Gym booking system

2017 Computing Coursework

Aleksander Dzialak

Candidate no. 2521

Centre no. 52423

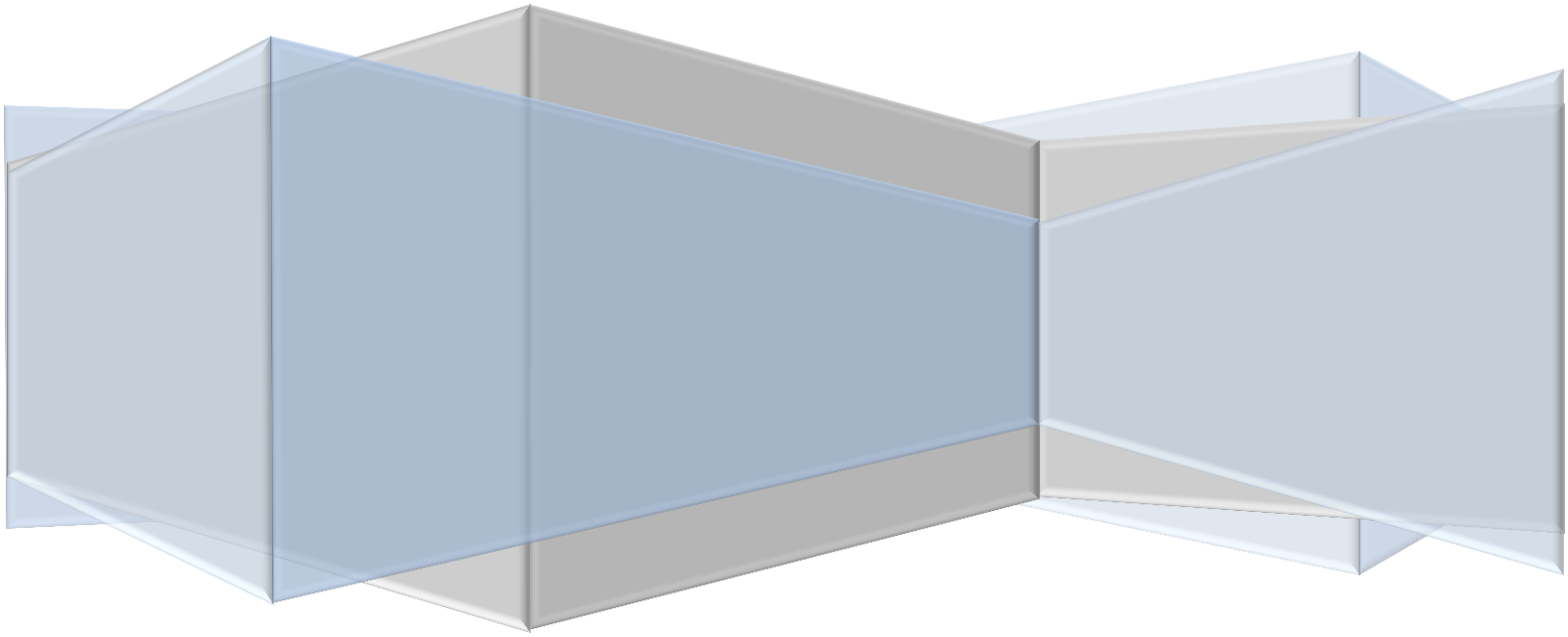


Table of Contents

Analysis

Problem identification

Bodybuilding and fitness has experienced huge growth over the past couple of years. In modern day society, more and more people are attending the gym whilst staying on top of our other main activities like work and school. It is difficult therefore for us to stay on track of our gym programs, our Body Mass Index (BMI) and our weight throughout a certain period which is important if gym users want to see changes in their body. When I attend the gym, I see a lot of people with notepads and a pen which means they do everything manually, like writing your plan everyday by hand which is a waste of time considering how much limited time we have as mentioned above. The problem can be solved easily through a computing program which would save a lot of time for many people going to the gym and involved in fitness. This can be solved through fitness software by giving gym users the ability to see existing workout plans specific to their final goal – whether it is to lose weight or gain more muscle or to transform. Through this feature the person would not have to write everything by hand and instead they could see their plan on screen every time they log in to their account. This is simple through a computing approach as I would collate different programs and put them into one place, split by sections so that each user can pick a workout plan suited to their needs. Now this is a much easier solution than a user having to look through many websites searching for a plan when they have a place to see multiple plans in one place. Another problem is the user having to calculate their BMI and many people do not know how to. Since this is an important factor in determining whether you progress, it needs to be included in my program as this also adds to the cohesive idea that my program will be. This also can be solved through a computing approach simply by having a calculator setup to calculate the weight against the height of the user using the BMI formula. This will be a cohesive program that brings in all important aspects of the gym as this is what I found during my research – not enough people can find the all-in-one program.

Type of software

The software I'm proposing solves these problems that gym users experience and brings in all the important factors that they need into one place. Through using PyQt4 I can create a suitable user interface with widgets which make for satisfying user navigation. PyQt4 offers widgets which make this idea very easy to implement and it is perfect for setting up menus and buttons which are exactly what the user is looking for – an easy to navigate menu through which they can choose all the options from. A good appearance for the user interface is important as it welcomes the user in and makes them come back every day and using graphics and well-rounded design that PyQt4 offers, it will be a suitable option to use. Using Python, I will be able to link all the widgets and different user interfaces to each other. Python is a suitable language for me as I have had experience with it for the past 3 years. Also, PyQt4 works very well with python as the designer is a simple drag and drop of widgets which means every single detail can be changed to make the user interface as friendly as possible. Also, the coding is relatively simple as most of it is the same and the code must be changed into a way that fits your specific interface, therefore a lot more time can be spent making the interface as user friendly as possible instead of having to write chunks of code just to change one variable in the program. This is what will make my software stand out – the simplicity yet effectiveness of the interface.

Output

There are many forms of output required in this program. First, the outputs from the database. The admin should be able to view all workouts and edit user details if required. This requires for certain tables to be output to the screen. The admin should also have a graph showing a certain characteristic match within the table e.g. male to female ratio. For the users, output such as their previous weight should be displayed to make for easy comparison. Also, their BMI should be displayed along with what health category they are in. This is easily done by the computer as the weight and height calculations to get the BMI can be done without the user needing to calculate this every time via a calculator for example. The user can simply enter their weight and height and anytime they update their weight, they should be able to calculate their BMI. With the workout booking a person shouldn't have to worry about selecting the payment for each trainer and shouldn't need to calculate the final amount depending on the duration of the workout. All the workouts are stored in the database and the workouts are tailored to what their goal at the gym is (build, cut or transform their body) so they don't have to look through plans which don't match their needs. The user should see the workouts they have booked and what plan they did on that day.

State

This program will be a prototype as today most people don't log in to a computer to use a piece of software when they can use their mobile phone to do so. Therefore, all the windows would have to be much smaller on a phone and the ability to rotate the orientation from portrait to landscape. Having a program like mine on a phone would bring portability to my booking system as well as being able to check for workouts. I am prototyping the overall features that would be used in a mobile app. The GUI is made for a personal computer/laptop and this would need to be changed for an app on a phone. The reason this is a prototype is to see if the features would work well on a personal computer and if they are transferrable to a small based platform i.e. a mobile phone.

A computational approach

The reason this program is best solved through the user of a computational approach is because anything else does not store data as well and as effectively. If the user was to use paper to record everything and had to work out the totals of each session, it would be a waste of time which is against our criteria of a student who must attend school and keep on top of their gym details. Therefore, this platform makes it easy for them to store details of their workout as well as seen the progression of their weight, and displaying BMI without having to manually look up the formula, workout it out and then compare the value to a table. Using a computational approach makes this so convenient for the user who has limited time throughout the day. Moreover, a computer can calculate precise values much quicker than we can (decimal numbers) and storing everything on the computer makes it much easier to search through later as opposed to a paper solution (in the short term the user could find their wanted workout on a piece of paper easily, but with more data it would make it very difficult), whereas with a computer the time difference to search through a data set won't be noticed by the user. If there are hundreds of workouts for a user, it is much easier to search for them using a computer database than it is looking through pieces of paper.

Stakeholders

The final user for my program will be a member of a gym of age 18. His name is Andrew Hersh. The software should also be simple enough for a non-gym user to understand and navigate through therefore the people I interview should be members of the gym as well as people who don't go to

the gym so that the full range of users is considered so the user interface is as smooth and easy to use for everyone. People of all ages go to the gym; however, most gyms have an age restriction of at least 16 years of age; therefore, I will only interview anyone over that age. I am looking at interviewing 6 people – 5 gym users to give me a good idea of what it is they are looking for in fitness software and 1 non-gym user so that the full range of people is considered (all above 16 years of age). Also, both genders must be considered in the design and development of my program and men and women will be interviewed. This makes it so that the program represents all gym users across a range of ages as well as both genders. However, the Andrew is a person who attends the gym at least 3 times a week as they will be the one that will benefit from the software the most, yet the program needs to be suitable enough for almost anyone to use it whether they are just starting at the gym or already have experience with workout plans, calculating BMI and keeping track of their weight. Andrew will be using the program to improve their time management, and not wasting unnecessary time. This will help to focus more on school as well as other aspects such as social life.

Interview

I asked my stakeholders, mentioned above, to answer some questions as it would be them that would be using my software in the end. These are the questions and the results I got back from them, which I then analysed:

1) **Do you go to the gym?**
    a. Yes – 5
    b. No – 1
2) **How often do you go?**
    a. 0 times a week - 1
    b. 1-3 times a week – 3
    c. 3+ times a week – 2
3) **Have you used any bodybuilding/fitness apps before?**
    a. No – 3
    b. Yes – 3
4) **If yes, what did you like about it?**
    a. “I could keep a track of my weight easily”
    b. “Nothing, they were all bad”
    c. “I didn’t like it”
5) **If you go to the gym do you write everything manually?**
    a. Yes – 5
6) **Would you prefer an easier more computerised approach?**
    a. Yes – 4
    b. No – 1
7) **What do you like about existing user interfaces from any programs you’ve used in the past?**
    a. “Colour and graphics need to be attractive and appealing to the eye”
    b. “Easy to navigate”
    c. “Simplicity”
8) **If you don’t go to the gym and you started going, do you think a software approach like this would be beneficial to you?**

a. “Yes, because I could find all of what I need in one place instead of navigating through the web looking through many websites to find what I need”

Analysis of my findings

From the interviews, I conducted, it was clear that all the gym users that have used these apps and programs, were not fully satisfied with them. Also, the one non-gym user said it would be much easier if he found all he needed in one place if he was to start going to the gym. Furthermore, it means that the software I’m developing must be “simple” and “easy to navigate”. I must make sure that I take the feedback and make sure I include what was said into my program as they are the people that will be using my program.

Research

I will be using secondary market research – by looking at existing systems online of workouts. This is a free and easy way to conduct my research as the internet gives a wide variety of programs and can give my ideas for my program. I can also research what is missing from the current systems which I can then add to my program to make it better than the current system.

