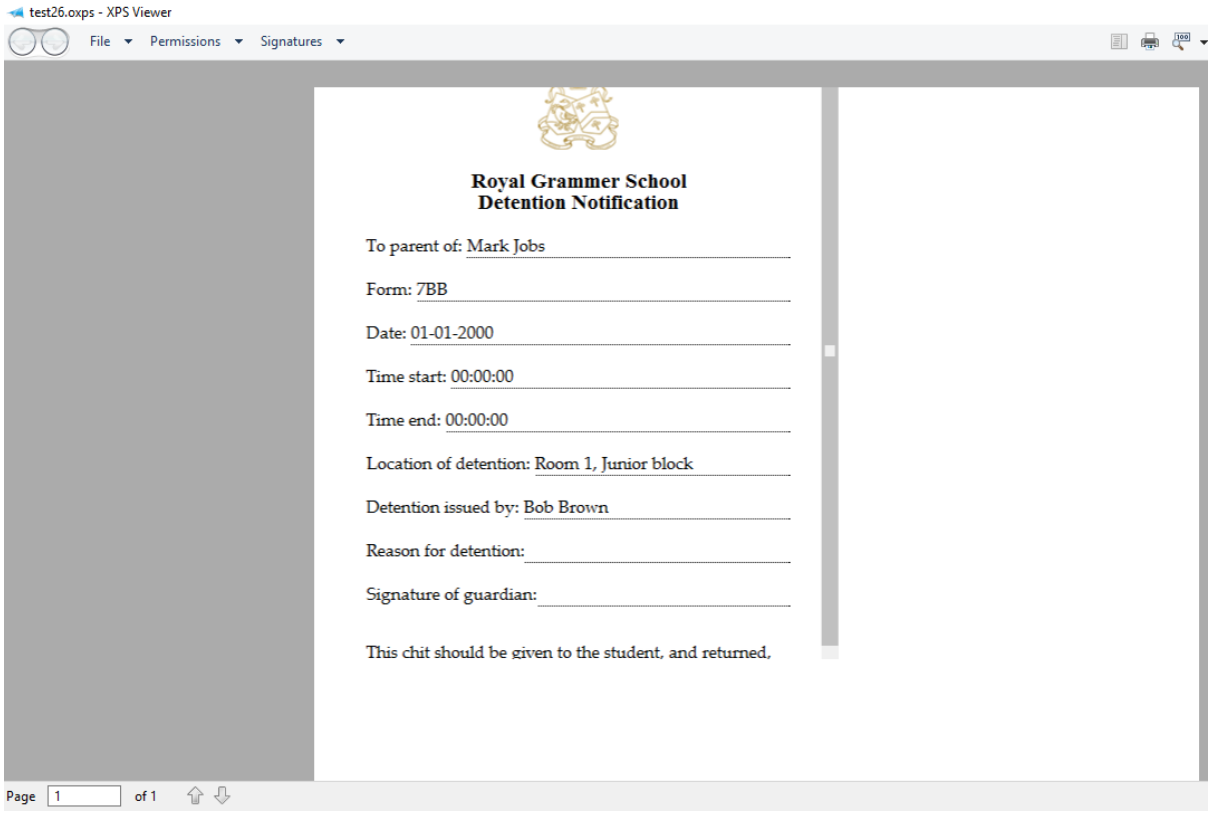
As seen here, the detention slip was created as show, thus showing the objectives 40 and 49 have been achieved.

Attempting to save detention form as an XML file

Initially, I had attempted to include the functionality for saving the output as an XML file within this window. However, as seen in the various tests below, there were various formatting issues that led to different parts of the form being cut off, creating an unsatisfactory slip.







As seen in all these different tests, the form was not created as intended, and therefore I had to come up with an alternate solution which would allow me to fit the necessary parts of the slip on, without cutting them off.

My solution, as mentioned earlier, was to create a new class, using multiple inheritance, to create another slip altogether. This slip would hold all of the details relevant to the detention, as well as keeping the RGS logo at the top. However, it would not contain the instructions for the detention slip on the printout version.

Detention print class

```
class DetentionPrintWindowClass(QtGui.QMainWindow, DetentionPrintWindow, DetentionFormWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
    def Print(self, parent):
        file = open("Detention form HTML2.txt", "r") #A new HTML file is used to create a printout version
        HTMLfile = file.read()
        HTMLfile2 = JavaScriptInputs(HTMLfile)
        self.webView.setHtml(HTMLfile2) #webView again used to view HTML in GUI
        ##########################
        printer=QtGui.QPrinter() #Creates printout
        #printer.setResolution(300)
        dialog = QtGui.QPrintDialog(printer, self)
        if(dialog.exec_() != QtGui.QDialog.Accepted):
            return
        p=QtGui.QPixmap.grabWidget(self.webView) #Loading the specific widget
        printLabel = QtGui.QLabel()
        printLabel.setPixmap(p)
        painter = QtGui.QPainter(printer) #Formatting
        ####
        #rect = painter.viewport()
        #painter.setViewport(rect.x(), rect.y(), size.width(), size.height())
        #painter.setWindow(-125, -125, 500, 500)
        ####
        printLabel.render(painter)
        painter.end()
```

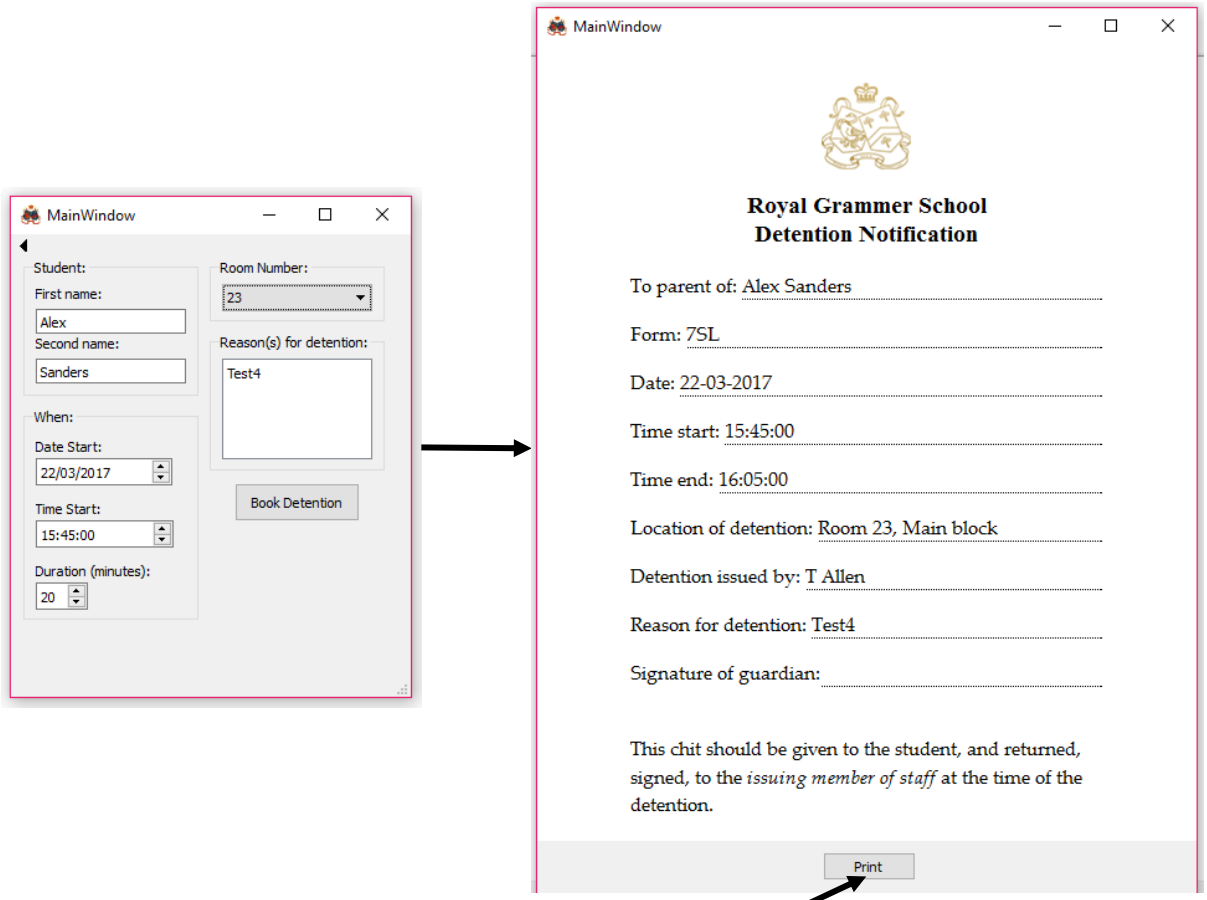
The above class will inherit all of its attributes from the previous detention form class. Much like the GenerateHTML() method used in the previous class, this code will locally load a HTML file for the sake of replacing text within it, and then locally loading it into the GUI.

The HTML file it uses, however, is different. The main difference between “Detention form HTML2.txt” and “Detention form HTML.txt” is that the sentence present at the bottom of the file has been removed. As well as this, the size of the vertical margin has been reduced, to help it fit into the XML file. Other than that, there are no changes present.

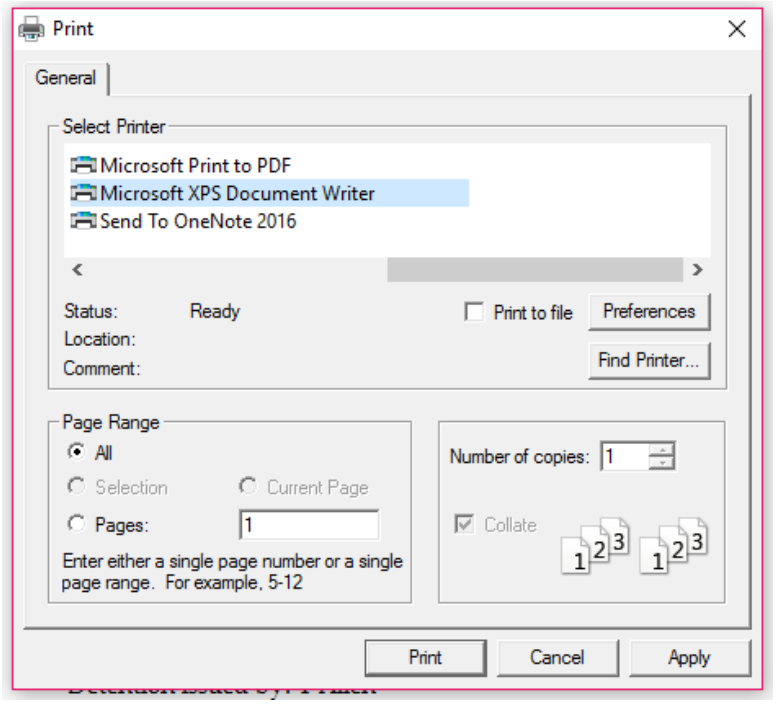
The rest of the code is simply used to load the file into the WebView widget of the GUI, and then various inbuilt print functions within PyQt are used to create the printout. The user will then have the option to save the file, as an XPS document, or to print it directly.

To test this functionality, I will press the “print” button on the previous detention form window. I will then show me saving this detention form, and attempting to test objectives 51, 52 and 53 using the created XML file.

Testing detention print window

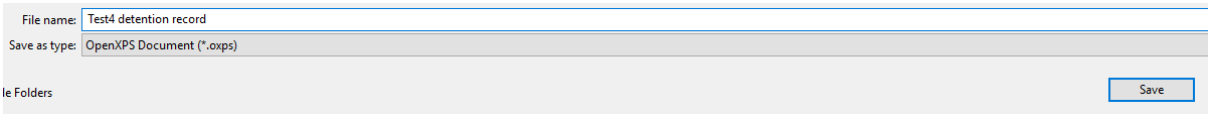


After booking the detention, I then pressed the print button. This created the following pop-up:

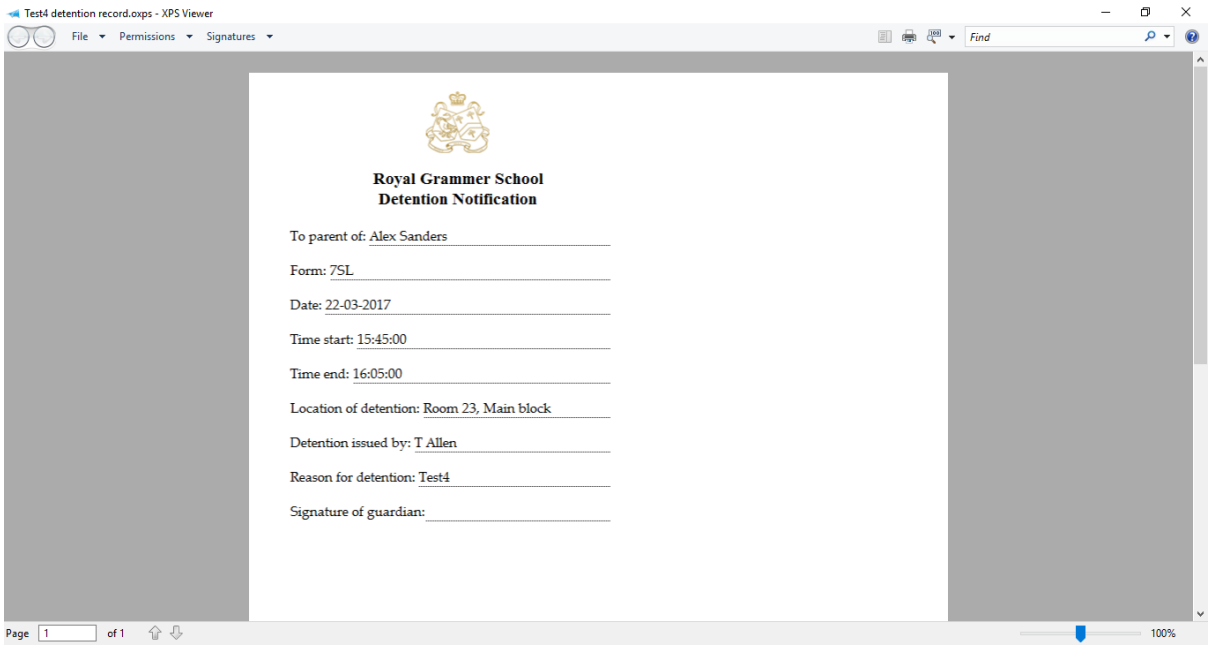


Saving an XPS file

From here, I selected “Microsoft XPS Document Writer” and pressed print. I then specified the path of the file as follows: C:\Users\ryanb\Desktop\Coursework\Project\Test printouts\Test4 detention record.oxps



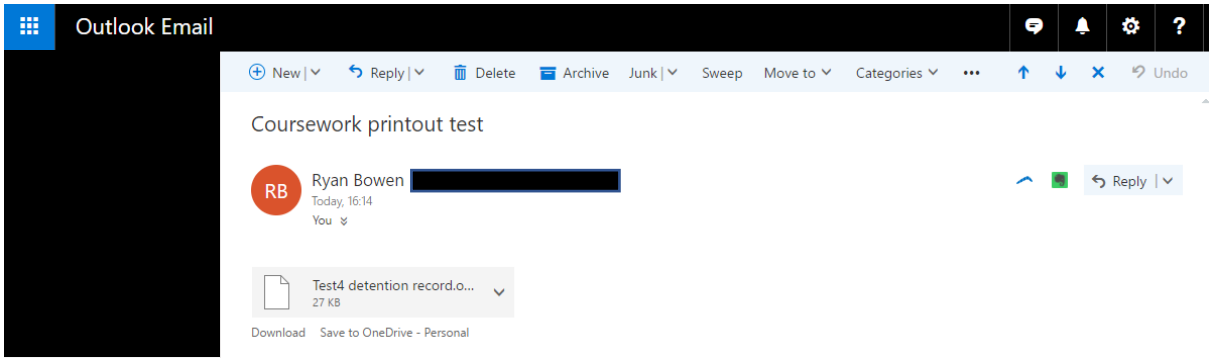
When I opened this file, the XML file was shown as follows:



As seen, I was able to successfully crop the slip by changing the HTML contents. This produced the desired slip, without cropping out any important information, or damaging the aesthetics of the slip by cropping the RGS logo.

This shows that objective 51 has been achieved, by successfully saving the detention form as an XML file. In order to test objectives 52 and 53, I will simply have to show that the file can be emailed, and then be viewed as expected on separate devices, or simply printed out and a physical copy of the version above may be seen.

Sending the XML file as an email attachment

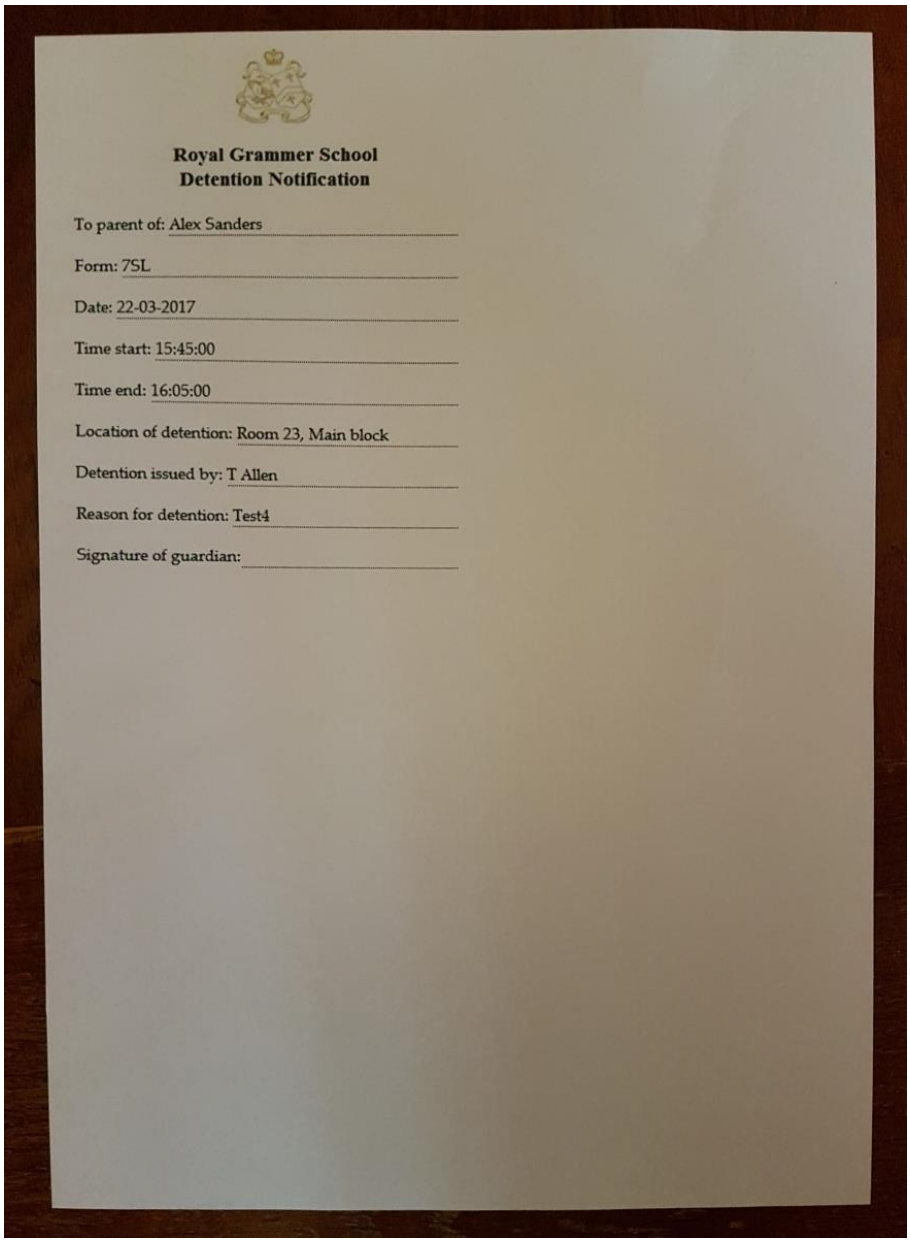


The above screenshot shows the email that was sent containing the attached file.

And when opened, appeared as seen in the above screenshot. I can therefore conclude that objective 52 was achieved as intended.

Printing the XML file

Finally, I will show the outcome of attempting to print the prior mentioned XML file.



Royal Grammer School
Detention Notification

To parent of: Alex Sanders

Form: 7SL

Date: 22-03-2017

Time start: 15:45:00

Time end: 16:05:00

Location of detention: Room 23, Main block

Detention issued by: T Allen

Reason for detention: Test4

Signature of guardian:

As seen here, the XML file was successfully printed, thus showing that objective 53 has been achieved.

Objective	How it was tested	Test data used	Expected outcome	Actual outcome	Success? (yes, no, partial)
40,49.	Function used to book a detention, and the output is observed.	See test detention bookings table, under “input test data”.	If detention is successful, it will create a HTML form with the credentials of the detention. This should appear similar to the RGS form.	Form was created using HTML, using the credentials of the detention. It is very similar to the RGS form.	Yes
51.	Function used to save the form as XML file.	See test detention bookings table, under “input test data”.	Screenshots prove XML file was saved successfully.	Screenshots prove XML file was saved successfully.	Yes
52.	XML file was emailed, and then opened by the recipient.	See test detention bookings table, under “input test data”.	Screenshots show that the email received contained the XML file as an attachment, and was successfully opened by recipient.	Screenshots show that the email received contained the XML file as an attachment, and was successfully opened by recipient.	Yes
53.	XML file was printed out, and photos were taken of the physical printout.	See test detention bookings table, under “input test data”.	Photos prove that the XML file was able to be printed out.	Photos prove that the XML file was able to be printed out.	Yes

Detention Viewer

Detention viewer class

```
class DetentionViewerWindowClass(QtGui.QMainWindow, DetentionViewerWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        ##################
        self.TB_BACK.clicked.connect(self.BACK)
        self.BTN_ViewDatabase.clicked.connect(self.ShowDatabase)
        self.BTN_DrawGraph.clicked.connect(self.Graph)
        self.BTN_Summarize.clicked.connect(self.Summarise)
        ##################
    def BACK(self):
        DetentionViewerWindow.hide()
        MenuWindow.show()
    def ShowDatabase(self):
        ViewDatabaseWindow.show() #shows database view window
    def Graph(self): #HTML is being loaded
        file = open("Constant graphs.txt", "r") #HTML file for graphs
        HTMLgraph = file.read()
        #HTMLgraph = JavaScriptInputs2(HTMLgraph)
        self.webView.setHtml(HTMLgraph) #webView used to view HTML in GUI
    def Summarise(self):
```

This code shows an early structure of the code, with the scaffolding for the methods that will allow creation of graphs, and statistical summarisations of detentions.

At this stage, the abilities to use the back button as well as present a raw view of the database to the user were both available.

As seen in this code, there are four different buttons that call four different methods in this class.

ShowDatabase()

To reduce clutter, I decided to make the database view occur on a completely separate window, without closing the previous window (the detention viewer). This also serves important purpose of allowing the user to make reference to the database, while simultaneously being able to use the functionality of the detention viewer.

Because of this, I had to make a new class altogether:

```
class ViewDatabaseWindowClass(QtGui.QMainWindow, ViewDatabaseWindow):
    def __init__(self, parent=None):
            QtGui.QMainWindow.__init__(self, parent)
            self.setupUi(self)
            ##################
            self.data=[]
            self.model = QtGui.QStandardItemModel(self)
            self.tableView.setModel(self.model)
            self.load_data()
            ##################
    def load_data(self):
            Fetch=("""SELECT * FROM Detentions""") #all records from Detentions
            cur.execute(Fetch)
            self.data = cur.fetchall() #returns 2D tuple
            self.model=QtGui.QStandardItemModel(self)
            self.tableView.setModel(self.model)
            for row in self.data:
                        items = [
                            QtGui.QStandardItem(str(field))
                            for field in row
                        ]
                        self.model.appendRow(items)
```

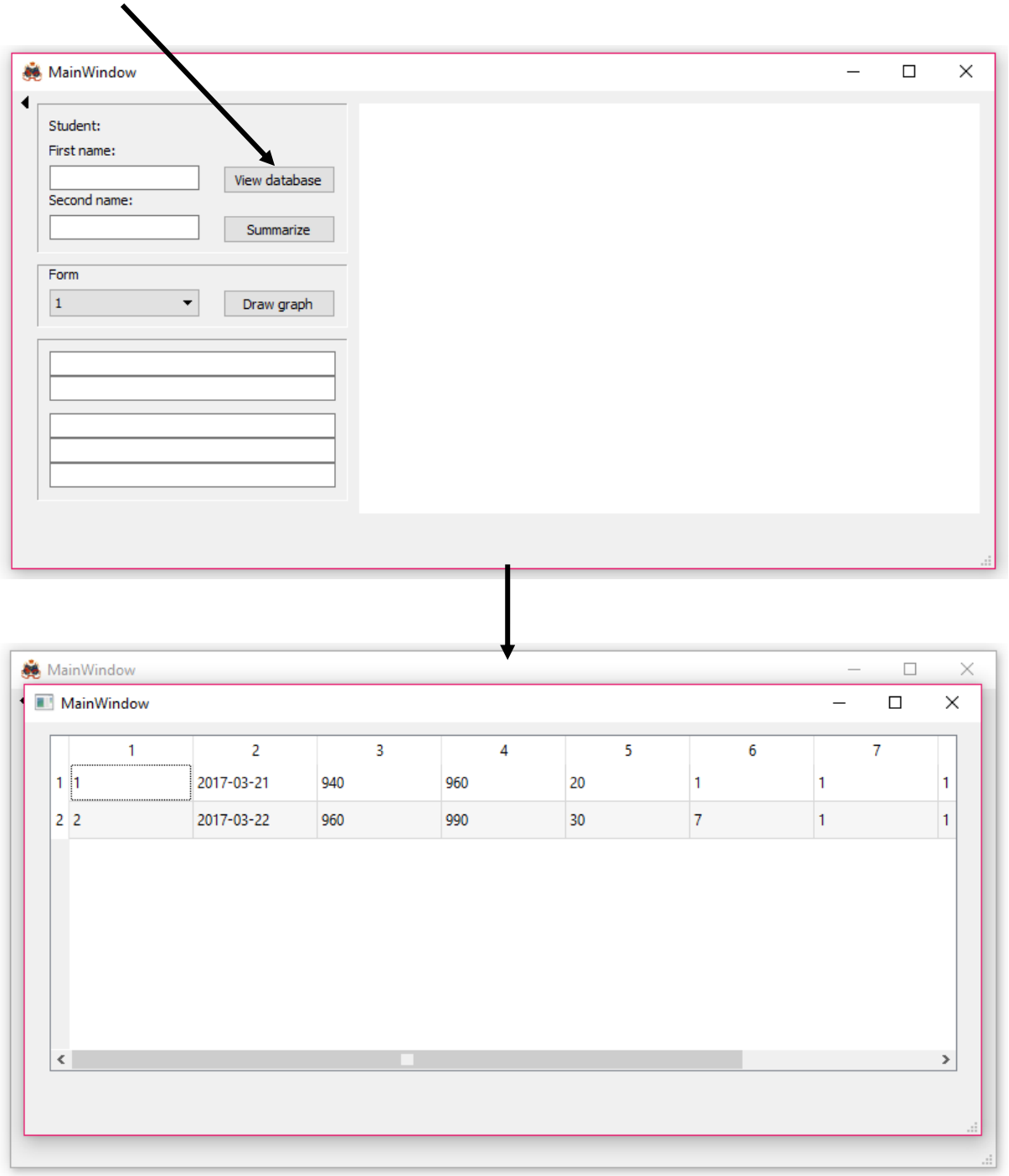
The window tied to this class is shown by pressing the “view database button” on the detention viewer window. This button calls a method, which simply executes this code:

```
    def ShowDatabase(self):
        ViewDatabaseWindow.show() #shows database view window
```

As seen, it is very simple, though it achieves the attended result, as shown in the following test.

Testing: raw database view

Due to objective 60 requiring the user to be able to view a raw version of the database in the program, I must prove that this functionality has or has not been achieved.



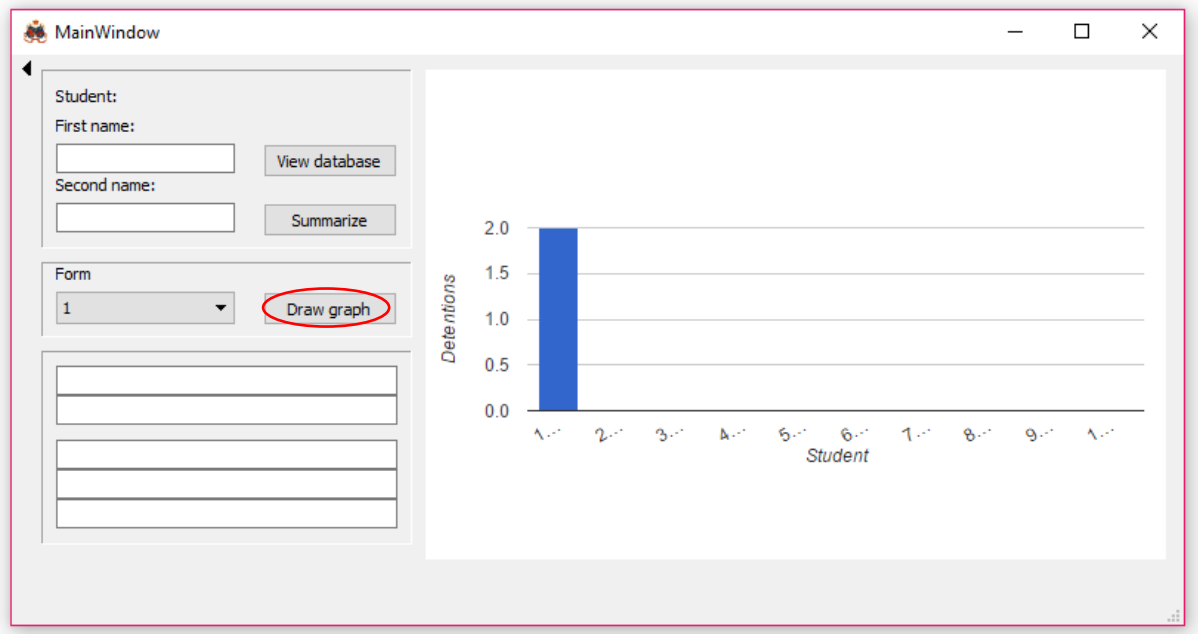
This screenshot shows how objective 60 has been achieved successfully.