Perhaps the most popular web site for gym enthusiasts is www.bodybuilding.com/ where you can find 20,000+ articles about training, supplementation and dieting. It has over 9.5 million members and therefore it was the first place I looked for in terms of research. The screen below shows what you can find on the front page of the website. It lets you go through each step of your details, such as height, age, gender and goal. It then bases your results and finds the most suitable plan for you. This is what I want to achieve through my program. However, when you finally see the workout plans presented to you based on your results as shown in figure 2, there are articles and a lot of reading which is unnecessary especially when someone just wants to find a decent plan without having to read about e.g. who created the program (it’s unnecessary). In my program, I intend to keep it simple and just have the workout routine needed instead of the not needed articles under every workout plan. I think this is a much better solution and www.bodybuilding.com/ gave me an insight of what my software should look like. However, in my program things will be made much simpler, which is what each user wants since they do not want to be reading through thousands of words of articles which will not help them in the gym. Figure 2 is what I do not necessarily want in my program as it does not help Andrew who is not concerned with irrelevant information and wants to just get their workout.

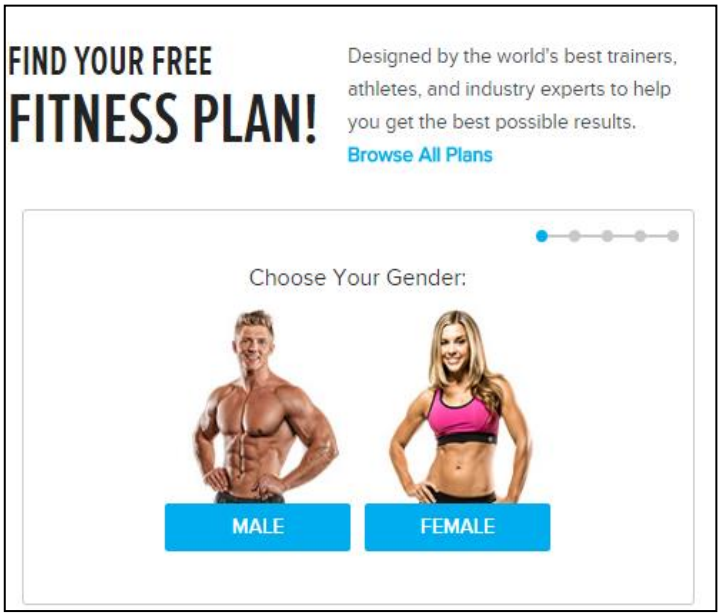


Figure 1a – www.bodybuilding.com/ screen where you input your details

That's how Dave Draper recalled his early perception of Arnold in an interview with Bodybuilding.com in 2008. But all ribbing aside, he and the other Venice Beach boys knew even then that Arnold was on a trajectory all his own.

"We liked him, helped him, taught him by not teaching him, and watched him grow and grow," Draper recalled. "The rumble you heard in the background was bodybuilding in its early stages of take-off. Five, Four, Three, Two, One..."

Now it's your turn to take off. Over the next eight weeks, you're going to train and eat like Arnold in the days when he was forging the physique that introduced bodybuilding to the mainstream. But that's not all. You're going to steep yourself in his legend, in the form of stories, videos, and lore from Arnold and the training partners who knew him best. If you think Arnold's Blueprint is just an arrangement of reps and sets, you need a lesson in what truly made him the greatest of all time.

Figure 2 - http://www.bodybuilding.com/fun/arnold-schwarzenegger-blueprint-trainer-day-1.html an article which is before the actual workout plan

Further Research

The next website I looked at was www.muscleandfitness.com/ and when I looked at the workout plans I realised they had pictures of what the exercise looked like. I would like to implement this into my program as it would give the users (especially beginner users) the chance to see what exercise they are going to perform if they are not sure how to do it. Figure 3 shows this; however, some are missing, which is not suitable if a user does not know how to perform a specific exercise.

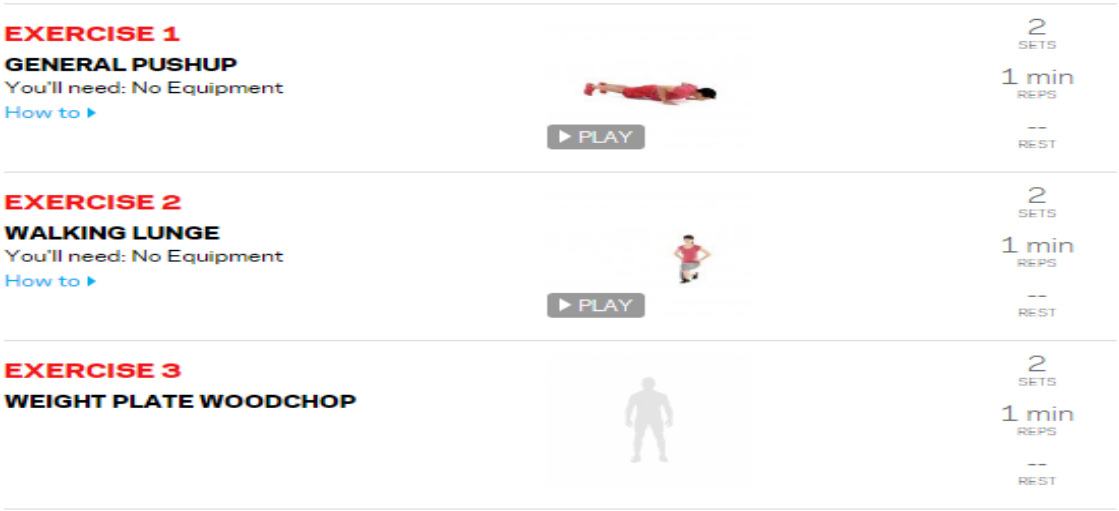


Figure 3 - http://www.muscleandfitness.com/training/routines/ironclad-cardio-program A workout plan with a picture missing

Further Research

The next website I visited was www.exercise.com/ and I realised there was no BMI calculator of any sorts and I think this was a problem as gym users need to be able to calculate their BMI and this meant I had to implement a BMI calculator into my program. Even though the website is focused on workouts I feel that there needs to be a program where there is everything in one. However, this website gave me a lot of ideas for workouts which I could include in my program as they are deemed effective for gym users and written by experts in the bodybuilding/fitness industry. Figure 4 gave me an idea for my program where the user can do check boxes for their selection of certain factors such as if they want a beginner, intermediate or expert level workouts. This helped me to understand the different sections the workout plans can be split into.

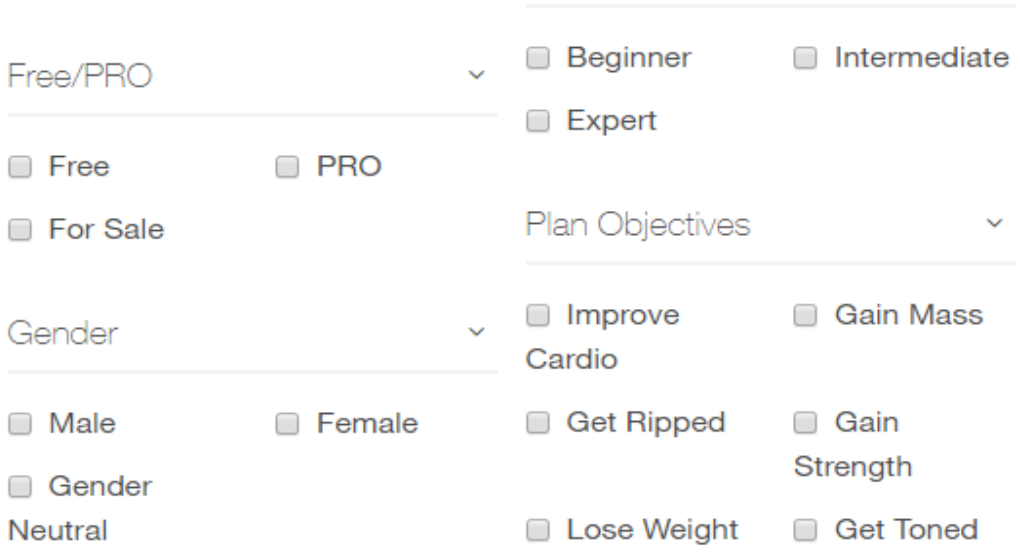


Figure 4 - A menu from the www.exercise.com/ website categorising the different workout plans

Essential features

The essential parts of my system should include a login based system. This means a user should be able to login using credentials and access their account and edit their details. Also, it means that if a new user wants to join they have the option of registering as a new user and filling in details which will eventually lead them to setting up their account. Furthermore, when a user logs in they should be able to choose an option of what they want to do. These options should include: checking to see existing workout plans and adding them to their own workout session which they can then look at and check later; a BMI calculator to see what their Body Mass Index and compare to how healthy they are is; a graph to see the weight loss/gain over a certain period. An important feature that will be included in this feature is the ability to edit your profile, and when this happens your given choice of workouts will also change. These are all important factors that will need to be included in the software I am creating. A password reset feature in case the user has forgotten their password. With these features, navigation and ease of access are important as there are a lot of windows to navigate through. With the BMI calculator, the user can be quick to see what their BMI is and what status their health is.

Limitations

There will be limitations to this program. For example, the users will only have a limited amount of workout plans to choose from which will all be pre-set. There may also be limitations to the amount of personalisation a user can receive for their individual profile. Furthermore, the level of

personalisation will be limited – this takes time to implement and is not one of the core features that is needed in the program.

Proposed Solution

Requirements

The requirements of the system are:

Usability

- Every window will have a back button to go back to the previous window (does not apply to windows where you cannot go back e.g. start screen)
- Buttons must big enough to click easily for navigation and for confirmation where required
- All windows must have pictures relating to the gym to make for an easy to see interface
- The user must never have to scroll down within windows
- See the list of workouts for their goal easily and can select them with a button
- Text labels in bold where the user can register a new account or reset their password
- A custom gym weights icon for each window
- Being able to tab through text boxes instead of clicking each one at a time
- Enter button (only where there are 'confirm buttons') to take user to the next screen
- Combo boxes where there is a finite amount of choices (e.g. for goal a combo box for 3 options to "cut", "build, or "transform")
- A calendar for the user to select the date of their workout
- An incremental number box for weight and height with a certain number of decimal places (e.g. 82.5kg for weight)
- A graph to show differences between weight so the user can see easier
- Labels next to each text box displaying what needs to be input
- Labels at the top of windows to say what the name of the current window is
- Large enough size font to make sure the user can read the labels without zooming in
- Buttons labelled to say what they do when clicked e.g. proceed will take user to next screen whereas update labelled button will update information

Performance

- There must 2 account types – an admin type (only one user can have this type of account), and a user account (all other users)
- The user must able to register for a new account if they do not have one
- The user must able to log in with this new account after registering using the username and password
- The program must store all the details of the user given during registration
- The user must able to reset their password using their username and email
- An email must be sent to the user with the reset password with which they can log in
- The user must able to update their personal details which will then be saved to the database
- The user must able to change their current password and save this to the database
- The user must able to calculate their Body Mass Index by clicking one button

- The BMI calculator must display the weight and height used to calculate the BMI, the BMI value, and what status of health (e.g. overweight) depending on the value.
- A bar graph must be displayed showing the previous weight of the user found in the database (none if previous weight has never been stored) and the current weight
- The admin must able to view details of all the users in the database
- The admin must able to update, delete, add and create new users in the database
- The admin must be able to show all workouts with a name joined from the users table
- The user must be able to select a trainer from the database in a combo box
- The user must be able to select a date from a calendar for their given workout and this input be stored in the database when the workout is booked
- The total must be calculated for the workout using the payment per hour of the trainer multiplied by the duration of the workout and displayed to the user
- The user must be able to book their workout and this be stored in the database
- The user must be able to switch between windows using buttons as a way of navigation
- Workouts must be displayed depending on the goal of the user
- A pie chart for the admin must be displayed when a button is clicked to show the percentage of male to female users.
- The program must store the previous weight of the user if a new weight has been updated to
- The user must be able to log out of the program once finished

Reliability

- The program must hash passwords into the database so that even with access to the database, that person will not be able to access a given account
- If a text box requires password input, this must always be dotted out so that it cannot be seen
- Users may only be able to log in with their password and not any other user's password from the database
- When registering, passwords must match for the user to continue
- Usernames must be at least 5 characters long and can only contain letters and numbers
- When executing SQL queries, question marks must be used to prevent injections by any users
- A user's name may only contain letters
- A user's surname may only contain letters
- The age of a user cannot be below 16 or above 100
- The email of a user must be valid by sending an email and seeing if there is a response
- Weight of the user can only be between 30 and 120kg
- Height of the user can only be between 100cm and 210cm
- Only available genders are male and female and nothing can be written to combo boxes
- Text boxes which are required cannot be left empty
- When booking a workout, a user cannot select a date prior to the current day i.e. cannot book a session for yesterday or any other days before today
- A workout must be chosen for the program to proceed, otherwise the user will not be able to book a session

- When updating user records, a copy cannot be made of the current record, but instead an update query must be used
- When a user views their workouts, they cannot see the workouts of other users

Maintainability

- Comments should be written next to lines of code to make sure it can be understood by other programmers.
- Meaningful variable names should be given
- Should be able to add new users easily as admin or be registering for a new account
- Same formatting to code throughout the program
- Testing to make sure the program does not crash with extensive use

Hardware and software requirements

This is what is needed to run my software on a given computer. The user will need a version of python 3.5 or higher to run the code given. The requirements a computer would need to run Python 3.5.2 are:

- Windows 98 and ME, or later. PyMOL will not run on Windows 95 and NT.
- 3D OpenGL compatible graphics accelerator card.
- 256 MB RAM.
- 500 MHz Pentium 3 processor
- Internet connection

Taken from http://pymol.sourceforge.net/newman/user/S0120install.html

The user will also need a mouse and keyboard to access all the functions of the interface as well as input details. Speakers are not required as there will not be any sound from the software. Most computer systems meet these requirements by far and it means many users will be able to access my software. The monitor resolution required is a minimum of 900x800.

Design

Decomposition of the problem

My gym booking program is a complex program which must be broken down into smaller, manageable parts. The stakeholders will each have their own needs from the program and therefore each function can be broken down for those stakeholders e.g. the login system is for both the administrator and the customer, whereas the BMI calculator function is only for the customer as they are using it. For the customers, the login is a function which reads the usernames and passwords from the database. There will need to be a program which checks for the username and password of a customer. If the username is not found in the database, then an error message will be displayed showing the username was not found in the database. If the username is found the program must make sure the password matches the username e.g. cannot use another persons' password to sign into a different account. If the username is found and the password matches the username given in the username input box, then the customer can log in. For the BMI calculator, there needs to be a formula that calculates the BMI based on weight and height. Depending on the number calculated in the end, there needs to be text which is displayed showing how healthy the customer is e.g. if BMI is high (above certain value), then they are overweight. The customer should be able to book a given date at the gym with a trainer of their choice. Depending on how long they would like to spend at the gym, an hourly rate for trainers is calculated as a total of hours e.g. for 1 trainer the rate could be £5 an hour so 2 hours of workout is £10. A graph should be displayed showing the difference between the previous weight and height and the current weight and height.

First, I will set up the database in SQL. Then, I will hard code the login in Python and make sure checking the username against the password outputs successful login. Then I will read the username and password from the database and check the program still works. Then I will create the GUI for the login window in PyQt with pictures. Then I will take the input into the line edits and make sure all the code works inside the GUI. Then the welcome window will be made and buttons. The buttons will then be linked to the windows and the windows for those buttons will be made. Then I will make the register windows in PyQt and link all the line edits to Python and validate the inputs. Then the workouts window will be made. I will add a picture to the background, add a calendar and buttons that link to the functions. Then the routines windows will be made for all the goals – cut, build transform. All the exercises will be read from the database for this workout. Then the booking window will be made in PyQt where the confirmation for the booking will be made and the total will be displayed. Then a window for the user details will be made and all the user's details will be read from the database and displayed into the line edits. Then the BMI will be tested using the Python console first. Then a window will be made and the BMI will be displayed on the label. Then the graph will be made and shown in a table view in PyQt. Then, the Password reset window will be made and the username and email will be used to reset the password. Then, the workouts window where the workouts for the user will be made in PyQt. It will be linked to Python. Then the admin window will be made and then buttons assigned to functions. Each button will display different labels and then the database will show the data depending on the button e.g. show all users, show all workouts. Then a pie chart will be made to show the percentage of male to female users.

Pseudocodes

Pseudocode for the login

```
CLASS LoginWindow
ATTRIBUTES:
        Username: STRING
        Password: STRING
Input Username
Input Password
C: =OPEN “CUSTOMERS.TABLE”
Customers: =C. READ_TABLE ()
FOR i=0 TO LENGTH (Customers)
        IF Customers[i]. Name=USERNAME_INPUT THEN
                Username_password_list=Customers[i]. Username + Customers[i]. Password
        END IF
NEXT i
IF Username_input =Username_password_list [1] THEN
        IF Username_password_list [2] = Password_input THEN
        PRINT “Login successful”
        HIDE_WINDOW (CURRENT)
        SHOW_WINDOW (OPTIONS)
        END IF
ELSE IF Username_input=Admin_username THEN
        IF Password_input=Admin_password THEN
        PRINT “Login successful”
        HIDE_WINDOW (CURRENT)
        SHOW_WINDOW (ADMIN)
        END IF
ELSE
        PRINT “Incorrect login details”
END IF
```

The reason this works is because the password is in a list when it is read from the text file and if it does match then the user can log in. For example, if the username was olek123 and this was read from the text file it would read the password associated with the username. If the password was hello3 then when python read the data it would show (olek123, hello3) therefore the index of the username+1 is the password because it comes straight after. This the basic pseudocode for the login for my program.

BMI Pseudocode

```
CLASS BmiWindow
ATTRIBUTES:
        Weight: FLOAT
        Height: FLOAT
        Graph Colour: YELLOW
        BMI: FLOAT
        Status: STRING
METHODS:
        Calculate (Weight, Height) RETURN BMI
C: =OPEN “CUSTOMERS.TABLE”
Customers: =C. READ_TABLE ()
FOR i=0 TO LENGTH (Customers)
        IF Customers[i]. Name =Username_input THEN
                User_weight_height_list=Customers[i]. Weight + Customers[i]. Height
        END IF
NEXT i
Weight = User_weight_height_list [1]
Height = User_weight_height_list [2]
BMI = WEIGHT/(HEIGHT*HEIGHT)
BMI = ROUND_NUMBER (2) //Round the number to 2 decimal places
IF BMI <=19.5 THEN
        PRINT “YOUR BMI IS”, BMI,” YOU ARE UNDERWEIGHT”
ELSE IF BMI >19.5 AND BMI <25 THEN
        PRINT “YOUR BMI IS”, BMI,” YOU ARE HEALTHY WIEGHT”
ELSE IF BMI >25 AND BMI<30 THEN
        PRINT “YOUR BMI IS”, BMI,” YOU ARE OVERWEIGHT”
ELSE IF BMI >30 AND BMI <35 THEN
        PRINT “YOUR BMI IS”, BMI,” YOU ARE OBESE”
ELSE IF BMI >35 AND BMI <40 THEN
        PRINT “YOUR BMI IS”, BMI,” YOU ARE SEVERLY OBESE”
ELSE IF BMI >40
        PRINT “YOUR BMI IS”, BMI,” YOU ARE MORBIDLY OBESE”
END IF
```

The BMI is calculated for the user and using ‘If’ statements the program will be able to display the status of your current health e.g. underweight

Registration Pseudocode Window 1

```
Input Username

Input Password

Input Re-entered password

IF password_input="" THEN

        PRINT "Password Box Empty"

ELSE IF length(Username_input) <5 THEN

        PRINT "Username too short"

ELSE IF length(Password_input) <5 THEN

        PRINT "Password too short"

ELSE IF password_input ≠ Retyped_password THEN     // If password input is not equal to retype password

        PRINT "Passwords don't match"

ELSE IF Username_input="" THEN

        PRINT "Username box empty"

ELSE IF ""IS IN Username_input OR Password_input THEN

        PRINT "Spaces not allowed"

ELSE

        HIDE_WINDOW (CURRENT)

        SHOW_WINDOW (REGISTRATION PAGE 2)

END IF
```

This pseudocode is meant for registering a new user. It will make a list of checks in the input boxes and will send a test email to see if the user's email exists. This is important as any password resets later will need to have a working email.

Registration Pseudocode Window 2

```
Input Name
Input Surname
Input Email
Input Postcode
Input Date of birth
Input Weight
Input Height
Input Weight
Gender_input=Combo Box_pick_options ("Male/Female")
Goal_input=Combo Box_pick_options ("Cut/ Transform/ Build")
Age = Current_Date – Date_of_birth
```

```
Once button pressed GO TO FUNCTION CHECKER
        FUNCTION (CHECKER)
                Send Test Email to Email_input
                IF CHECK=FALSE THEN
                        PRINT “Email invalid”
                ELSE IF [a-z] not in Name_input OR Surname_input THEN
                PRINT “Only letters allowed in forename/surname”
                ELSE IF Age>100 OR <16 THEN
                        PRINT “Incorrect age to use this program”
                ELSE
                        C: =OPEN “CUSTOMERS.TABLE”
                        Customers: =C. WRITE_READ_TABLE()
                        J: = LENGTH(Customers)
                        Customers[j+1]. Name: =Name_input
                        Customers[j+1]. Surname=Surname_input
                        Customers[j+1]. DOB=Date_of_birth_input
                        Customers[j+1]. Weight: =Weight_input
                        Customers[j+1]. Height: =Height_input
                        Customers[j+1]. Gender: =Gender_input
                        Customers[j+1]. Goal: =Goal_input
                        Customers[j+1]. Postcode: =Postcode_input
                        NEXT i
                        HIDE_WINDOW (CURRENT)
                        SHOW_WINDOW (LOGIN)
                END IF
```

Admin Pie Chart Pseudocode

```
C: =OPEN “CUSTOMERS.TABLE”
Customers: =C. READ_TABLE ()
Male_count=0
Female_count=0
```

```
FOR i=0 TO LENGTH (Customers)
        IF Customers[i]. Gender: =" Male" THEN
                Male_count=Male_count+1
        ELSE
                Female_count=Female_count+1
        END IF
NEXT i
Total_amount_of_members= Female_count+Male_count
MALE_AREA = ((Male_count / Total_amount_of_members) *360
FEMALE_AREA = (Female_count / Total_amount_of_members) *360
SIZES = [FEMALE_AREA, MALE_AREA]
COLOURS = ['PINK', 'BLUE']
PLOT_PIE_CHART (SIZES, COLOURS=COLOURS)
```