Testing: graphs

Graph 1

This first test will be carried out on all students in the database, using the following dataset:

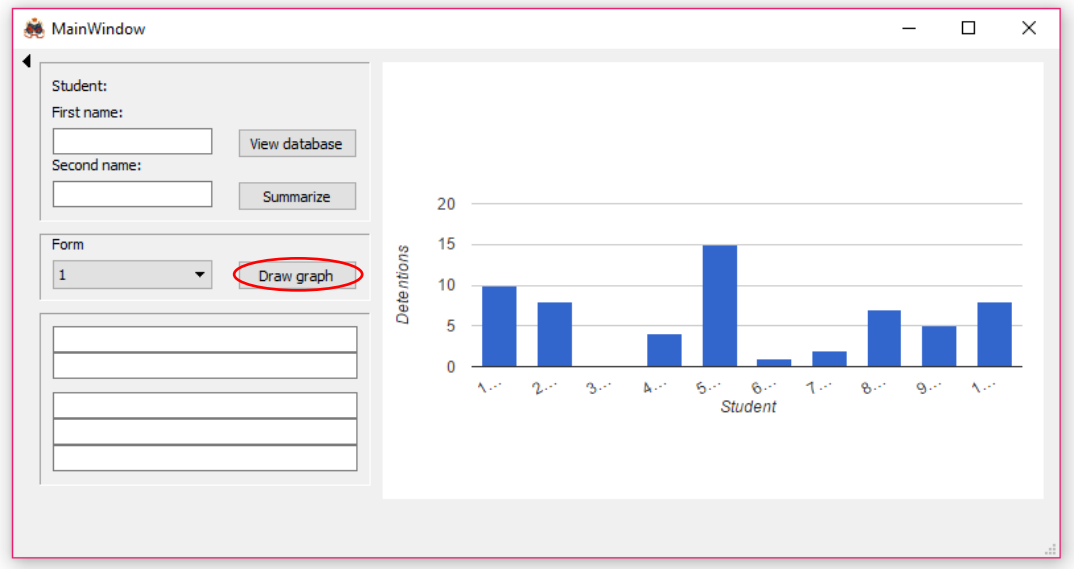
StudentID	SecondName	FirstName	CurrentYear	DetentionCount	FormID
Filter	Filter	Filter	Filter	Filter	Filter
1	Jobs	Mark	7	2	1
2	Sanders	Alex	8	0	2
3	White	Hugo	9	0	6
4	Taylor	Thomas	7	0	1
5	Harris	Matthew	10	0	8
6	Mayers	Michael	10	0	8
7	Clack	Dexter	8	0	4
8	Dixon	Daniel	10	0	7
9	Vase	Jason	8	0	3
10	Lawrance	Noah	9	0	5

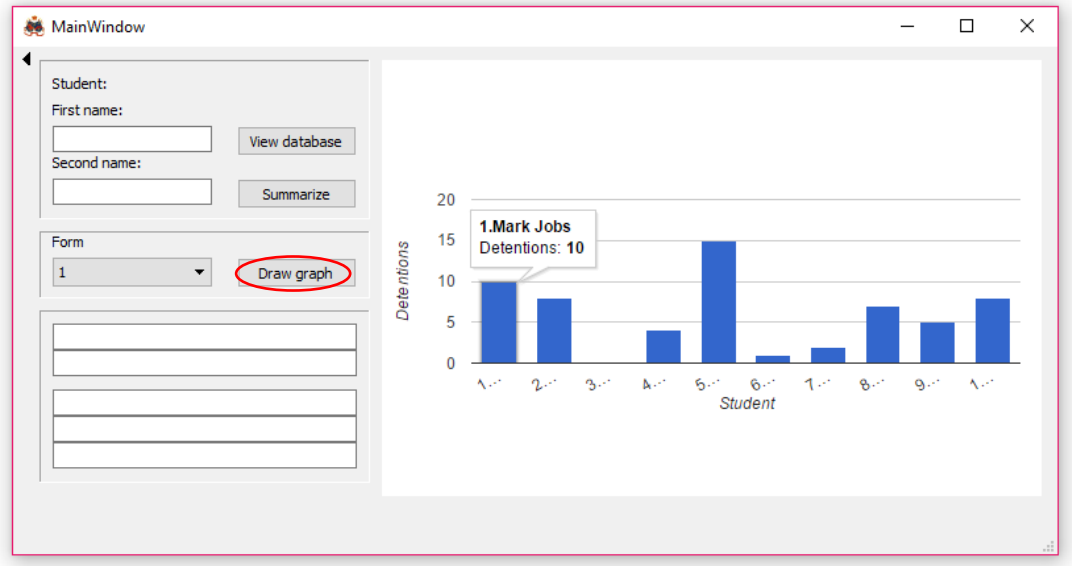


Graph 2

The second test will be carried out on all students in the database, using the following dataset:

StudentID	SecondName	FirstName	CurrentYear	DetentionCount	FormID
Filter	Filter	Filter	Filter	Filter	Filter
1	Jobs	Mark	7	10	1
2	Sanders	Alex	8	8	2
3	White	Hugo	9	0	6
4	Taylor	Thomas	7	4	1
5	Harris	Matthew	10	15	8
6	Mayers	Michael	10	1	8
7	Clack	Dexter	8	2	4
8	Dixon	Daniel	10	7	7
9	Vase	Jason	8	5	3
10	Lawrance	Noah	9	8	5





Summarise()

The Summarise() method will be used to carry out statistical analysis on the database, according to the user's selected student. It should be able to determine if the specified student exists, what their total number of detentions are, the percentage of the total number of all detentions that their amount accounts for, the mean number of detentions, the highest number of detentions, and the student who has the highest number of detentions.

The first part of this method is finding the StudentID for the user's specified student.

```
def Summarise(self):
    self.FirstName = str((self.LE_FirstName.text())) #variables obtained from GUI
    self.SecondName = str((self.LE_SecondName.text()))
    #StudentID
    #####
    CheckSelectedStudent = ("""SELECT StudentID FROM Students
    WHERE (FirstName = ?) AND (SecondName = ?)""")   #SQL statement finds StudentID where first name and second name are present
    cur.execute(CheckSelectedStudent, (self.FirstName, self.SecondName, ))
    self.StudentID = cur.fetchall()
    if len(self.StudentID) == 0: #if no StudentID's are returned, the student does not exist
        QtGui.QMessageBox.information(self, "Booking error", "Student does not exist.            ")
    else:
        self.StudentID = self.StudentID[0] #else, there is only 1 student with the name, and so a detention may be booked
        self.StudentID = module_FormattingFunctions.FetchTreater(self.StudentID) #module_FormattingFunctions.FetchTreater() used t
        print("STUDENT: "+self.StudentID)
        ######
```

It does this for two reasons, firstly because this allows the program to identify whether the specified user exists or not, and secondly, because it is unique and therefore is better served to search through the table by.

After it has verified that the student exists, a very simple SELECT statement is used to find the total number of detentions for all the students in the database.

```
#TotalDetentions
AllStudentDetentions=("""SELECT FirstName, SecondName, DetentionCount FROM Students""")
cur.execute(AllStudentDetentions)
StudentsList = cur.fetchall() #returns students
TotalDetentions = 0
for Student in StudentsList:
    TotalDetentions = TotalDetentions + int(Student[2]) #totals student's detentions
```

It does this by returning a 2D array which contains all the students from the students table. The program then increments through each student array, adding the amount of detentions for each student to the TotalDetentions variable.

The next stage of this method is to find the number of detentions for the specified student. This is also a relatively simple SQL SELECT query.

```
#StudentsDetentions
DetentionCount=("""SELECT DetentionCount FROM Students
WHERE StudentID = ?""")
cur.execute(DetentionCount, (self.StudentID, ))
StudentsDetentions = int(module_FormattingFunctions.FetchTreater(cur.fetchall()[0])) #detentions for specified student
self.LE_StudentsDetentions.setText(self.FirstName+" "+self.SecondName+" has "+str(StudentsDetentions)+" detentions,")
```

A line edit is used at this stage, displaying a sentence to the user which says the amount of detentions for their specified student.

After this, a few lines of code work out the percentage of total detentions which this student is accountable for. Another line edit displays this figure to the user.

```
#PercentageDetentions
PercentageDetentions = int((StudentsDetentions/TotalDetentions)*100)
PercentageDetentions = str(PercentageDetentions)+"%"
self.LE_Percentage.setText("which accounts for "+PercentageDetentions+" of all detentions.")
```

It does this by simply dividing their detentions by the total, and then multiplying it by 100.

Next, the mean number of detentions for the students is worked out.

```
#MeanDetentions
MeanDetentions = TotalDetentions/len(StudentsList) #average detentions
self.LE_MeanDetentions.setText("The current mean is "+str(MeanDetentions)+" detentions,") #pop-up
```

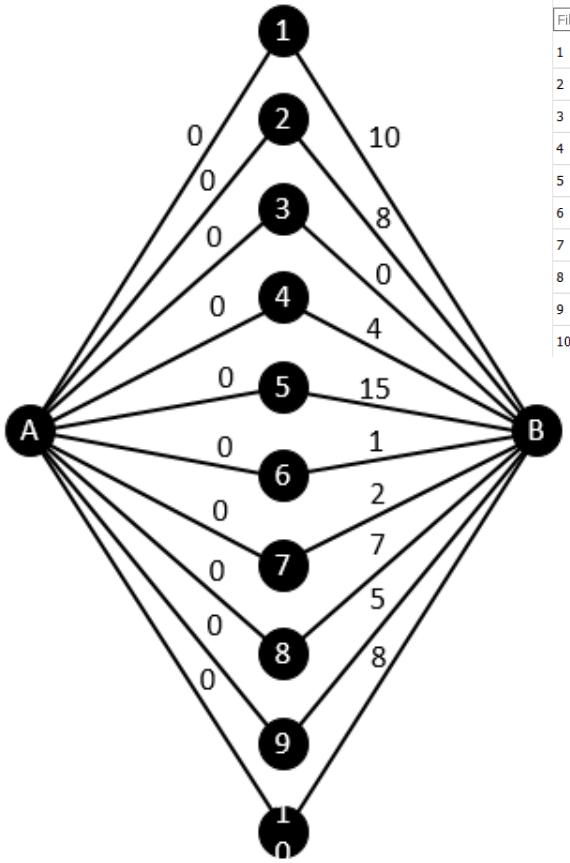
It does this by dividing the total number of detentions by the number of students. A line edit is also used here to notify the user of this statistic.

The final stage of this method involves identifying which student holds the highest number of detentions, and what this number is.

It does this by using the general principles behind Dijkstra's algorithm, but instead of finding the shortest path, it finds the longest path. The code used may be seen here:

```
#HighestDetentions
HighestDetentions = 0
for Student in StudentsList:
    if Student[2] > HighestDetentions: #next student have more detentions?
        HighestDetentions = int(Student[2])
        HighestStudent = (Student[0])[0]+"."+Student[1] #name of HighestStudent
    else:
        pass
self.LE_MostDetentions.setText(HighestStudent+"'s "+str(HighestDetentions)+" detentions being the highest.")
```

The method of setting "HighestDetentions" equal to 0 initially is based on how Dijkstra's algorithm sets the shortest distance to each node to infinity initially. The reason I use 0 instead of infinity however, is because I am attempting to find the largest path, as opposed to the shortest path. The reason I can use principles of Dijkstra's algorithm is due to how I can view my database as a weighted graph data structure.



StudentID	SecondName	FirstName	CurrentYear	DetentionCount	FormID
Filter	Filter	Filter	Filter	Filter	Filter
1	Jobs	Mark	7	10	1
2	Sanders	Alex	8	8	2
3	White	Hugo	9	0	6
4	Taylor	Thomas	7	4	1
5	Harris	Matthew	10	15	8
6	Mayers	Michael	10	1	8
7	Clack	Dexter	8	2	4
8	Dixon	Daniel	10	7	7
9	Vase	Jason	8	5	3
10	Lawrance	Noah	9	8	5

When you view the data structure as a graph, with nodes A and B being connected by nodes 1-10 (representing the StudentID's of 10 different students), with the weight of their paths representing the DetentionCount of the student.

Then, by using an algorithm to work out the longest path, as opposed to the shortest path, you could easily identify the student with the highest number of detentions.

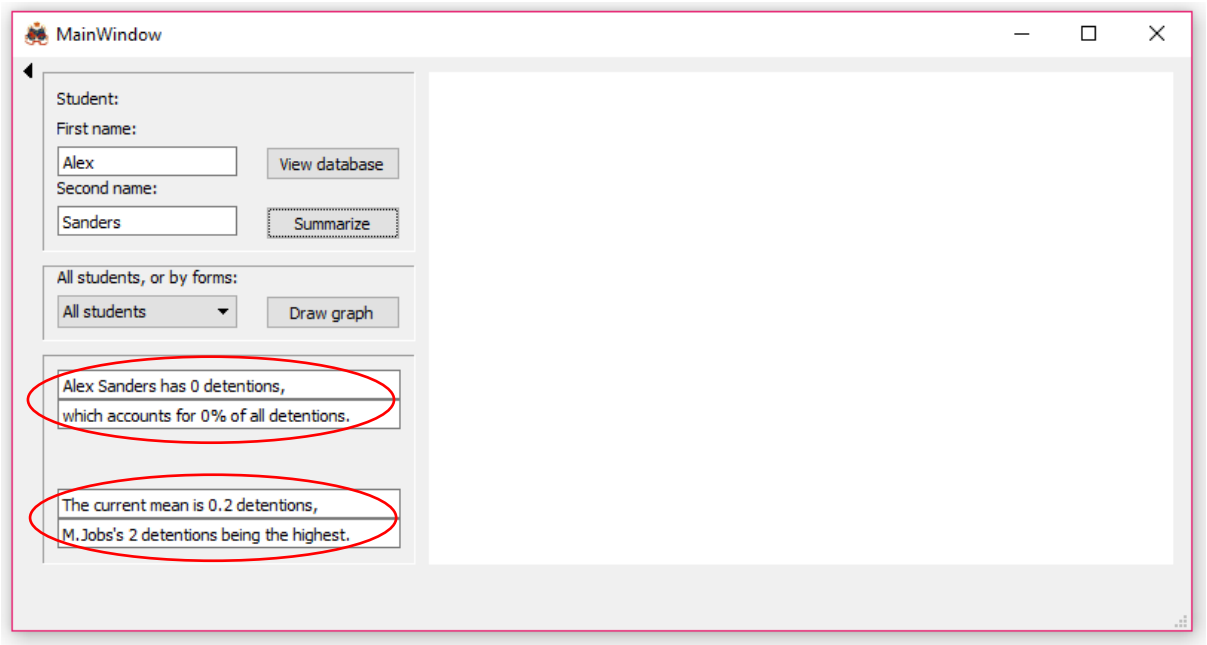
Testing: Summarise()

Now that I have demonstrated the code that makes this method work, I can show how it works using different data sets.