Password Change Pseudocode

```
Input Current password
Input New password
C: =OPEN "CUSTOMERS.TABLE"
Customers: =C. READ_TABLE ()
FOR i=0 TO LENGTH (Customers)
        IF Customers[i]. Username =Username_input THEN
                User_password=Customers[i]. Password
        END IF
NEXT i
IF User_password=Current_password THEN
        User_password=New_password
ELSE
        PRINT "Current Password Incorrect"
END IF
```

Booking Workouts Pseudocode and Selecting Trainers

```
NAME_LIST= []
C: =OPEN "Trainers. TABLE"
```

```
Trainers: =C. READ_TABLE ()
FOR i=0 TO LENGTH (Trainers)
        Name=Trainers[i]. Name
        NAME_LIST_ADD (Name)
NEXT i
ADD_TO_COMBO_BOX (NAME_LIST)
MINIMUM_DATE (CALENDAR) = CURRENT_DATE
```

Picking Workouts Pseudocode

```
C: =OPEN "CUSTOMERS.TABLE"

Customers: =C. READ_TABLE ()
FOR i=0 TO LENGTH (Customers)
        IF Customers[i]. Goal = "Cut" THEN
                SHOW Cut_Plans WINDOW
                IF Plan1Button Clicked THEN
                        SELECT Exercises (Plan1)
                ELSE IF Plan2Button Clicked THEN
                        SELECT Exercises (Plan2)
                ELSE
                        SELECT Exercises (Plan 3)
                END IF
        ELSE IF Customer[i]. Goal=" Build" THEN
                SHOW Build_Plans WINDOW
                IF Plan1Button Clicked THEN
                        SELECT Exercises (Plan4)
                ELSE IF Plan2Button Clicked THEN
                        SELECT Exercises (Plan5)
                ELSE
                        SELECT Exercises (Plan 6)
                END IF
        ELSE
                SHOW Transform_Plans WINDOW
                IF Plan1Button Clicked THEN
                        SELECT Exercises (Plan7)
                ELSE IF Plan2Button Clicked THEN
                        SELECT Exercises (Plan8)
                ELSE
                        SELECT Exercises (Plan 9)
                END IF
        END IF
NEXT i
```

Displaying Total Pseudocode

```
CLASS BookingWindow INHERITS WorkoutsWindow

ATTRIBUTES:

    Payment_rate: FLOAT

    Total: FLOAT

    Duration_of_workout: FLOAT

METHODS:

    CalculateTotal (Duration_of_workout, payment_rate) RETURN TOTAL

IF PROCCED BUTTON CLICKED THEN

    Payment_rate=TRAINERS_DATA [INDEX 2] WHERE TRAINER NAME=COMBO BOX INPUT

    TOTAL = Duration_of_workout * Payment_rate

    SET_LABEL_TEXT (TOTAL)

END IF
```

Algorithms

I will be using an algorithm to search through a nested list. An example for this is using the FOR loop. If we were looking for the second record and the first piece of data inside that record we can output that piece of detail for the record

```
FOR each IN LIST

    FOR x IN each

        OUTPUT (x)
```

This would output all the details one by one of all the records. This can be useful as then another 'if' statement can be run to see if 'x' is equal to a certain criterion e.g.

```
…

    FOR x IN, each

        IF x = "Luke" THEN

        PRINT (INDEX [x])

        END IF
```

Data Dictionary

Exercises

Column Name	Data type	Description	Example	Validation
ExerciseID	Integer	Primary key for this table	4	Only numbers allowed – which are autoincremented
ExerciseDescription	String	Describes the exercise name	Bench Press	Cannot have numbers – only letters allowed

Trainers

Column Name	Data type	Description	Example	Validation
TrainerID	Integer	Primary key for this table	1001	Only numbers allowed – which are autoincremented
TrainerName	String	Name of the trainer	Josh Taylor	First name and surname needed – no numbers allowed
PaymentRates	String	Payment rates per hour for each trainer	7.20	Real numbers only – no letters

Customers

Column Name	Data type	Description	Example	Validation
CustomerID	Integer	Primary key for this table	10001	Only numbers allowed – which are autoincremented
Username	String	Username of the person – stored during registration process	Michael12	Must be at least a length of 5 numbers or letters – no invalid characters allowed
Password	String	Password of the user – created during registration	5afhdsa657dsajvcags	Will be hashed in the database which will be a long string of numbers and letters
Name	String	First name of the user	John	Letters only – no numbers allowed
Surname	String	Surname of the user	Smith	Letters only – no numbers allowed
DOB	String	Stores the date of birth of the user in a	1998/10/04	Given user can only be between

		YYYY/DD/MM format		16 and 100 years of age
Postcode	String	Stores postcode of the user	HP13 6JW	None
Email	String	Stores email of the user (used at password reset)	Odzialak@yahoo.com	Has to have an '@' sign. Cannot have a dot after the sign. When registering this will be checked by sending a test email to the user
Weight	Integer	Stores the weight of the user in kg	80kg	Cannot be smaller than 30kg and bigger than 120kg
Height	Integer	Stores the height of the user in cm	190cm	Only 100cm-210cm accepted
Gender	String	Stores the gender of the user – either female or male	Male	Male or Female
GoalID	Integer	The foreign key from the workout goal table	1	Integers only
PreviousWeight	Integer	Stores the previous weight of the user when they update their profile	58	Integers only – only between 30kg and 120kg

Workout Goal

Column Name	Data type	Description	Example	Validation
WorkoutGoalID	Integer	Primary key for this table	9001	Only numbers allowed – which are autoincremented
WorkoutGoalDescription	String	Name of the goal the user is going for	Build	Can only be build, transform or cut

Workouts

Column Name	Data type	Description	Example	Validation
WorkoutID	Integer	Primary key for this table	1000001	Only numbers allowed – which are autoincremented
CustomerID	Integer	Foreign key in this table from the Users table	10001	Integers only
TrainerID	Integer	The ID for the trainer from the trainer's table	1001	Integers only
DateBooked	String	Date of the workout	05/10/2016	Only dates

		sessions stored in the format DD/MM/YYYY		stored as string separated by a '/' allowed
ExercisePlanID	Integer	Foreign key from the ExercisePlanList	1001	Numbers only
Price	Float	Shows the total price of the booked workout	7.2	Floats only – will be converted later to string but stored in database as float

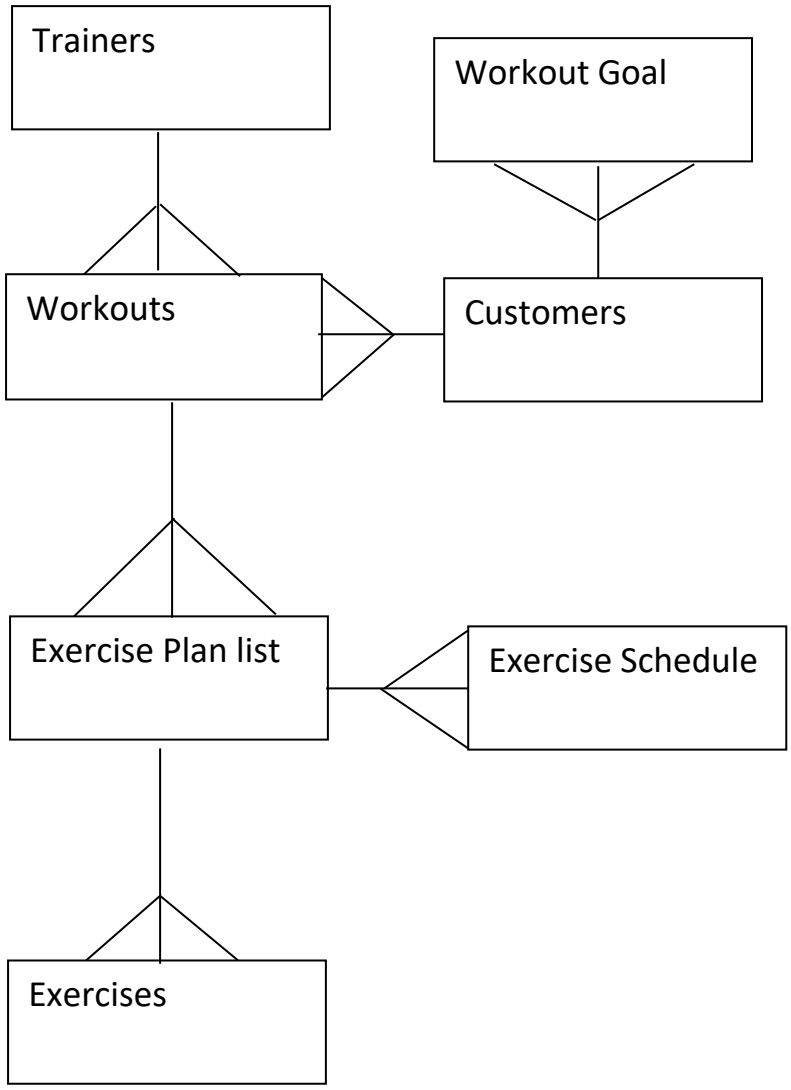
ExercisePlanList

Column Name	Data type	Description	Example	Validation
ExercisePlanID	Integer	Foreign key in this table from the Users table	10001	Only numbers allowed – which are autoincremented
ExercisePlanName	String	Name of the plan give to a set of exercises	Plan to building big muscles	Letters only – no invalid characters

ExerciseSchedules

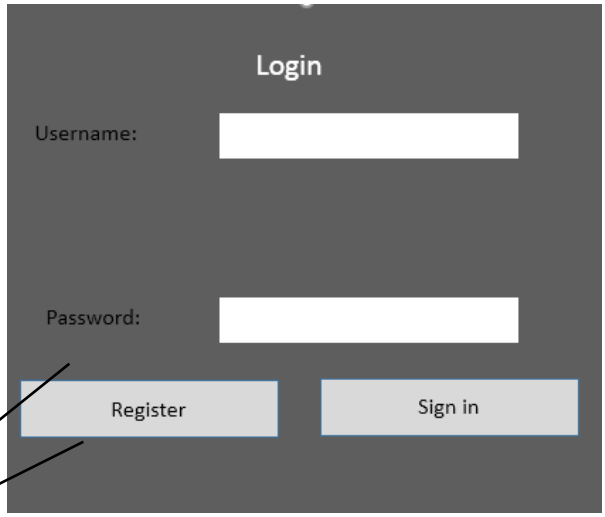
Column Name	Data type	Description	Example	Validation
ExerciseScheduleID	Integer	Primary key for this table	101	Only numbers allowed – which are autoincremented
ExercisePlanID	Integer	Foreign key from the exercises table	1001	Integer only
ExerciseID	Integer	Foreign key from the exercises table	10001	Integer only

Entity Relationship Diagram



Usability Features

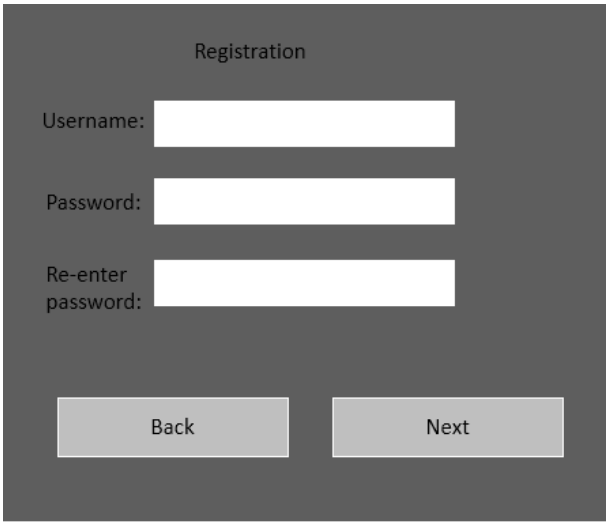
Login screen:



This is the initial login screen I made in PyQt4. The inputs for this screen are the username and password. The text goes into the white boxes where the input is read from. The username is unique and the user cannot login if the password does not match the username. The register button takes the user to the register window and the sign in button takes the user to the welcome screen with all the options. Also, if the username does not exist in the database, the user must register for a new account

Buttons with labels for ease of access for users. Text boxes labelled so the user understands what input is required in each box

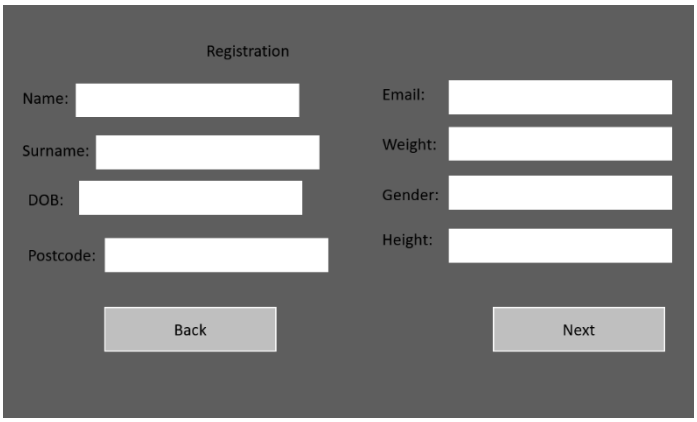
Registration screen:



If the user clicks the register button, they will be taken to this screen. The inputs in this screen are the username, password and re-enter password input boxes. The password and re-enter password must match for the user to proceed. The back button will take the user back to the initial login screen and the next button will take them to the next page of registration.

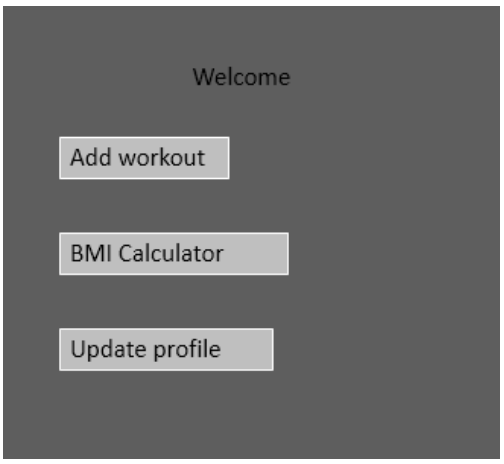
Registration screen page 2:

This is the next page of the registration screen. This has all the user's details like name, surname, date of birth (DOB), postcode, mobile, weight, gender and height. Once all the details have been filled in by the user, and the next button is clicked, it will save the users details into the database using SQL. These details will then be later retrieved if needed, such as weight and height to calculate the BMI. When the next button is clicked and the details have been saved, it will take the user back to the login screen.



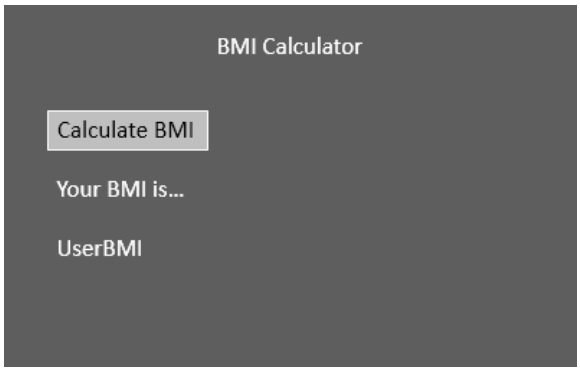
Welcome screen:

This is the screen users see when they login. It has three buttons – add workout, BMI calculator and update profile. Each one will take the user to a different screen



BMI Screen:

This is the BMI screen. BMI is Body Mass Index which shows how healthy a person is based on their height and weight. When the user clicks the calculate BMI button, their BMI is displayed on screen and depending on what the user's BMI is, a label is displayed showing if they are healthy weight, underweight, or overweight. The UserBMI text will be replaced with the user's BMI.

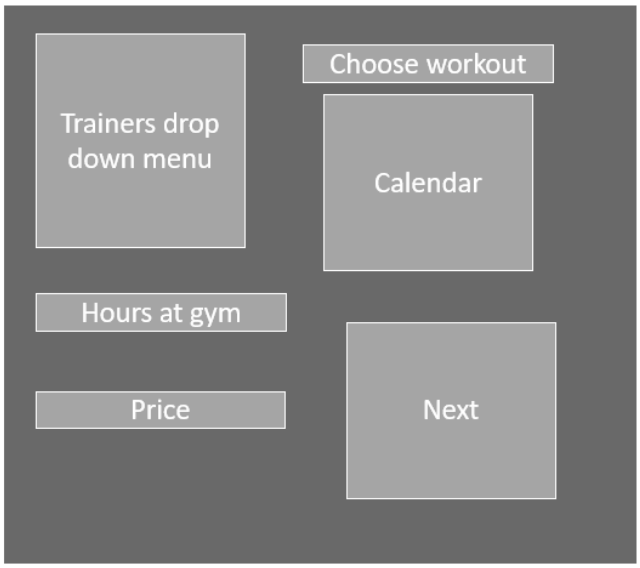


Update profile screen:

The user's details are displayed and can be updated. This is important since the weight or height might have changed, which will impact the calculation of BMI.

Book session screen:

This is the main part of the program. The user can book a session at the gym. They can choose the trainer, number of hours, the workout they want and what date. The price is displayed at the bottom left of the screen.



Inputs/variables

Input	Purpose	Example of input	Validation
Username input box	This takes in the user's username. This is then checked to see if it exists in the database. This is unique for each user and 2 users cannot have the same username.	username	No invalid characters allowed such as $, £. The input box cannot be left empty and the length of the username cannot be longer than 15 characters long.
Password input box	This input box takes in the password of the user. In the database, these passwords are hashed and the actual password is never stored. This is then compared to the username of that password so the user can log in.	Wind123	When registering, the user must re-type the password and they must match for the user to continue registration. Must be at least 5 characters in length. Cannot be left empty
Name	Takes in the name of the user during the registration	Olek	20-character max, and only letters allowed. Box is not allowed to be left empty
Surname	Takes in the surname of the user during registration	Dzialak	20-character max, and only letters allowed. Box is not allowed to be left empty
DOB	This is the date of birth of the user. This can be used to display popular workouts for a certain age group. The date of birth will be used to calculate the age of the user.	01/01/1998	User cannot be older than 100 years old and cannot be younger than 16.
Postcode	This is the postcode of the user.	HP13 6WJ	Input cannot be longer than 12 characters long. Numbers and letters allowed and space but no invalid characters ($, £). Can be left empty
Email	The user's email is stored in case they forgot their password an email resetting it can be sent to them	olek@gmail.com	Must have an '@' sign. At least one dot and no dot after the '@' sign. Cannot have 2 '@' signs. Cannot be left empty
Weight	The weight of the user which can be used to calculate the respective user's BMI	72kg	Cannot be higher than 120kg and lower than 30kg.
Gender drop combo box	Takes the user's gender. This is used for future reference to pick workouts for each gender	Male	Can only be male or female; drop-down box from which the user selects this
Goal	The user must pick what their goal is at the gym – is it build, cut, or transform. This will then determine which exercise plans are picked for this user	Cut	Can only be cut, transform or build. There will be a drop-down box from which the user selects this
Height	Takes the height of the user in cm's. This is also used to calculate the BMI	170cm	Cannot be lower than 100cm and higher than 210cm.

Calendar	Takes the date that the user wants to book for their workout session. It is then stored in the database and the user can view it later	05/02/2016	Cannot be anything before the current day.
Trainers list	This the list of trainers read from the database. This list is then inserted into the trainer name combo box	(Josh Taylor, James East)	Pre-set trainers from the database: the user cannot add their own trainers
Trainer name combo box	Displays all the names of the trainers in a combo box from which the user can pick from	Josh Taylor	Nothing can be input into the combo box by the user. A trainer must be selected for a workout
Exercise Plan button	The user can press a button to choose the corresponding exercise plan for their workout	Button clicked with label 'Plan to build muscle'	The user must select an exercise plan before they can book a workout
Book workout button	When clicked, all the necessary details of the workout are written to the database	Button clicked with label 'Book'	Once this button is pressed the user will be taken back to the welcome screen so it cannot be clicked multiple times in rapid succession to prevent the program from crashing
Calculate BMI button	Once clicked by the user the BMI, weight, height and health result is displayed on a label	Button clicked with label 'Calculate'	When it is pressed once, any more clicks won't reset any labels on screen
BMI	Takes the weight and height of the user and using the BMI formula (weight / (height^2)) calculates the BMI value of the user	24.33	BMI value will be rounded to 2 decimal places. Data type is a float
Updated Name	If the user ever changes their name, they can change this in the update window of the program	Andrew → Andy	20-character max, and only letters allowed. Box is not allowed to be left empty
Updated Surname	Again, if the user ever changes their surname, they can update this also	Hersh → Wilkins	20-character max, and only letters allowed. Box is not allowed to be left empty
Updated Postcode	If the user ever changes place of residence, they can edit this in the update window. If the user also does not want their postcode stored anymore they can remove it	HP12 4PB → HP13 6UW	Input cannot be longer than 12 characters long. Numbers and letters allowed and space but no invalid characters ($, £). Can delete this
Updated Email	The updated email of the user if they ever change it or if the old one does not work anymore	odzialak@yahoo.com → olek@gmail.com	Must have an '@' sign. At least one dot and no dot after the '@' sign. Cannot have 2 '@' signs. Cannot be empty
Updated Weight	The updated weight of the user if they lose or gain weight. When this is updated the old weight of the user is stored as the Previous Weight if there is a difference	82kg → 85kg	Cannot be higher than 120kg and lower than 30kg.
Updated Gender	If the user wants to update their gender – perhaps they did not pick	Male → Female	Can only be male or female, there will be a drop-down box

	the correct one during registration		from which the user selects this from
Updated Goal	If the user decides to switch their goal e.g. no longer wants to transform, but only build muscle	Cut → Build	Can only be cut, transform or build. There will be a drop-down box from which the user selects this
Update details button	This will update the current user details with the updated variables and a message box will be displayed saying 'Details updated'	Update button clicked	This will not use the INSERT query in SQL. It will use the UPDATE query to prevent from multiple records existing which are the same
Current Password	If the user wants to change their current password to a different one, they must enter the current password and this must match the one from the database assigned with their username	Olek123	This if hashed first before it is checked against the password in the database
New Password	If the user wants to update their password, this will be the new password	Olek321	This will be hashed before saved to the database