1st test

Using the following dataset, I will run the "Summarise()" method, and display the output.

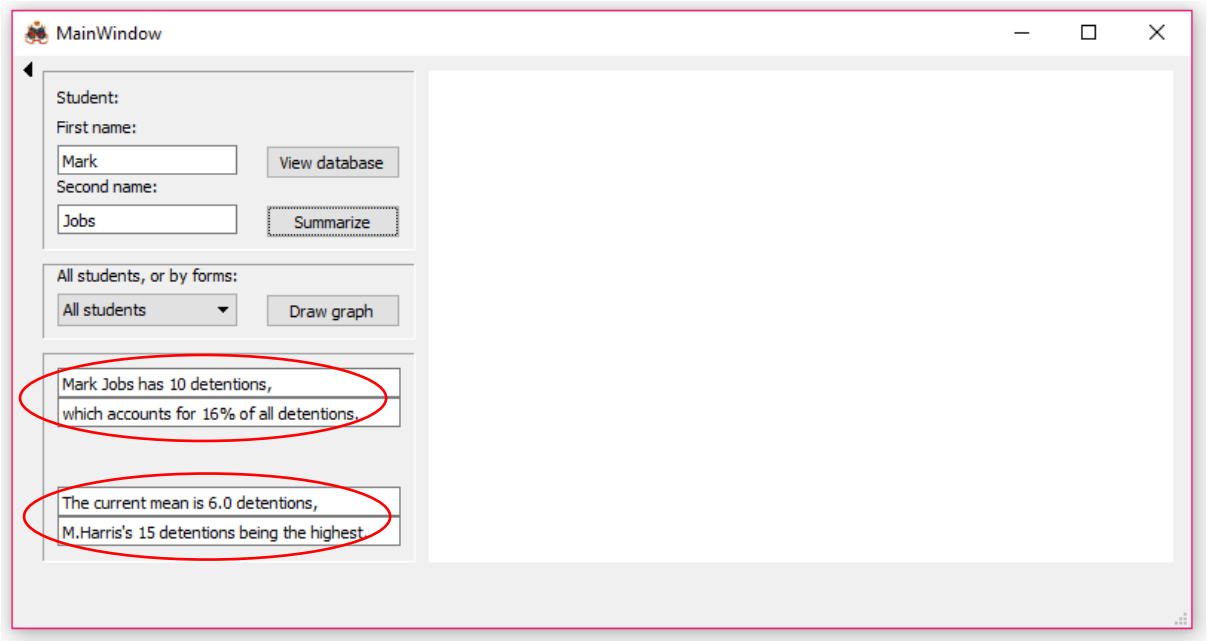
StudentID	SecondName	FirstName	CurrentYear	DetentionCount	FormID
Filter	Filter	Filter	Filter	Filter	Filter
1	Jobs	Mark	7	2	1
2	Sanders	Alex	8	0	2
3	White	Hugo	9	0	6
4	Taylor	Thomas	7	0	1
5	Harris	Matthew	10	0	8
6	Mayers	Michael	10	0	8
7	Clack	Dexter	8	0	4
8	Dixon	Daniel	10	0	7
9	Vase	Jason	8	0	3
10	Lawrance	Noah	9	0	5



2nd test

Using the following dataset, I will run the "Summarise()" method, and display the output.

StudentID	SecondName	FirstName	CurrentYear	DetentionCount	FormID
Filter	Filter	Filter	Filter	Filter	Filter
1	Jobs	Mark	7	10	1
2	Sanders	Alex	8	8	2
3	White	Hugo	9	0	6
4	Taylor	Thomas	7	4	1
5	Harris	Matthew	10	15	8
6	Mayers	Michael	10	1	8
7	Clack	Dexter	8	2	4
8	Dixon	Daniel	10	7	7
9	Vase	Jason	8	5	3
10	Lawrance	Noah	9	8	5



As seen in the above tests, the program could successfully carry out statistical analysis on the given data set.

Therefore, the final objectives have been addressed which are related to this window.

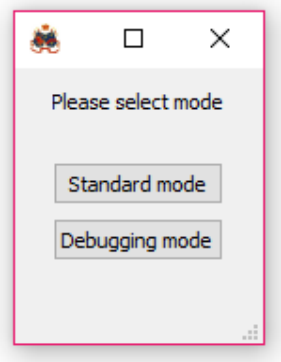
Objective	How it was tested	Test data used	Expected outcome	Actual outcome	Success? (yes, no, partial)
54-55.	Use of graph functionality.	See existing detention records, under "existing test data".	Graph created using HTML, CSS and JavaScript to show trends in student's detentions.	Graph created using HTML, CSS and JavaScript to show trends in student's detentions.	Yes
57-59.	Statistical analysis functionality used and output observed.	See existing detention records, under "existing test data".	Various outputs able to be produced from existing detention records.	Various outputs were produced from existing detention records.	Yes
60.	Raw view of database functionality was tested, and the output was observed.	See all tables under "existing test data".	Raw view of the database was available from the Detention-Viewer window.	Raw view of the database was available from the Detention-Viewer window.	Yes

Final version of program

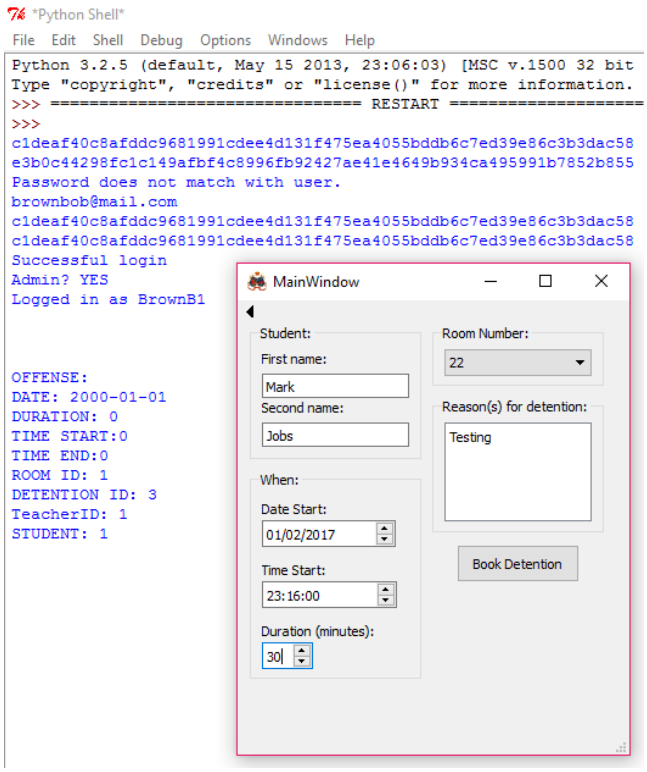
After creating all of these features in order to achieve my success criteria, I decided to improve my program by adding features such as a debugging mode, as well as finally adding a feature to determine if the necessary UI files are present or not (which is objective 1 in my success criteria).

Debugging mode

This was very simple to make, as I all I had to do was create a new window which would open as soon as the user ran the program. It would then prompt the user to choose which mode to run the program in. The standard mode would run the program as normal, however, the debugging mode would print out the contents of various variables relevant to the current processes. This would help identify to the user what is going on at a certain time, in the event that they are trying to identify an issue with the program.



I proceeded to use the program as normal to evidence how the program would output relevant variables.



Checking if resources are available

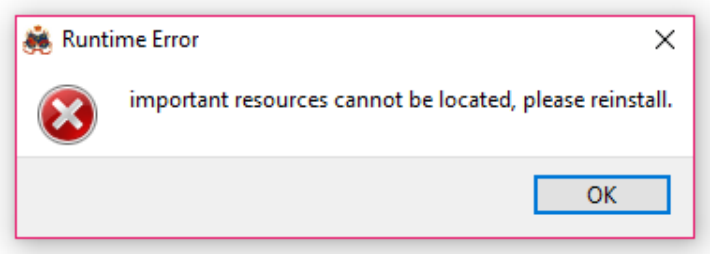
```
try:
    Debugging = uic.loadUiType("Debugging.ui")[0]
    LoginWindow = uic.loadUiType("Login.ui")[0]
    MenuWindow = uic.loadUiType("Menu.ui")[0]
    DetentionBookingWindow = uic.loadUiType("NewDetentionBooker.ui")[0]
    ChangePasswordWindow = uic.loadUiType("ChangePassword.ui")[0]
    ManageUsersWindow = uic.loadUiType("ManageUsers.ui")[0]
    CreateUserWindow = uic.loadUiType("CreateUser.ui")[0]
    DetentionFormWindow = uic.loadUiType("DetentionForm.ui")[0]
    DetentionPrintWindow = uic.loadUiType("DetentionFormPrint.ui")[0]
    DetentionViewerWindow = uic.loadUiType("DetentionViewer2.ui")[0]
    ViewDatabaseWindow = uic.loadUiType("ViewDatabase.ui")[0]
except:
    LoadingFiles = "False"
```

Normally, the UI files will appear with the following names, and if they are successfully loaded, then the line “LoadingFiles = ‘False’” should never be executed.

To ensure my program can identify when the files cannot be located, I have changed the above code to the following:

```
try:
    Debugging = uic.loadUiType("12345")[0]
    LoginWindow = uic.loadUiType("12345")[0]
    MenuWindow = uic.loadUiType("12345")[0]
    DetentionBookingWindow = uic.loadUiType("12345")[0]
    ChangePasswordWindow = uic.loadUiType("12345")[0]
    ManageUsersWindow = uic.loadUiType("12345")[0]
    CreateUserWindow = uic.loadUiType("12345")[0]
    DetentionFormWindow = uic.loadUiType("12345")[0]
    DetentionPrintWindow = uic.loadUiType("12345")[0]
    DetentionViewerWindow = uic.loadUiType("12345")[0]
    ViewDatabaseWindow = uic.loadUiType("12345")[0]
except:
    LoadingFiles = "False"
```

The program should then be able to identify to the user that files have not been located properly.



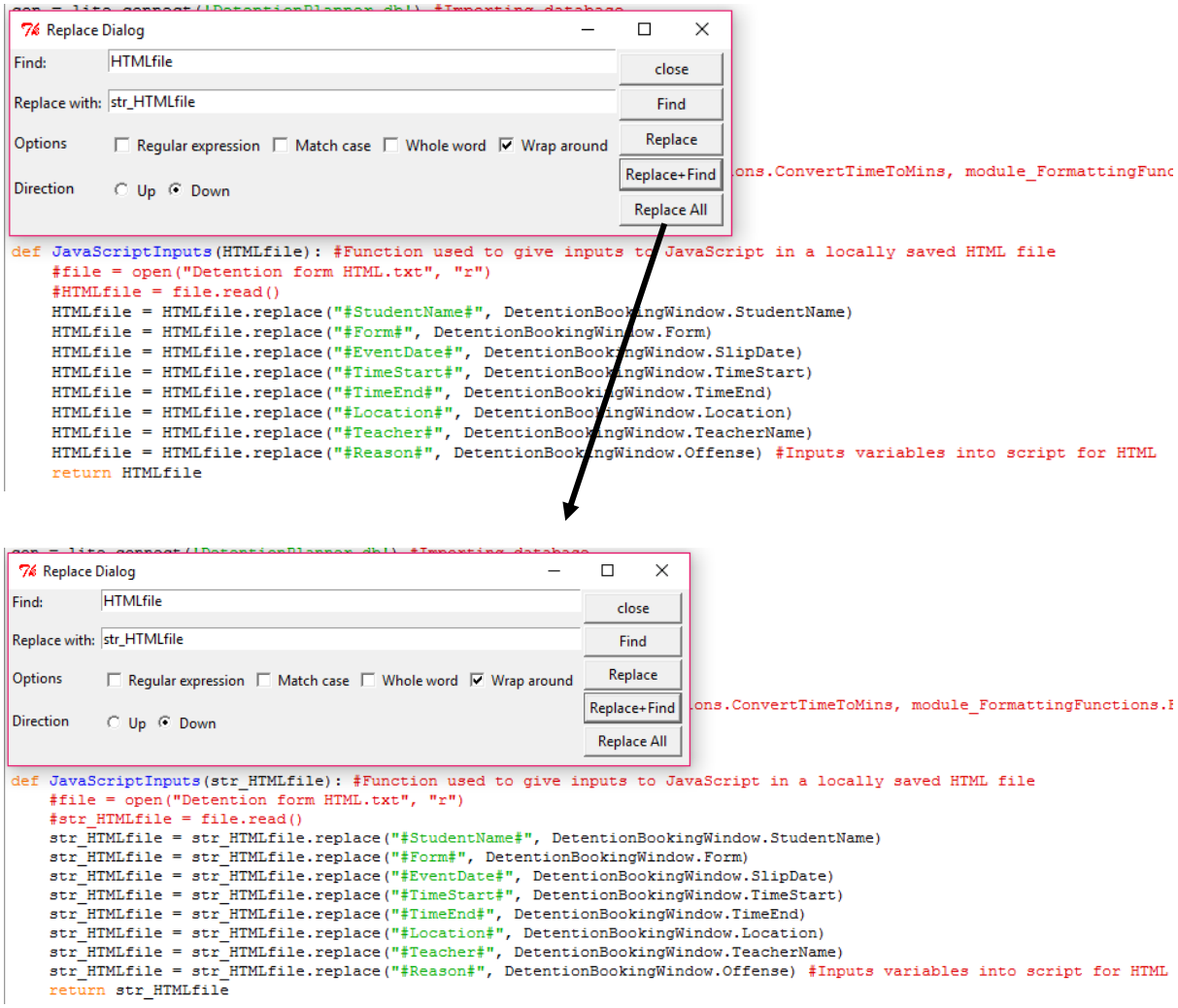
As shown above, an error message was produced when I attempted to run the program while important resources could not be located. Although the issue that this precaution is in place to identify (missing files) is different to the idea of having files named improperly, it should be able to identify the issue nonetheless.

Improving readability

Despite using modularity, lots of commenting, camel case variable names and occasionally underscores where appropriate, readability could be improved by using Hungarian notation to help anyone reading the code to identify the data type of a given variable.

Due to the whole program being near 1000 lines of code long (when including my various modules), I will not show screenshots of all the improvements to my code's readability.

However, by using the "find and replace" feature in IDLE, the process of replacing various variable names wasn't too long.




```
class DetentionBookingWindowClass(QtGui.QMainWindow, DetentionBookingWindow):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        ###################
        self.TB_BACK.clicked.connect(self.BACK)
        self.BTN_BookDetention.clicked.connect(self.CreateDetention)
        ###################
        global str_username
        self.int_StudentID = None
        self.str_Student = None
        self.str_Date = None
        self.str_TimeStart = None
        self.int_Duration = None
        self.str_Description = None
        self.int_Room = None
        self.str_FirstName = None
        self.str_SecondName = None    #all variables required for creating a detention record
        self.int_TimeStartMins = None
        self.int_TimeEndMins = None
    def BACK(self):
```

These screenshots show how I managed my approach of improving readability of my code.