Widgets, variables, lists, classes

Type	Explanation	Validation
Login class	The class for the fist login window where all the methods and attributes are contained	Making sure all functions are within the class and making sure the user can proceed to other windows
Username line edit	This is where the user will input their username into a line edit (this is not the variable that it is stored as)	Making sure the username matches the one from the database. Storing this input as a variable (refer to table above)
Password line edit	This is where the user will input their password into a line edit (this is not the variable that it is stored as)	Making sure the password matches the one from the database. Storing this input as a variable (refer to table above)
Help button label	This is the label in PyQt that will display the help message inside the login window	Making sure that when pressed a help dialog box is displayed
Reset label	This is the label when pressed takes the user to the password reset window	Making sure the user is taken to the reset window. The user must press the area of the label to make the button register within the program
Register label	This is the label which will take the user to the register window	Making sure that the user is taken to the register window upon pressing the button. User must press the area of the label to make the button register within the program
Login button	Once pressed by the user they will be able to log in and the welcome window will be shown to them	Can only log into the system if the details are correct
Username and password list	This is the list read from the database/ text file which will show the username and password	Will only contain 2 strings – the username at index 1 and password at index 2.
Welcome class	The class for the welcome window where all the methods and attributes are contained	Making sure all functions are within the class and making sure the user can proceed to other windows
Add new workout button	The button which takes the user to the workouts window	Making sure the user is taken to the workout window upon pressing the button
BMI window button	The button which takes the user to the BMI calculator window	Making sure the user is taken to the BMI calculator window upon pressing the button
Update profile button	The button which takes the user to the update profile window	Making sure the user is taken to the update profile window upon pressing the button
My workouts button	The button which takes the user to the my workouts window	Making sure the user is taken to the my workouts window upon pressing the button

Type	Explanation	Validation
Log out button	Button which logs the user out and takes them back to the login window	Making sure the line edits are cleared and the user is taken back to the login window when confirmed
Workouts class	The class for the workouts window where all the methods and attributes are contained	Making sure all functions are within the class and making sure the user can proceed to other windows
Trainers names combo box	The trainer name combo box from where the user can take a drop down of names of trainers to pick for their workout	Can only select one trainer for each workout
Booking workouts class	The class where the user confirms their booking and all methods and attributes are contained within	Making sure the user can book their workout within this class
Back button	The button which takes the user to the previous window	Making sure the user is taken back to the previous screen, not a different one
Proceed button	This takes the user to the next window in order	Making sure all the details are filled in necessary before proceeding onto next window.
Book workout button	This confirms the booking of the user with a message box once pressed	Making sure the workout is saved into the database when button is pressed.
Admin class	This is the class where the admin can see all the user's details, and make changes and where the methods and attributes are contained	Making sure the admin can view all the user's details from the database in the GUI.
Home button	Takes the user to the login window once pressed	Making sure the user is taken back to the login screen upon pressing the button
Graph button	Displays a graph for the admin of male and female users as a pie chart when pressed	To make sure the calculations are correctly calculated for the angels of the pie chart.
Password reset class	This is the class where the email is reset for the user and where the methods and attributes are contained	Making sure the username corresponds to the email in the database.
User details class	This is the class where the user details are updated and where the methods and attributes are contained	Making sure the user can update their details in this class
Trainers details list	A list which contains the names, payment rate and ID of the trainers	Making sure the list is read from the trainers table in the database

Test plan

Test no.	Example of input	Relation to success criteria	Expected output	When tested
1	Back button pressed (normal data)	User should be able to go to back to previous screen from any screen (apart from home screen)	Program will not crash – user will be taken to previous screen	Throughout development stage and post development
2	Back button pressed 10 times (extreme data)	The user should be able to keep going back quickly through the windows	Program should not crash – user should be able to keep going through screens	Throughout development stage and post development
3	Check all windows have pictures relating to the gym (normal data)	An easy and visible interface for the user	All windows have a picture	Post development
4	Check if there are any scroll bars within any windows (normal data)	The user should never have to scroll down within any window	To make sure the user doesn't scroll at all through any windows	Post development
5	Check the list of workouts and see if they can be selected with a button (normal data)	User must be able to select their workout to book it	User must be able to select their workout and these workouts must be based on their goal	Throughout development stage and post development
6	Check the text labels are in bold for registering a new account and resetting password (normal data)	The registration and reset password labels must be in bold and when clicked must take user to the window	To make sure the labels are visible and can navigate the user to the reset password/ register windows	Throughout development stage and post development
7	Check each window has a custom gym weights icon (normal data)	Each window must have an icon with gym weights	To make sure the interface is user friendly and makes the program unique	Post development
8	Press 'tab' button in each window and see if taken in order of appearance of text boxes/ inputs (normal data)	User must be able to tab through input boxes in order	To make the interface easy to use for the user	Post development
9	Enter button pressed to confirm certain inputs on	User must be able to press enter button to confirm	To make the interface easy	Post development

	screen (normal data)	certain inputs	to use for the user	
10	Click on a calendar date to select the workout (normal data)	Check this date is then saved to the database	To make sure the correct date gets saved for a workout	Throughout development stage and post development
11	Check the weight and height number box increments when pressed (normal data)	An incremental box must be used for the weight and height input	To make it easy for the user to pick a decimal number	Throughout development stage and post development
12	Check the graph displays the last weight and the current weight of the user (normal data)	To make sure the user can see the difference easily and visually aid them using a graph	Make sure the graph displays the correct values and is the correct size on the window	Throughout development stage and post development
13	Check there is a label next to every single input box (normal data)	Labels must be shown next to each text box displaying what must be shown	Making sure the user knows what data to input into the input box	Post development
14	Go into each screen and check for label (normal data)	Labels at the top of windows to say what the name of the current window is	Each screen has a label	Throughout development stage
15	Go into each screen and check label is big enough font (ask Andrew) (normal data)	Labels must be large enough without the need of the user zooming in	Each screen's label font is large enough	Throughout development stage
16	Go into each window and check each button is labelled (normal data)	Buttons must be labelled to say what they do when clicked	Each button is labelled correctly	Throughout development stage
17	Log in as admin and log in as a normal user (normal data)	There must only be 2 account types – admin (only one person has access to) and normal user	Log in as user and taken to welcome screen, log in as admin and taken to admin window	Throughout development stage and post development
18	Log in using admin username and user's password (erroneous data)	Can only login as the type of user if the account is assigned to that level i.e. admin details for admin login	Program rejects the input and displays error message	Post development
19	"Godwin" (normal data)	User must be able to register for a new account if they do not have one	Details accepted – user can be registered	Throughout development stage and post development

20	"godwin2" "league" (normal data)	User must be able to log into their account	User can login with the username and password	Throughout development stage and post development
21	"godwin" "wind123" (erroneous data)	User can only log into their account using only their credentials	Display error message "incorrect details"	Throughout development stage and post development
22	Check if data has been written to the database after user has registered (normal data)	Program must store all the details of the user given during registration	Data changed in database	Throughout development
23	Reset password using email and password of user (normal data) "godwin2" "godwincherian@gmail.com"	The user must be able to reset their password using username and email	Reset password reset sent to email	Throughout development stage and post development
24	Reset password using email (which is not in the database) and username of different user (erroneous data) "godwin2" "o@gmail.com"	User cannot be sent an email unless they have registered it during the registration stage and cannot use another user's username to do so	Email not sent, password not reset	Throughout development stage and post development
25	Update personal user details (normal data) "Godwin" → „Alex" „Cherian" → „Jones"	User must be able to update their personal details which will then be saved to the database	User details updated to database	Throughout development stage and post development
26	Update personal user details (erroneous data) – updating name with just numbers "Godwin" →"12345"	The same validation from the registration screen should be taken and invalid data should not be updated	Error message displayed	Throughout development stage
27	Change current password (normal data) "league" →"hi321"	To make sure this password is checked for strength and saved to the database	Updated password in database	Throughout development stage and post development
28	Change current password (erroneous data) "league" → „^$£"	To make sure passwords with invalid data are not stored	Error message - password not accepted	Throughout development stage
29	BMI button pressed (normal data)	The user must be able to calculate their Body Mass Index by clicking one button	BMI displayed on screen along with weight and height	Throughout development stage and post development
30	BMI button pressed 10 times (extreme data)	The user must be able to press it many times without the program crashing	Label only displayed once – no crash	Throughout development stage
31	Go into BMI calculator and	To make sure the user	All labels	Post

	check all labels are displayed	knows which values were used to do the calculations	displayed in same font and size	development
32	Check the admin can view all the details of all the users in the database (normal data)	To make sure that any changes to these users can be made	Admin can view all user's details	Throughout development stage and post development
33	Check if table uses inner join function (normal data)	To make sure the inner join function works and the table is displayed for the admin	Inner join shown in table view	Throughout development stage and post development
34	Trainer picked from combo box (normal data)	To make sure a trainer can be picked for a workout	Stored as a variable	Throughout development stage and post development
35	No trainer picked from combo box (erroneous data)	To make sure the program rejects no input in the combo box	Program doesn't let user proceed	Post development
36	Check the payment is displayed as a label (normal data)	To make sure the user knows the total price and this also gets stored into the database	Payment is displayed as label	Throughout development stage
37	Click the book workout button (normal data)	To make sure all workouts which are booked are stored in the database	The workout is written to the database	Throughout development stage and post development
38	Click the book workout button 10 times (erroneous data)	To make sure bookings arent repeated many times	Program only books one session	Post development
39	Check each window has a button to go back and next (if applicable)	To make sure the user can navigate through the program	Each window has back and next button	Post development
40	Check workouts are for cut workout (normal data)	To make sure the workouts are picked for the goal e.g. Cut workouts for cut goal not build workouts for cut goal	Workout displayed for cut routine if goal is cut	Throughout development stage
41	Click the pie chart button (normal data)	To make sure when a female or male is added the percentages change and therefore the pie chart	The male to female percentage is shown	Throughout development stage and post development
42	Log out button pressed (normal data)	User must be able to log out of the program	Once the button is pressed, the user is taken back to the	Throughout development stage

			login screen	
43	Check passwords in database and see if hashed (normal data)	Passwords must be hashed in the database to prevent a person logging into an account even with access to the database	Passwords hashed	Throughout development stage
44	Input password (normal data)	To make sure any input boxes which require password input are always 'echoed out'	Password blacked out of view	Throughout development stage
45	Input password on register screen (normal data)	To make sure the passwords match when re-entering the password	Password should be stored at registration	Throughout development stage and post development
46	„league" password „hi321" re-enter password (erroneous data)	To make sure the passwords have to match	Error message displayed and input rejected	Throughout development stage
47	"godwin" entered into username box (normal data)	Usernames must be at least 5 characters long and can only contain letters and numbers	Accepts the username	Throughout development stage and post development
48	"Godwin65" entered into username box (extreme data)	Make sure the program accepts capital letters, numbers	Username is accepted	Throughout development stage
49	"Godw14u!%" entered into password box (erroneous data)	Make sure the program rejects any invalid characters for the username	Username rejected	Throughout development stage
50	Check all SQL queries contain question marks (normal data)	This is to prevent SQL injection	All SQL queries contain question marks	Throughout development stage
51	"olek" entered into name box (normal data)	To make sure names only contain letters	Accepted as valid name	Throughout development stage and post development
52	"Olek" entered into name box (extreme data)	To make sure capital letters are also accepted	Accepted as valid name	Throughout development stage
53	"124412" entered into name box (erroneous data)	To make sure the program rejects anything that is not letters	Rejected input – error message	Throughout development stage
54	Input date of birth to make the age 17 (normal data)	To make sure the program accepts any age between 16 and 100	Accepts age as valid	Throughout development stage
55	Input date of birth to make the age 101 (erroneous data)	To make sure the program rejects any ages below 16 and above 100	Rejects this age -error message	Throughout development stage
56	Type in valid email "odzialak@gmail.com" into	The program should validate real email	Accepts email as valid	Throughout development