Evaluation

Testing to inform evaluation

Due to having carried out thorough testing during my development process, I do not feel it necessary to test every part of my program again. For example, objective 24: *"Minimal clutter will be enforced by limiting number of buttons per window to 5…"* will not have changed this development, as I did not alter the windows after finishing the program.

As most of these tests will not have changed since I finished the section in development (due to me moving on from each section after I finish them, I will not show repeat tests that achieved the exact same outcome. Reference to the development section show how these tests were carried out and their respective outcomes.

However, I will run through the program again, testing general functionality of my program, and attempting to break it.

After this, I will have one of my end users test the program, in order to receive feedback on usability features. They will give me an idea of whether the program has good ease of access, is quick to use and intuitive.

From these different tests I will get feedback that should give me areas of improvement. This is especially relevant since this program was intended to be a prototype (as mentioned in my analysis of the problem), meaning its sole purpose was for testing and receiving feedback that could pave the way for achieving a polished solution to the problem.

I will only analyse the test results after I have carried out all post-development tests.

Testing robustness of the program

Initially, I will carry out the GUI testing that I proposed in my design of the solution. These tests will vary per the window that is being tested, but the general purpose is to attempt to use the program in unconventional ways which could result in unforeseen errors that I would have not picked up upon during my standard testing of the program.

Fast, repeated usage of buttons

The first tests are very self-explanatory, and I will simply show the output of the program when these tests were carried out, then later explain the outcome.

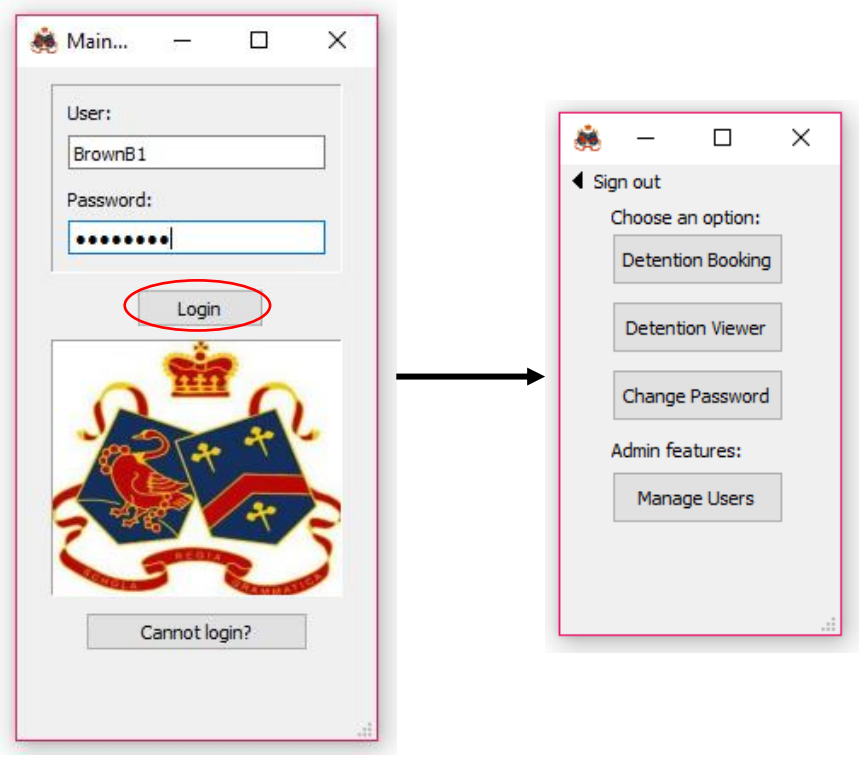


Figure 1. Second window opened so fast that a second click could not even be made before login screen closed.

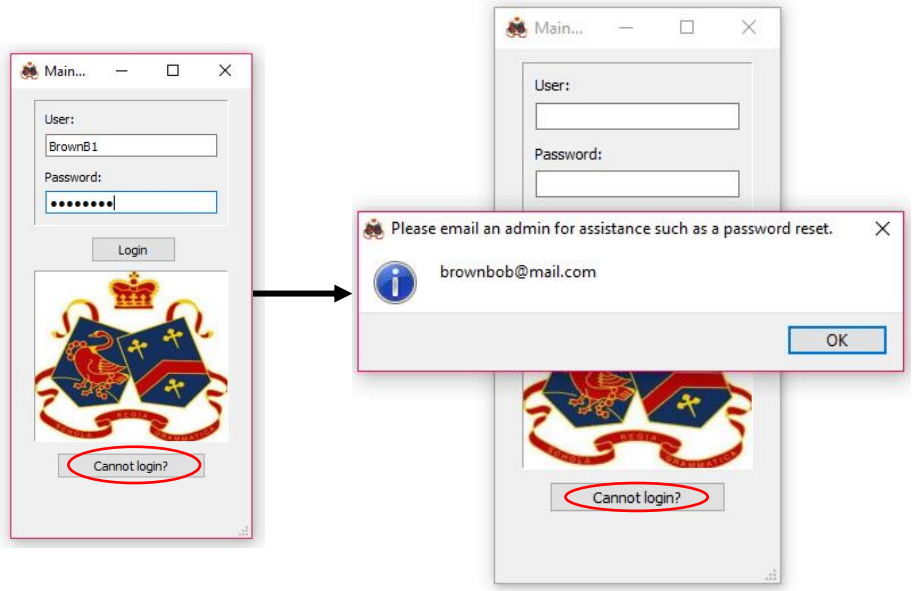


Figure 2. Pop-up immediately appeared, not allowing the button to be pressed multiple times.

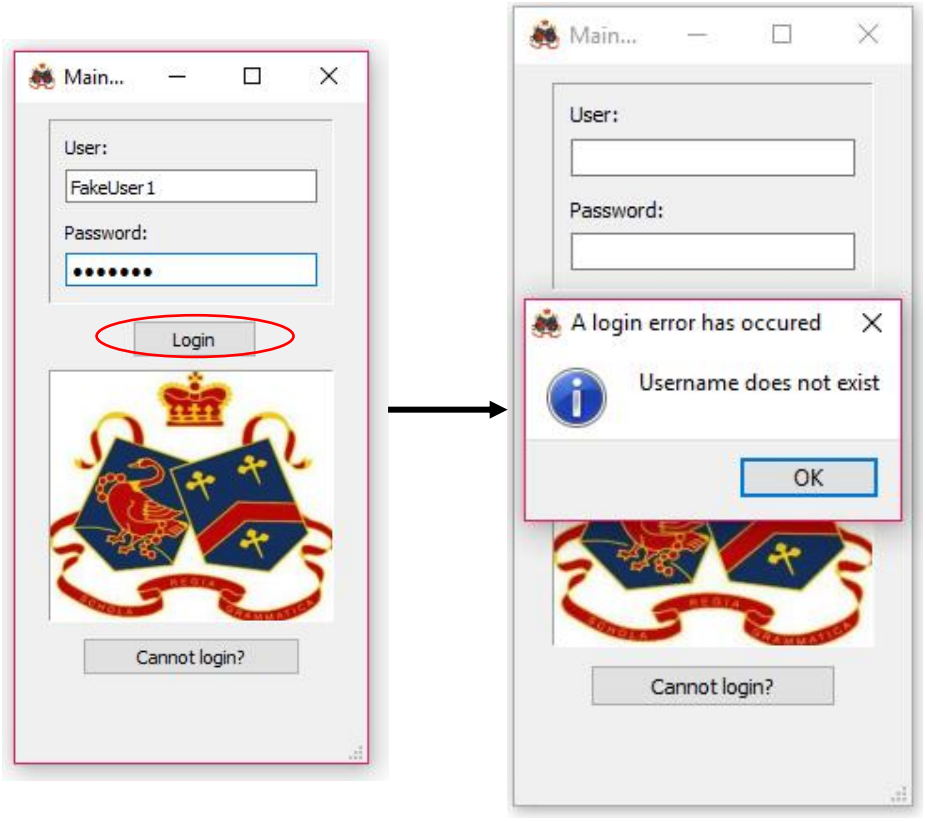


Figure 3. Pop-up was immediately created, before any successive clicks could be made.

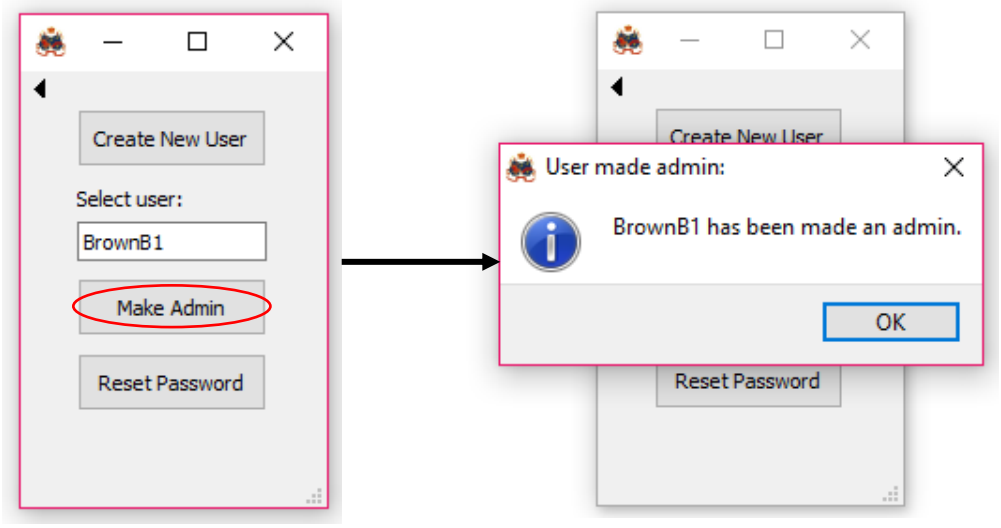


Figure 4. Pop-up appeared slower (possibly due to SQL query taking more time), thereby allowing two successive clicks. However, it made no apparent effect.

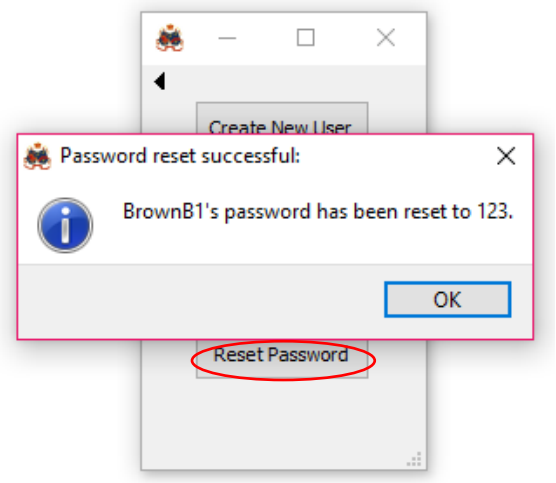
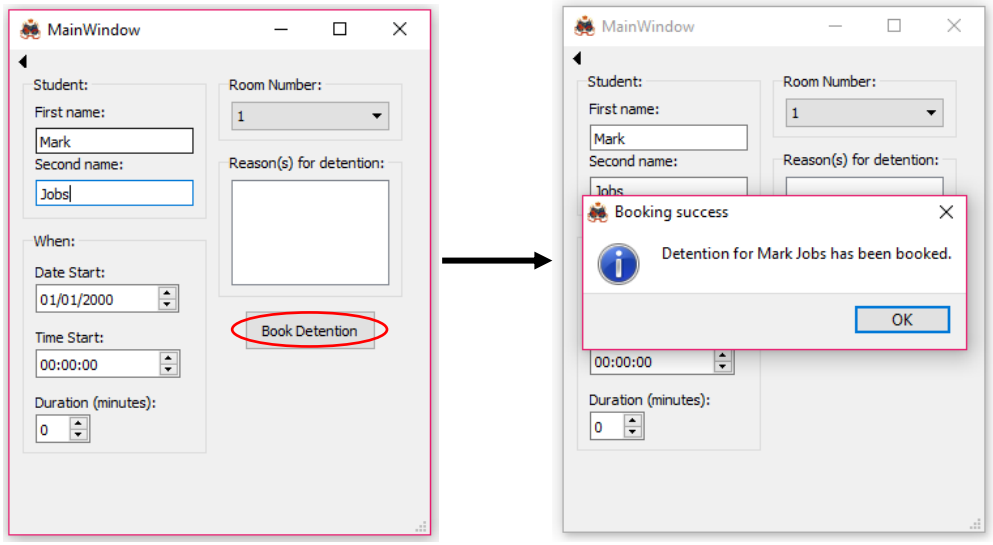


Figure 5. Similar to the above test, the pop-up prevented a significant number of clicks, and therefore no effect was observed.



DetentionID	EventDate	TimeStart	TimeEnd	Duration	RoomID	TeacherID	StudentID	Offense
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	2017-03-21	940	960	20	1	1	1	Late
2	2017-03-22	960	990	30	7	1	1	Rude
3	2000-01-01	0	0	0	1	1	1	

Figure 6. Due to the query being greater, the pop-up took longer to be created. Therefore, I was able to click the button approximately 5 times. However, when observing the database, it can be seen only 1 record was created.

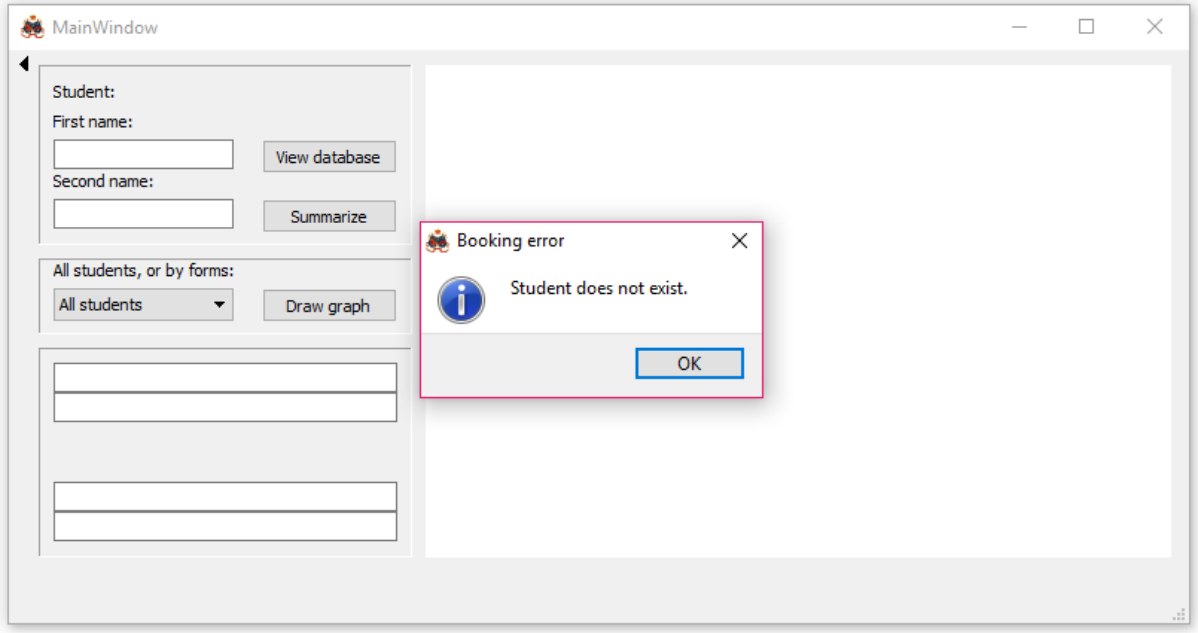


Figure 7. The pop-up was created so fast the there was only 1 click of the button.

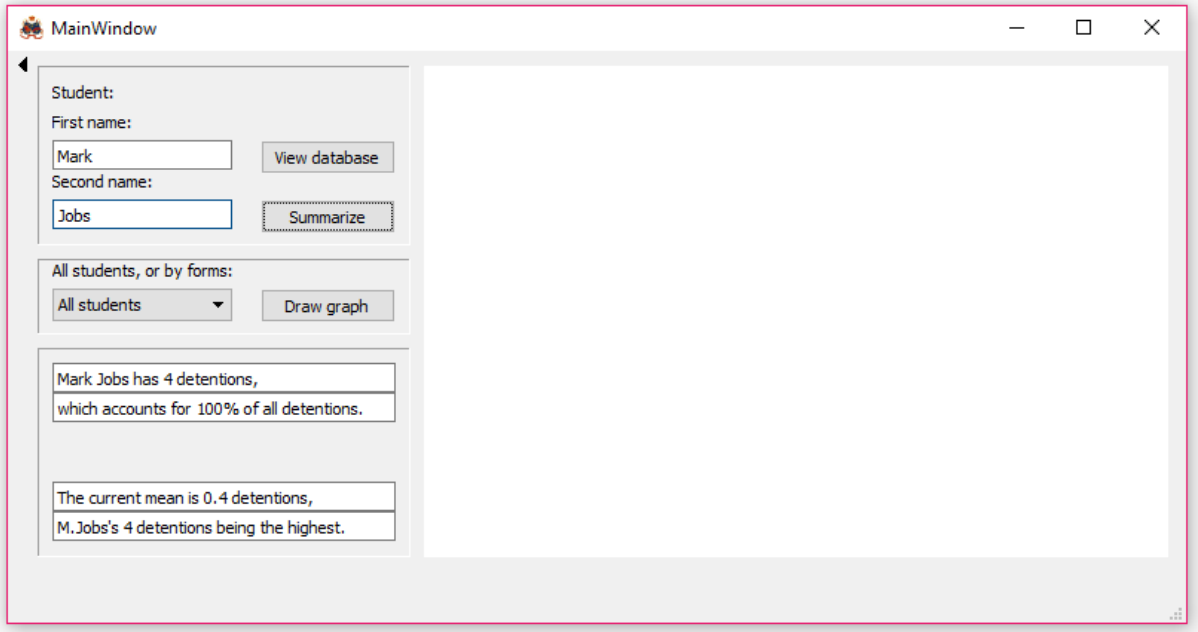


Figure 8. This summarise function was used approximately 50 times, by click the button as fast as I could. No visible effect was seen, however.

Analysis of robustness test

As seen in the above tests, the ability for the program to create pop-ups which notify users of information has served a hidden benefit of not allowing users to perform potentially destructive actions by pressing widgets too much.

However, as shown in figure 6, when the user *could* press the button a significant number of times before the pop-up was created, most likely due to the larger INSERT statement requiring a longer time to be carried out, no apparent effect was noted. I expected that potentially multiple new records would have been inserted, though this was clearly not the case when I viewed the database and took a screenshot of it.

The other unique test can be seen in figure 8, which shows the summarise button being pressed continuously, did not create a single pop-up which meant that the button could be pressed as frequently as I wanted. However, the only apparent effect was that the method was simply being run numerous times, with no other negative effect. The program did not freeze or crash, when in debugging mode I could see that the output was being generated as frequently as the button was being pressed.

As far as my “crash tests” are concerned, the program appears to be relatively robust. However, I may be able to put this down to the fact that the current device I am running it on contains a processor capable of reaching a processing speed of 2.6 GHz.

Usability testing

My usability testing will be carried out by Sam Hunt, due to him being one of my stakeholders that does not have any specific ties to working with computers. My other stakeholders work either as computing teachers, the head of IT or aren't a teacher at all, and therefore I thought it most appropriate to have the end user who would most represent teachers at schools.

To test usability features, I will simply have Sam Hunt attempt to carry out various tasks by using the program, timing them from when he opens the program.

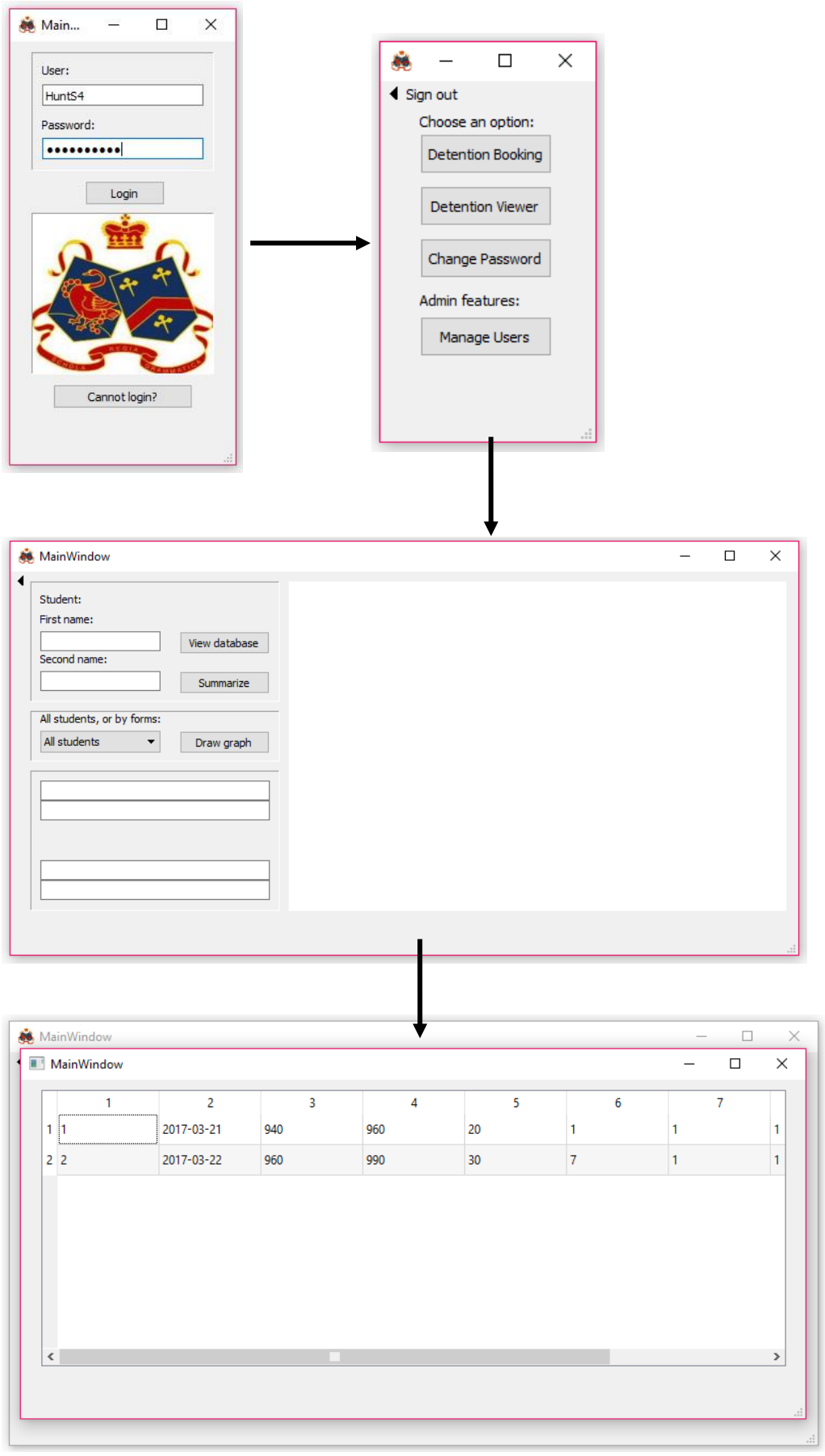
Another note is that this will be the first time the user has encountered the program in its entirety, so it should give a good indicator of how fast a user is able to familiarise themselves with the program, suggesting if it has good ease of access or not. After each task has been achieved, the user will log out and then attempt another task.

The account used for testing here will be HuntS4, password as "Password12" and the account will not be an administrative account.

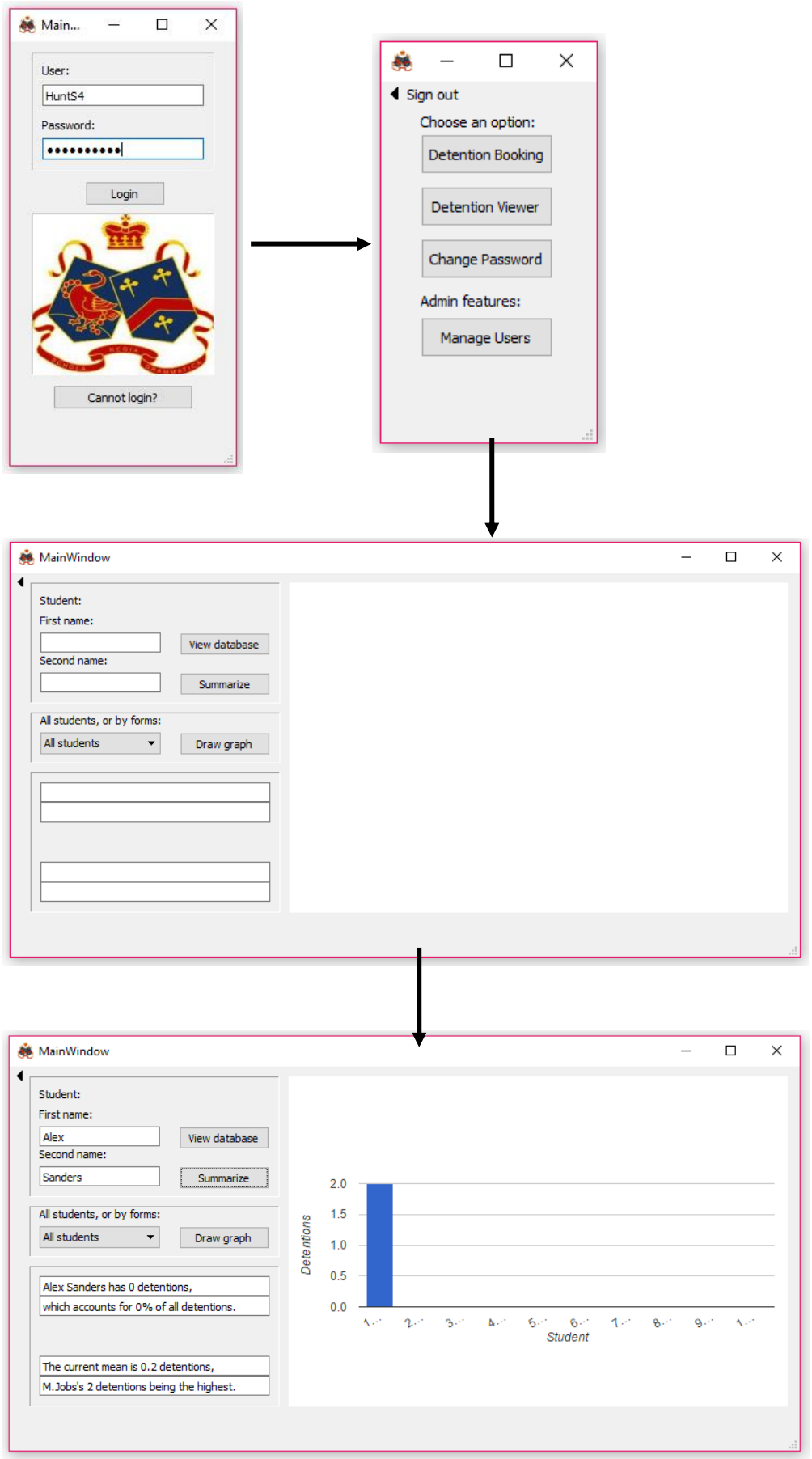
The tests will be as follows:

Task	Time taken	Comments from user
1. View the detentions records from the database.		
2. Create a graph of all the detentions had by students, and summarise Alex Sanders' detentions.		
3. Book a detention as follows: For Mark Jobs, start at 25/02/17, 16:00:00, duration of 30 minutes, in room number 25, with the reason that they had incomplete homework.		
4. Change own password to "Password123".		

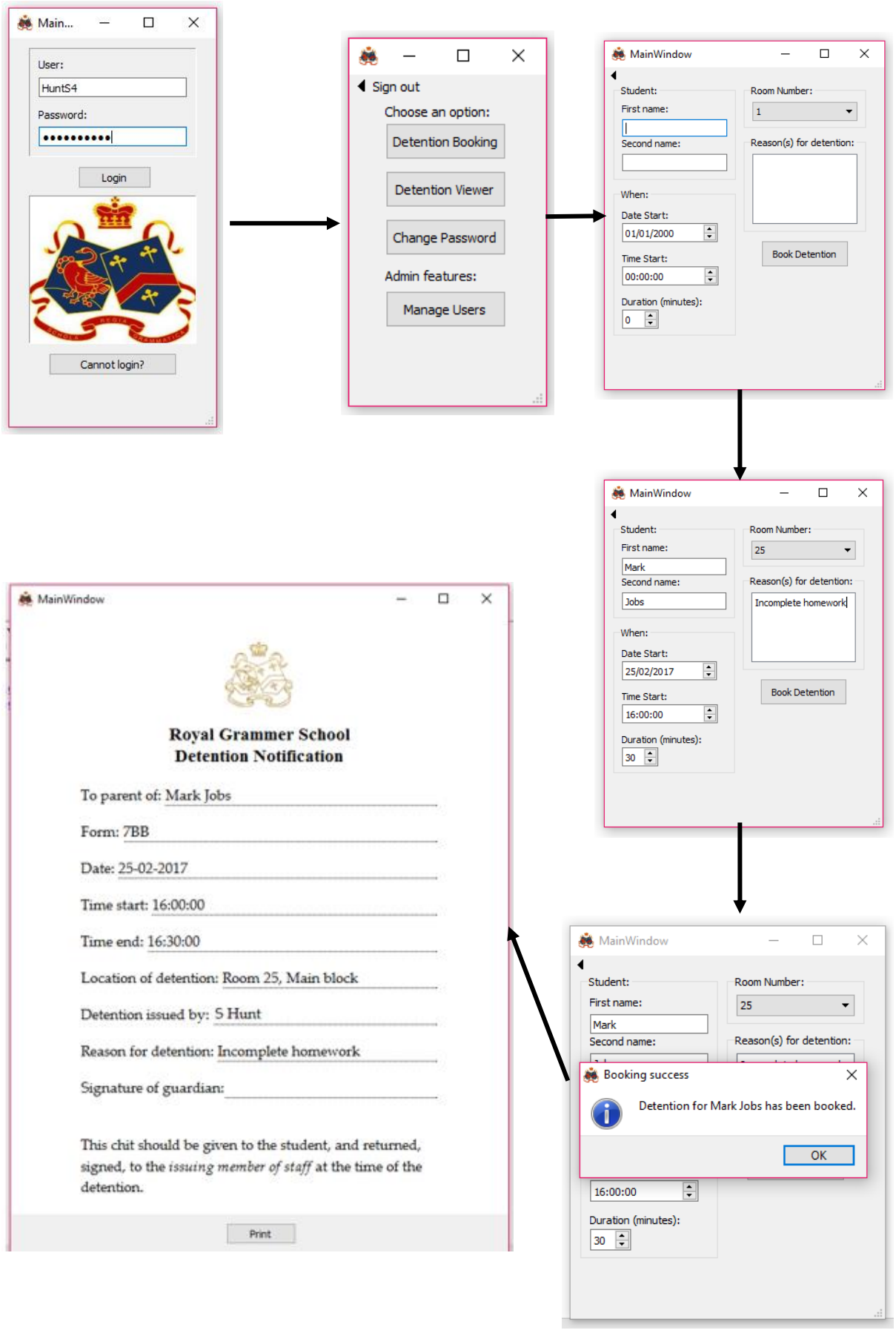
Task 1


Main...
User:
HuntS4
Password:
Login
Cannot login?
Sign out
Choose an option:
Detention Booking
Detention Viewer
Change Password
Admin features:
Manage Users
MainWindow
Student:
First name:
View database
Second name:
Summarize
All students, or by forms:
All students
Draw graph
MainWindow
MainWindow
1 2 3 4 5 6 7
1 1 2017-03-21 940 960 20 1 1 1
2 2 2017-03-22 960 990 30 7 1 1