	email box (normal data)	addresses		stage
57	Type in invalid email “olek35156@@g..co” (erroneous data)	The program should reject and invalid emails	Rejects email – error message	Throughout development stage
58	Type in weight of user as 35kg (normal data)	To make sure the weight is between 30kg and 120kg	Accepts the weight as valid	Throughout development stage
59	Type in weight of user as 10kg (erroneous data)	To make sure the program rejects any weight below 30kg and any above 120kg	Rejects this weight – error message	Throughout development stage
60	Type in height of user as 120cm (normal data)	To make sure the height is between 100cm and 210cm	Accepts this height as valid	Throughout development stage
61	Type in height of user as 10cm (erroneous data)	To make sure the program rejects any height below 100cm and any above 210cm	Rejects this height – error message	Throughout development stage
62	Check the only available gender options are male and female (normal data)	No other input into combo boxes is allowed	Only inputs into combo box are female and male	Throughout development stage
63	Input boxes which are required to have input checked if empty (erroneous data)	These input boxes cannot be left empty	Rejected – error message saying “cannot be empty”	Throughout development stage
64	Choose date for workout – today’s date (normal data)	A date must be picked for a workout	Program accepts date as valid	Throughout development stage
65	Choose date for workout – yesterday’s date (erroneous data)	A date cannot be picked before today	Rejects date – cannot be picked	Throughout development stage
66	Choose workout (normal data)	A workout must be picked to book a session	Program accepts workout	Throughout development stage
67	Proceed without choosing a workout (erroneous data)	Cannot proceed without selecting a workout	An error should be displayed telling the user to pick a workout	Throughout development stage
68	Check an update query is used every time the user updates their details (normal data)	A copy cannot be made of the current record	Update query shown each time details updated	Throughout development stage and post development
69	Check the user can only view their workouts (normal data)	To make sure the user cannot see other user’s workouts	Only user’s workouts shown	Throughout development stage
70	Check there are comments written next to all functions and loops (normal data)	To make the code maintainable for other users	Each loop has a comment next to it	Throughout development stage and post

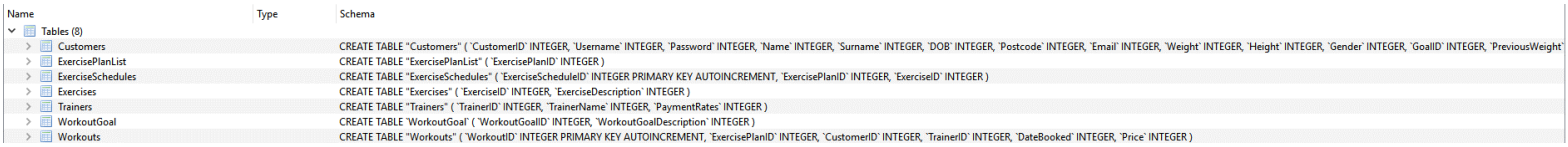
				development
71	Check all variable names (normal data)	To make the code maintainable for other users	Each variable has a meaningful name	Throughout development stage

Development

Iterative development

Database

First I set up the database using SQL. This is because it can be easily used in Python and SQL queries can be run within the programming language



Name	Type	Schema
Tables (8)		
Customers		CREATE TABLE "Customers" (`CustomerID` INTEGER, `Username` INTEGER, `Password` INTEGER, `Name` INTEGER, `Surname` INTEGER, `DOB` INTEGER, `Postcode` INTEGER, `Email` INTEGER, `Weight` INTEGER, `Height` INTEGER, `Gender` INTEGER, `GoalID` INTEGER, `PreviousWeight`
ExercisePlanList		CREATE TABLE "ExercisePlanList" (`ExercisePlanID` INTEGER)
ExerciseSchedules		CREATE TABLE "ExerciseSchedules" (`ExerciseScheduleID` INTEGER PRIMARY KEY AUTOINCREMENT, `ExercisePlanID` INTEGER, `ExerciseID` INTEGER)
Exercises		CREATE TABLE "Exercises" (`ExerciseID` INTEGER, `ExerciseDescription` INTEGER)
Trainers		CREATE TABLE "Trainers" (`TrainerID` INTEGER, `TrainerName` INTEGER, `PaymentRates` INTEGER)
WorkoutGoal		CREATE TABLE `WorkoutGoal` (`WorkoutGoalID` INTEGER, `WorkoutGoalDescription` INTEGER)
Workouts		CREATE TABLE "Workouts" (`WorkoutID` INTEGER PRIMARY KEY AUTOINCREMENT, `ExercisePlanID` INTEGER, `CustomerID` INTEGER, `TrainerID` INTEGER, `DateBooked` INTEGER, `Price` INTEGER)

Login Window

The first thing I decided to do was a login window. I did this using Python with using just the console without a GUI first.

```
username="Olek"
password="hi123"
username_string=input("Enter the username")
password_string=input("Enter the password")
if username==username_string:
    if password==password_string:
        print("Login successful")
    else:
        print("Login failed")
else:
    print("Login failed")
```

I hard coded the username and password in the program by setting the username as Olek and password as hi123. Then username and password is taken as input. Then this input is checked against the username and password. If the username is equal to the username input by the user, the password input is checked against the password. If this is successful, then a message is output saying login successful, otherwise a message is output saying login failed

```
Enter the usernameOlek
Enter the passwordhi123
Login successful
>>>
```

Figure 6 Successful attempt using correct username and password

```
Enter the usernameOlek13412
Enter the password1234124
Login failed
>>>
```

Figure 5 Failed attempt using different username and password

The next part was to get details of users from the database and remove the hardcoded username and password. I imported sqlite3 into python since I am using an SQL browser to create tables and store the data. The database is called “GymBookings” and db (database) is the file type.

```
import sqlite3
con = sqlite3.connect('GymBookings.db')
cur = con.cursor()
```

Conn.cursor() navigates through the database and acts as a positioning tool for this data.

I then ran an SQL statement to read the username and password of the user where the username is equal to the username input. This is then stored as user_details_list.

```
username_string=input("Enter the username")
password_string=input("Enter the password")
cur.execute("""SELECT Username, Password FROM Customers WHERE Username=?""",(username_string,))
user_details_list=cur.fetchone()
print(user_details_list)
```

This made the overall code look like this:

```
import sqlite3
con = sqlite3.connect('GymBookings.db')
cur = con.cursor()
username_string=input("Enter the username")
password_string=input("Enter the password")
cur.execute("""SELECT Username, Password FROM Customers WHERE Username=?""",(username_string,))
user_details_list=cur.fetchone()
print(user_details_list)
if username==username_string: #An if statement comparing username input of user and username in database
    if password==password_string:
        print("Login successful")
    else:
        print("Login failed")
else:
    print("Login failed")
```

Running the code gives the following output:

```
---
Enter the usernameOlek
Enter the passwordhi123
None
Traceback (most recent call last):
  File "C:\Users\oleks\OneDrive\Documents\Coursework tables pyQt\Program.py", line 9, in <module>
    if username==username_string: #An if statement comparing username input of user and username in database
NameError: name 'username' is not defined
```

The error here is that there are no records in my database with a username Olek. When the user_details_list is printed, the output is None i.e. does not exist and therefore the program cannot compare an empty list to an input. Therefore, I need to add the username Olek and password hi123 into my database. Also, the username is not defined and this means nothing is set to this variable.

Table: Customers

New Record | Delete Record

	CustomerID	Username	Password	Name	Surname	
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Olek	hi123	NULL	NULL	NULL

Also, when the program reads these details, it will display this as a list and not a string. Therefore, user_details_list must be indexed and the username and password must be set from this.

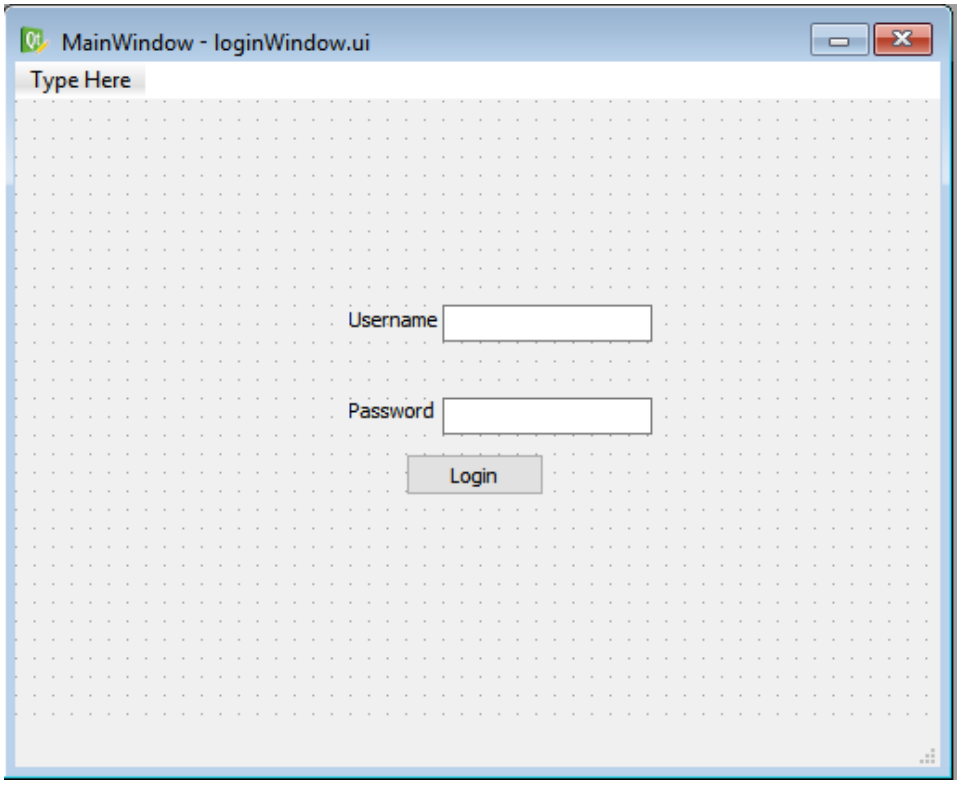
```
print(user_details_list)
```

```
username=user_details_list[0]
password=user_details_list[1]
```

The code gets changed to separate the list into two variables – the username and password from the database. Running the code and using the correct login gives the following:

```
---
Enter the usernameOlek
Enter the passwordhi123
Login successful
```