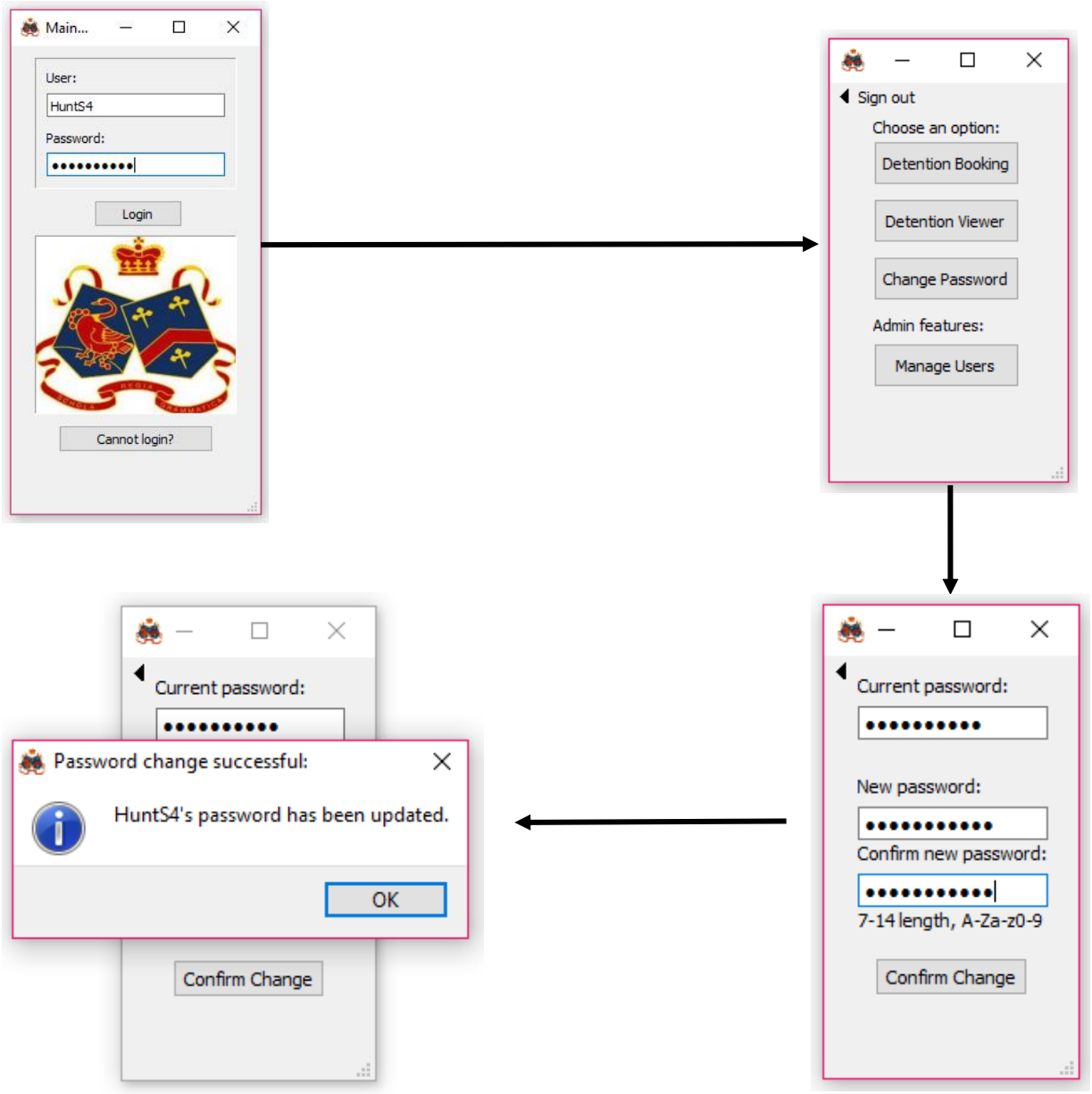
Task 2


Main...
User:
HuntS4
Password:
Login
Cannot login?
Sign out
Choose an option:
Detention Booking
Detention Viewer
Change Password
Admin features:
Manage Users
MainWindow
Student:
First name:
View database
Second name:
Summarize
All students, or by forms:
All students
Draw graph
MainWindow
Student:
First name:
Alex
View database
Second name:
Sanders
Summarize
All students, or by forms:
All students
Draw graph
Alex Sanders has 0 detentions,
which accounts for 0% of all detentions.
The current mean is 0.2 detentions,
M.Jobs's 2 detentions being the highest.
Detentions
2.0
1.5
1.0
0.5
0.0
Student

Task 3


Main...
User:
HuntS4
Password:
Login
Cannot login?
Sign out
Choose an option:
Detention Booking
Detention Viewer
Change Password
Admin features:
Manage Users
MainWindow
Student:
First name:
Second name:
Room Number:
1
Reason(s) for detention:
When:
Date Start:
01/01/2000
Time Start:
00:00:00
Duration (minutes):
0
Book Detention
MainWindow
Student:
First name:
Mark
Second name:
Jobs
Room Number:
25
Reason(s) for detention:
Incomplete homework
When:
Date Start:
25/02/2017
Time Start:
16:00:00
Duration (minutes):
30
Book Detention
MainWindow
Royal Grammer School
Detention Notification
To parent of: Mark Jobs
Form: 7BB
Date: 25-02-2017
Time start: 16:00:00
Time end: 16:30:00
Location of detention: Room 25, Main block
Detention issued by: S Hunt
Reason for detention: Incomplete homework
Signature of guardian:
This chit should be given to the student, and returned, signed, to the issuing member of staff at the time of the detention.
Print
MainWindow
Booking success
Detention for Mark Jobs has been booked.
OK

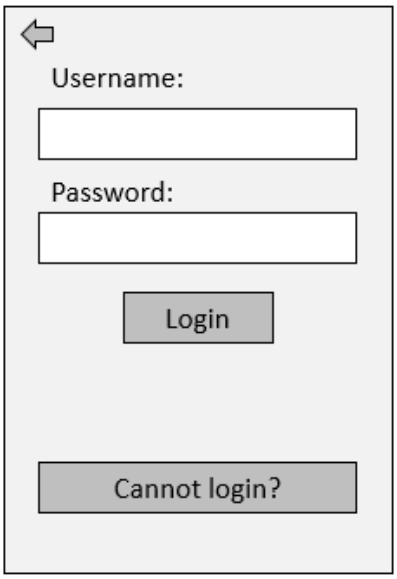
Task 4

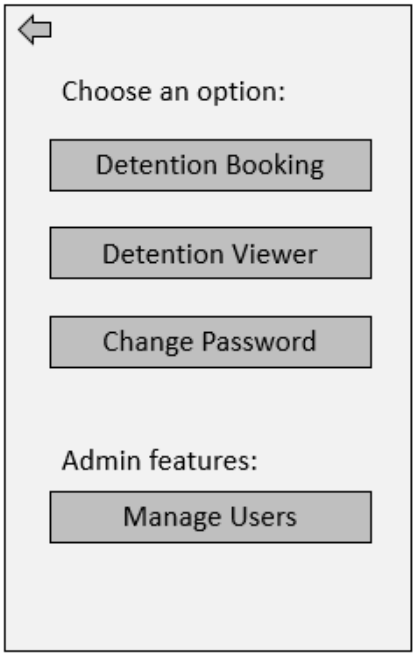

Main...
User:
HuntS4
Password:
Login
Cannot login?
Sign out
Choose an option:
Detention Booking
Detention Viewer
Change Password
Admin features:
Manage Users
Current password:
New password:
Confirm new password:
7-14 length, A-Za-z0-9
Confirm Change
Password change successful:
HuntS4's password has been updated.
OK
Current password:
Confirm Change

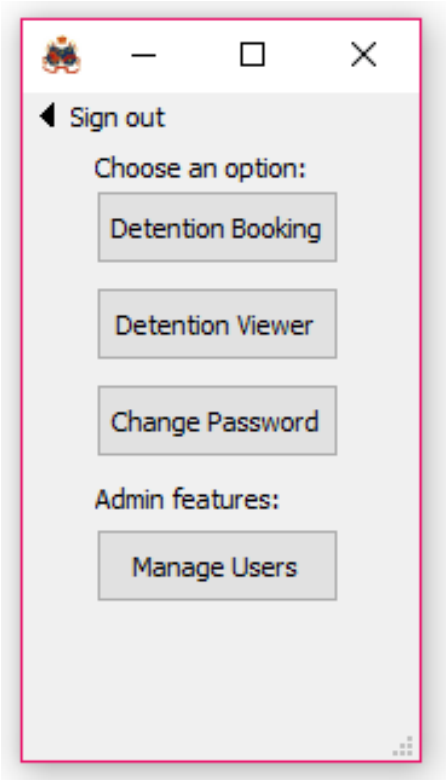
Overview of usability testing

Task	Time taken	Comments from user
1. View the detentions records from the database.	00:23.42	"This was very easy, thanks to all the windows being very simple. There was very little room for error when navigating to the windows that I needed to go to if I wanted to view the database. I started by logging in, which again, was pretty simple, as there was no other thing for me to do other than input my login details. I could have pressed the cannot login button, but obviously didn't because that was clearly going to be for login issues. My only concern was how login window didn't appear to set my cursor to the input boxes by default, so I had to click there myself. Not a huge concern, but it would just be objectively better if it did set your cursor by default."
2. Create a graph of all the detentions had by students, and summarise Alex Sanders' detentions.	00:27.90	"All the points as the ones above remain, and importantly I think that the detention viewer window was again, very easy to work with, despite it apparently doing a fairly large variety of things. When I first saw the white rectangles when I was trying to view the database in my first test, I was confused about their purpose, but after pressing the summarise buttons, when I put the student's name in, it became apparent that they were supposed to give sentences. Although it worked, and I could use both the graphing and summarising parts very quickly and easily, I do think that the white rectangles were not as aesthetically pleasing as a simple large white box for all the text would have been. The rest of the important things however, like using little colour, small windows and a low amount of buttons had clearly been achieved however."
3. Book a detention as follows: Mark Jobs, start at 25/02/17, 16:00:00, 30 min duration, room 25, because they had "incomplete homework."	00:56.39	"Although I took a fair bit longer to use this part of the program, it wasn't necessarily because it was particularly complicated, there was just a lot more to type. At first I had to take a small moment to understand what the window was doing. It was easy enough to locate since the detention booking button on the menu was pretty self-explanatory. Once I realised it was just going to be taking my inputs, it was only a matter of typing and clicking. I liked how this window automatically made the detention slip after the detention had been booked, it meant I just had to do less."
4. Change own password to "Password123".	00:21.03	"By this point, I was much quicker with the logging in process, and the change password window was easy to locate and use, hence why I was able to use the feature really quickly.

User's thoughts on general design of program (GUI)







After showing the above comparisons (and comparisons of more windows), I asked Sam Hunt what he thought about the final designs of the GUI now that he has interacted with the program.

The blue text shows my questions, with the italic quotations showing Sam Hunt's response.

Considering functionality only and not aesthetics, does the GUI serve its function?

"Yes. Much like the original designs, it has very minimal buttons per each, small window. However, all the buttons are definitely there to make the program works, it's a nice compromise between having enough complexity while still being simple enough to use. Also, I really liked the pop-up usage, there weren't too many to the point that they became annoying, but they did help a lot with informing me what was going on with the program after I did something."

What is your opinion on the aesthetics of the final iteration of the GUI?

"It's not incredible to look at, which isn't necessarily a bad thing. By this, I mean that it's design allows easy use of the program by not having much going on with each window. It's a sacrifice of looking really attractive, to work very well, so I think it is good."

What are your thoughts on the detention form's design that is generated?

"It's my favourite part of the whole design of this program. It definitely achieves replicating the original paper slip that we use at this school, and it definitely serves its purpose of giving information about a detention. I personally feel that the printout shouldn't have cropped out the message saying what should be done with chit however."

This interview shows that the GUI was successful regarding its design, though it could have some improvements. Most notably, the cropping of the message that states what should be done with chit.

I will discuss the analysis of these tests in greater detail later.

Success of the solution

By referring to the various testing during development, post-development, by users and crash testing, I can confidently present a table to show whether or not I have achieved the relevant success criteria.

For a more in-depth consideration of the test themselves, refer to the development section of this document. This table serves as an overview of the success criteria.

Objective	**Success? (yes, no, partial)**	**Additional comments**
1.	Yes	Program able to identify if UI files have been located or not.
2.	Yes	
3.	Yes	
4.	Yes	
5.	Yes	
6.	Yes	
7.	Yes	
8.	Yes	
9.	Yes	
10.	Yes	
11.	Yes	
12.	Yes	Button name was changed to “Cannot login?”, as it is more descriptive than “help”.
13.	Yes	Privileges specifically allow access to an extra window, which allows creation of users, resetting of passwords and creation of admins.
14.	Yes	
15.	Yes	
16.	Yes	
17.	Yes	
18.	Yes	
19.	Yes	
20.	Yes	
21.	Yes	
22.	Yes	
23.	Yes	
24.	Yes	
25.	Partial	Although it was initially achieved, I altered the detention viewer window, to have the text be displayed as separate line edits, increasing the widget number by 2. This put it over the 9-widget limit. Hence why I would class this as a partial success.
26.	Partial	Can be argued that “reason” for detention may lend itself to requiring more than one word at times. Hence why I class this as a partial success, though in reality it is closer to a complete success.

27.	Yes	
28.	Yes	All videos can actual be visited, once logged in, by only using 2 clicks, as opposed to the 3 stated by this objective.
29.	Yes	
30.	Yes	
31.	Yes	
32.	Yes	
33.	Yes	
34.	Yes	
35.	Yes	
36.	Yes	
37.	Yes	
38.	Yes	
39.	Yes	
40.	Yes	
41.	Yes	
42.	Yes	
43.	Yes	
44.	Yes	
45.	Yes	
46.	Yes	
47.	Yes	
48.	Yes	
49.	Yes	
50.	Yes	
51.	Partial	Although it could successfully save the detention form as an XML file, as Sam Hunt noted, the instructions related to the chit had been cropped off, which was not as planned in the original design.
52.	Yes	
53.	Yes	
54.	Yes	
55.	Yes	
56.	Yes	
57.	Yes	
58.	Yes	
59.	Yes	
60.	Yes	

This summarisation of the success criteria shows that the solution appears to have achieved all the set objectives, to an extent.

Analysing success (or lack of success) of usability features

Although most success criteria were fully achieved, this does not mean there is no room for improvement with the program. After interviewing Sam Hunt when he trailed the finished solution, it became apparent that there were issues worth note with the GUI, despite it almost completely achieving all the success criteria related to usability features.

One issue he noted, was how the printout forms instructions, which are usually located at the bottom of the chit, were missing. This was not intended when I attempted to create the functionality to save the chit as an XML file, however.

Initially, the form had about a quarter of it cropped, including all the instructions at the bottom, and half of the RGS logo. For a school, I deemed it to be very important that the RGS logo was present, as it will be seen by parents. It is reassuring for them to know that it is a genuine slip, which the school's official logo helps to show. Therefore, I decided the best course of action that I could take, with the limited amount of time that I had, was to choose to only crop the instructions off, which would keep the chit looking neat and official while still being able to be saved as an XML file.

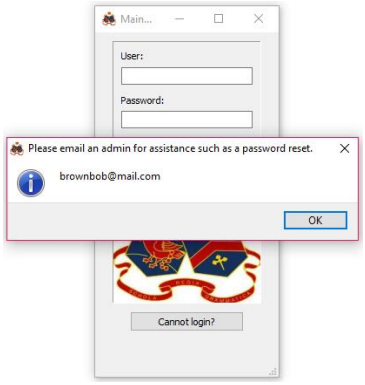
If I had more time to fix this problem, my approach would not have been to choose what to sacrifice with the form, but to find a solution that would let me save the form as originally intended. If I could not find another method of fixing the issue of the slip being cropped when saved as an XML file, then I could have explored solutions that involve saving the chit as a different type of file. For example, if I had intended to save it as a pdf file as opposed to using the XPS format specifically.

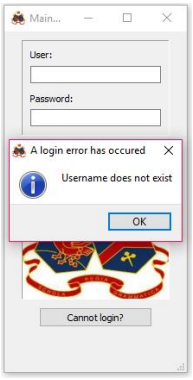
Describing the final product

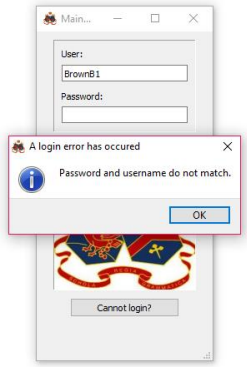
Usability features are very important to my end users, as I emphasized when drawing up my success criteria for this project, as many of which, are usability features. For example: objectives 20 and 21 state the program must use small windows, with little to no colour. Therefore, to observe whether the usability features have been achieved, it is worth noting the evaluation of my success criteria (under success of the solution), and my final interview with Sam Hunt about GUI design.

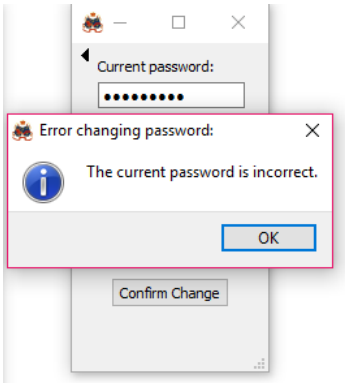
One heavily emphasised usability feature found in my program is the usage of pop-up windows to provide the users with information. These may be seen throughout the iterative development process section of this document; however, some examples may be seen below.

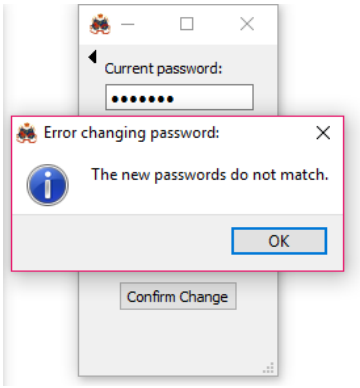
I would argue that these pop-up windows were achieved successfully, as evidenced in the final interview with Sam Hunt where he states that "*(he) really liked the pop-up usage, there weren't too many to the point that they became annoying, but they did help a lot with informing me what was going on with the program after I did something.*".

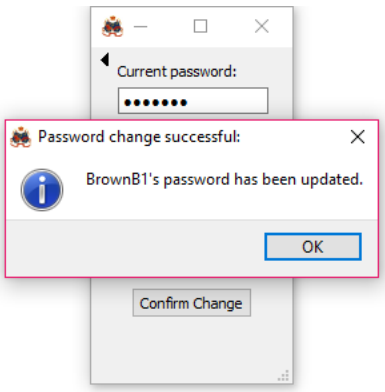


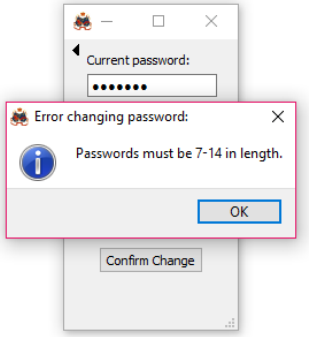


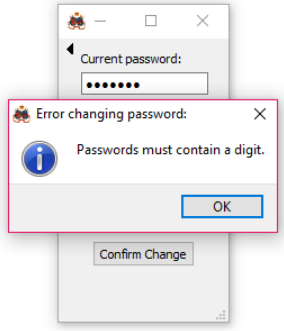


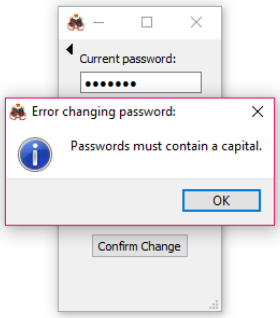












However, there are more usability features which I could use in hind sight, that are missing here. When Sam Hunt suggested that there weren't too many to the point where it was "annoying", I thought that others may disagree, and say there were too many. Thus, I have decided that I could improve the usability features of my program.

One idea which I have only considered once I finished the program, is to use labels within the windows for minor notifications to the user. For example, I do not need to create a pop-up if the user tried to use an invalid password when changing their password, I could have simply had small messages appear on the actual window. Another idea would be to change the colour of the line edits to red, thus identifying both the issue, and which input caused the issue, however I would need to consider what colours would be viable to use if I am keeping to objective 21 which was made to ensure the program catered to colour blind users.

Other ideas include allowing the user to have more customisation of their GUI. Although I still wish to cater to colour-blind users, and those with eye site problems, it would not be to anyone's detriment if the use of colour was down to the user. As stated by Sam Hunt, "It's not incredible to look at", when referring to the aesthetics of the final GUI. This suggest that there is room for improvement when referring to how the program looks, and not how it functions.

If I am to let the user change the colours present in the program, then it would also not be harmful to let them customise the fonts that go with it. For example, some user's may wish to have a bigger font.

Maintenance and development

Maintainability of solution

As the current state of my program stands, it is a very stable prototype which could realistically be converted to a fully functioning system to be implemented within schools. I will go into more detail on the potential developments of this prototype later, however for now I will be discussing maintainability of the current prototype that I have created.

As discovered in my analysis of the problem, a solution would mostly comprise of algorithms that have an O(n) complexity. Therefore, it is fair to assume that when implementing this same prototype into a school, it would be able to run relatively smoothly. This reason I can only declare a vague suggestion for how it would run is because datasets could reach over 1000 students. This would lead to a very large number of detention records, especially since my prototype as it currently stands would store detention records indefinitely, and not archive them by year for example (which would be a plausible solution to having a too large store of detentions).

As a rough calculation, assuming I have a school of 1000 students, and that each student would get a detention every 2 months, the total number of detention records at the end of a given school year would be as follows:

Approximately 4 detentions per each student over the whole year, multiplied by 1000 students. This would result in 4000 detentions records. Obviously this a very rough calculation due to it being based off massive assumptions, though it does serve to show the point that this prototype could result in requiring a very large database if it were to be fully implemented.

With a large database that has a lot of records, the algorithms being run on the program will take a longer amount of time.

For example, with 1000 student records, the algorithm I use to determine which student has the most detentions would iterate 1000 times before it could return an answer. It must individually check each record to determine whether the student has more detentions than the current highest number of detentions.

Understandably, these algorithms would take far longer to be executed. There are also many more factors to consider when understanding how long the program would take to carry out these already tested features of the program. For example, if the database is located on a network, the time taken to access this database would also take more time. Another factor may involve the devices that are being used. If the dataset is already large, then the time taken to run these algorithms will be magnified by a slower device being used.

It is therefore inconclusive as to how my program could be maintained in the event that it was fully implemented in a school, as intended.

The next discussion point for maintainability would be how someone other than me could maintain the program that I have created.

Due to me using modularity, I feel I have successfully created a framework of code that is relatively easy to read, with the ability to insert modules as the new editor would see fit. They would also have the ability to look at, and edit my existing modules individually. To give an example, if the new editor wished to remove the way that password validation enforces the use of a capital, they need only enter the “module_ValidationFunctions” module and alter the “RegexPasswords” function as they see fit. It would then change the way that function is used in every part of the main program. I therefore feel that it would be very viable to maintain my code in this aspect.

Another point would be for someone unfamiliar with my code to be able to read, and understand what it does. By use of Hungarian notation, camel case variable naming and underscores I feel that my variable names and sub-routine names are very meaningful and therefore self-explanatory. On top of this, I have very frequently appearing comments in my code, with every loop, conditional statement and sub-routine being explained. I have comments describing the use of most variables on top of this, if the meaningful variable names are not enough by themselves.

That being said, despite all this, due to my code reaching nearly 1000 lines (including my own modules I have created), it would be very difficult for someone to look at my code with no prior knowledge of what it does. I therefore think that if I had more time to work on this program, I would develop a guide of sorts, that could talk the new proposed editor of my code through the ins and outs of my code. It could potentially contain my thoughts on how I would implement potentially new features, and why it would be difficult and potentially ill-advised to attempt to implement other features.

Potential developments

As already mentioned, my final iteration of my solution was always intended to just be a prototype, due to the nature of me not being able to feasibly make use of a network for my solution, as well as other reasons. This has been discussed in more detail in my analysis of the problem, however here I will be discussing my thoughts on what more could be added, had I had the time, or skill.

Further analysis of success criteria

Due to my solution being a prototype, I did not create success criteria that I would not have been able to achieve, as the unachievable objectives were simply part of what my main implementation of the program would be able to do. This includes features such as network connectivity, hence why it is not found on any of my success criteria, as these criteria were drawn up knowing I was creating a prototype and not a real implementation of the solution.

Because of this, I have been able to conclude that my solution did not have a significant amount of partially failed tests, with no completely failed tests. This is evidenced by the large number of tests carried out in my development section, which have been compared to my success criteria. Even tests which I had not been able to prepare my program for in the same way, such as he GUI “crash tests”, were majorly successful.

Because of these above reasons, I can state with confidence that further development into this same prototype would achieve very little, and the time would be best spent in achieving objectives that are relevant for the actual implementation of the program. In other words, if I had more time and more programming skill, I would create new success criteria, involving network connectivity, support of different devices and more.

Potential future success criteria

I will show a small list of features that I would need to create given I had the time to implement a full solution to the problem, as opposed to a prototype:

- Network connectivity.
- Support of various devices and therefore operating systems.
- System allowing automation of importing a school’s data into a database.
- An external document giving an in-depth analysis of my code.

This is only a rough list of overarching goals that a complete implementation of my prototype should be able to achieve, however, these would be better split into smaller sub-objectives were a complete new list of objectives to be drawn up. As well as this, there would be more objectives worth noting, however these would be more easily identified were there to be an analysis of the new problem, which differs from the problem relevant in this project.

Network connectivity

Due to not having a broad knowledge of working with school networks, especially on the large scale for a school network, I would have to spend a lot of time researching. On top of that, I would need to

spend time in contact with the IT workers at a given school that I would be working with, to understand how I would cater my program to their situation specifically.

The reason network connectivity would be on an update success criteria list is that the only way this system would be useful is if teachers could book new detentions on their own devices, at any time. Without network connectivity, the database would only be available on a single device, meaning users would only have access at the location of the device, presumably at the school, and only 1 user could access the program at any given time. This prototype served to show how this system can be created, and how it can tackle the problem that I identified in m analysis, but in order for it to fully achieve its solution, a full implementation with network capabilities would be required.

Support of different devices

For device support, I would need to carry out further research. For example, I have never worked with MAC-OS before, meaning I have very little knowledge of what would, and wouldn't work with it. The same applies to any UNIX based systems.

Specifically, for working with phones, which would be a very useful feature of my program, I would need to investigate what different sizes of screens I would be working with. From there, I could find how I would need to shrink my windows to have them fit on the screen, but still be big enough to work with.

Once I have found out this information, I would need to re-imagine the design of my system, as some approaches that I have used would simply not work as well when using a mobile device. Examples that I have begun to consider include the following:

- Having multiple windows open at the same time isn't feasible, due to limited screen space. This would make opening the database view require a new approach.
- Using combo boxes would become difficult, as the choices would be too hard to select on a smaller screen. Therefore, a pop-up to appear which occupies the whole screen to present the user with input choices, may be more appropriate.
- A mobile version would require the previously mentioned network connectivity, as phones don't have the storage space which may be required for locally storing the large database records that a whole school may require.
- Consideration of different programming language may be necessary, and potentially may require me to work with a different database management software.

These are simply rough ideas for now, but these could easily be extended upon if I were to re-analyse the problem, and plan an updated design.

Importation of existing school records

I have already mentioned the importance of being able to import existing records into my database in the analysis section of this document, and I still feel it would be very important. To summarise what I suggested previously, because some schools have student numbers approaching the thousands, it would take an extremely long amount of time to manually input all the necessary data into the database.

Documentational guide of the program

Finally, as suggested in the “maintainability of the solution” section, I think it would be very beneficial to have a document that would help new editors of the program to understand how it works. Thus, allowing any future editor to have an easier time understanding, maintaining and improving my program.

Bibliography

1. https://en.wikipedia.org/wiki/School_Information_Management_System#cite_note-CapitaSoftwareInfo-2
2. http://www.capita-sims.co.uk/
3. http://www.legislation.gov.uk/ukpga/1998/29/contents
4. https://www.gov.uk/data-protection/the-data-protection-act
5. http://www.getalma.com/product-tour.html
6. http://portablepython.com/wiki/PortablePython3.2.5.1/
7. https://www.python.org/download/releases/3.2.5/
8. https://riverbankcomputing.com/software/pyqt/intro
9. https://sqlite.org/releaselog/3_9_2.html