Now we must make a GUI for this using PyQt4



I have made two labels for each of the line edits where the input will go and a login button to confirm this input

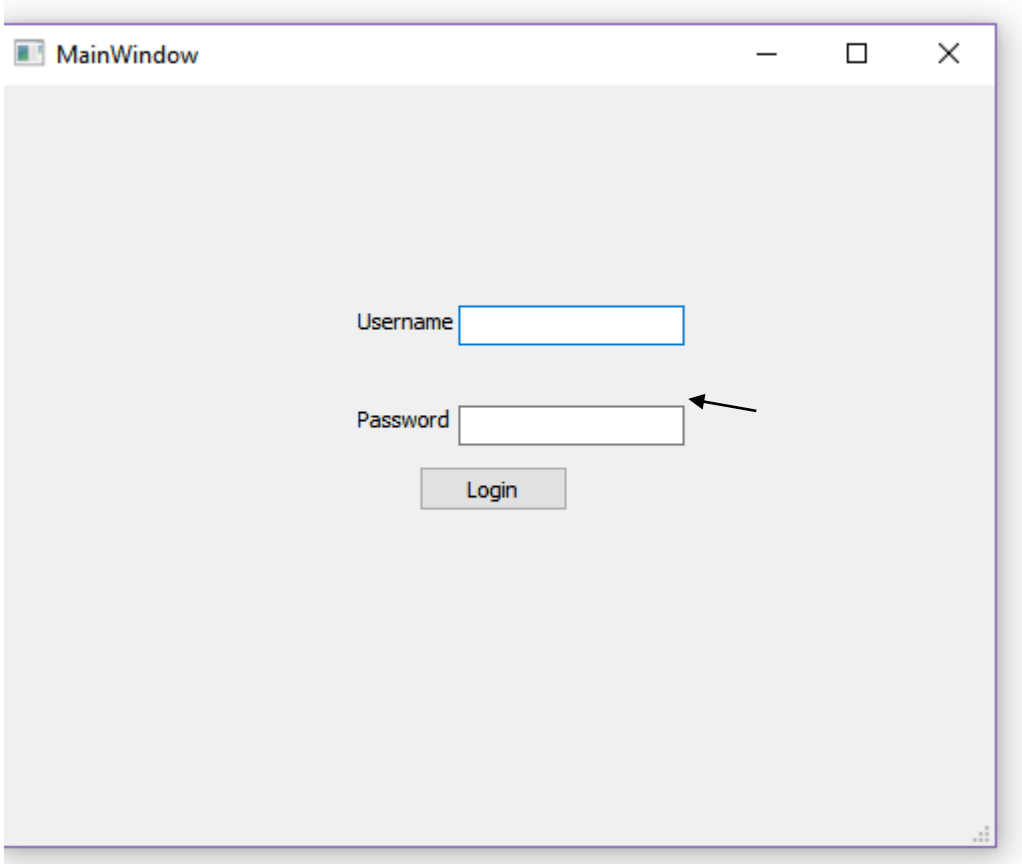
```
import sys, os

from PyQt4 import QtCore, QtGui, uic
from PyQt4.QtCore import *
from PyQt4.QtGui import *
```

We must import the necessary PyQt4 libraries into python to make sure the windows can be loaded. Next, we will need standard 'boiler plate' code to setup the windows within python

```
login_win = uic.loadUiType("loginWindow.ui")[0]
class FirstWindow(QtGui.QMainWindow, login_win):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)

app = QtGui.QApplication(sys.argv)
login_window = FirstWindow(None)
login_window.show()
app.exec_()
```

The reason this is called ‘boiler plate’ code is because there is no need to change it. It says the same for every window setup and is used by everyone in the same way. When run, the program asks us again for the username and password in the console, and the window is displayed after this has finished

```
Enter the usernameOlek
Enter the passwordhi123
Login successful
```



Now we need to change the username and password input to get inputs from the line edits.

```
self.username_string=self.username_input.text()
self.password_string=self.password_input.text()
```

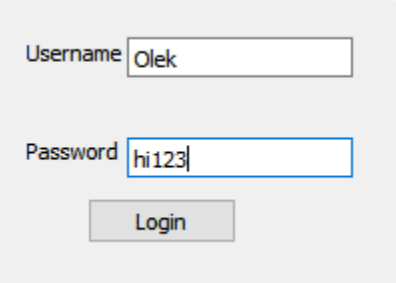
This takes the input from the line edits and sets them as two variables ‘self. password’ and ‘self. username’.

```
self.loginBtn.clicked.connect(self.login)
```

This line will connect the button. Once the button is pressed it will take us to the function login. The self means this function is within this window i.e. login_window.login. The code for the login window looks like this so far:

```
import sqlite3
import sys,os
from PyQt4 import QtCore, QtGui, uic
from PyQt4.QtCore import *
from PyQt4.QtGui import *
con = sqlite3.connect('GymBookings.db')
cur = con.cursor()
login_win = uic.loadUiType("loginWindow.ui")[0]
class FirstWindow(QtGui.QMainWindow, login_win):
    def __init__(self, parent=None): #Sets up the UI
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.loginBtn.clicked.connect(self.login)
    def login(self): #This is the function for when the login button is pressed
        self.username_string=self.username_input.text()
        self.password_string=self.password_input.text()
        cur.execute("""SELECT Username, Password FROM Customers
                    WHERE Username=?""",(self.username_string,))
        user_details_list=cur.fetchone()
        username=user_details_list[0]
        password=user_details_list[1]
        if username==self.username_string: #An if statement comparing username inp
            if password==self.password_string:
                print("Login successful")
            else: #If the password is incorrect login failed
                print("Login failed")
        else: # If username is incorrect login failed
            print("Login failed")

app = QtGui.QApplication(sys.argv)
login_window = FirstWindow(None)
login_window.show()
app.exec_()
```

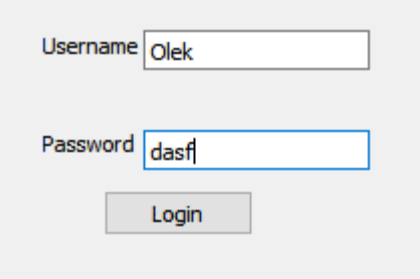
This input for the username and password:



Gives this output:

```
Login successful
```

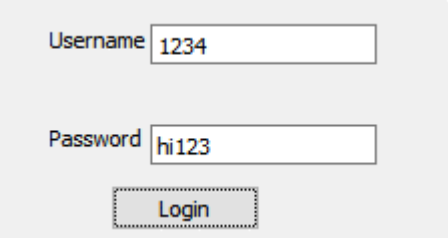
This input for the username and password



Gives this output:

```
Login failed
```

The messages displayed are what we expected. If the username and password input both match the one in the database, then login is successful, and if the password is incorrect, login fails. However, when the username is wrong this is the error displayed

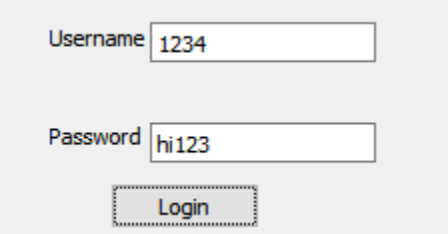



```
Traceback (most recent call last):
  File "C:\Users\oleks\OneDrive\Documents\Coursework tables pyQt\Program.py",
ne 20, in login
    username=user_details_list[0]
TypeError: 'NoneType' object is not subscriptable
```

The way to fix this is to have a try and except in python as we can try reading the data first and seeing if it exists therefore, if the username does not exist, an error message can be displayed. This is the easiest way as we can also later add an else if we want any more processing e.g. the admin login.

```
try:  #Try reading the username from the database
    cur.execute("""SELECT Username, Password FROM Customers
                WHERE Username=?""",(self.username_string,))
    user_details_list=cur.fetchone()
    username=user_details_list[0]
    password=user_details_list[1]
    if username==self.username_string: #An if statement comparing username i
        if password==self.password_string:
            print("Login successful")
        else: #If the password is incorrect login failed
            print("Login failed - password incorrect")
except: #If the username cannot be found in database
    print("Login failed - username not found")
```

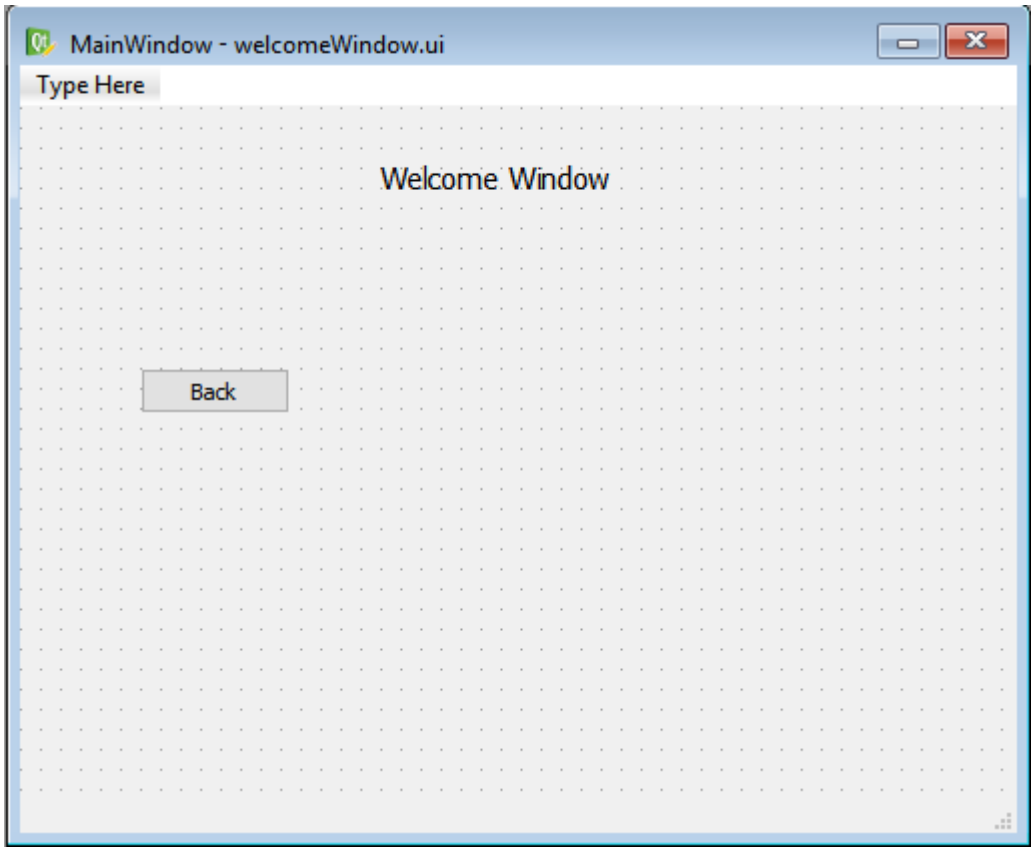
Using the same input:



Gives this output:

```
---
Login failed - username not found
```

The next step was to make use of this output. If the login is successful we want to hide the current window and display our welcome page with features, and if login is unsuccessful we want to stay in the current window. Firstly, I made the welcome page with a back button as we will be focusing on the rest of the buttons and features later



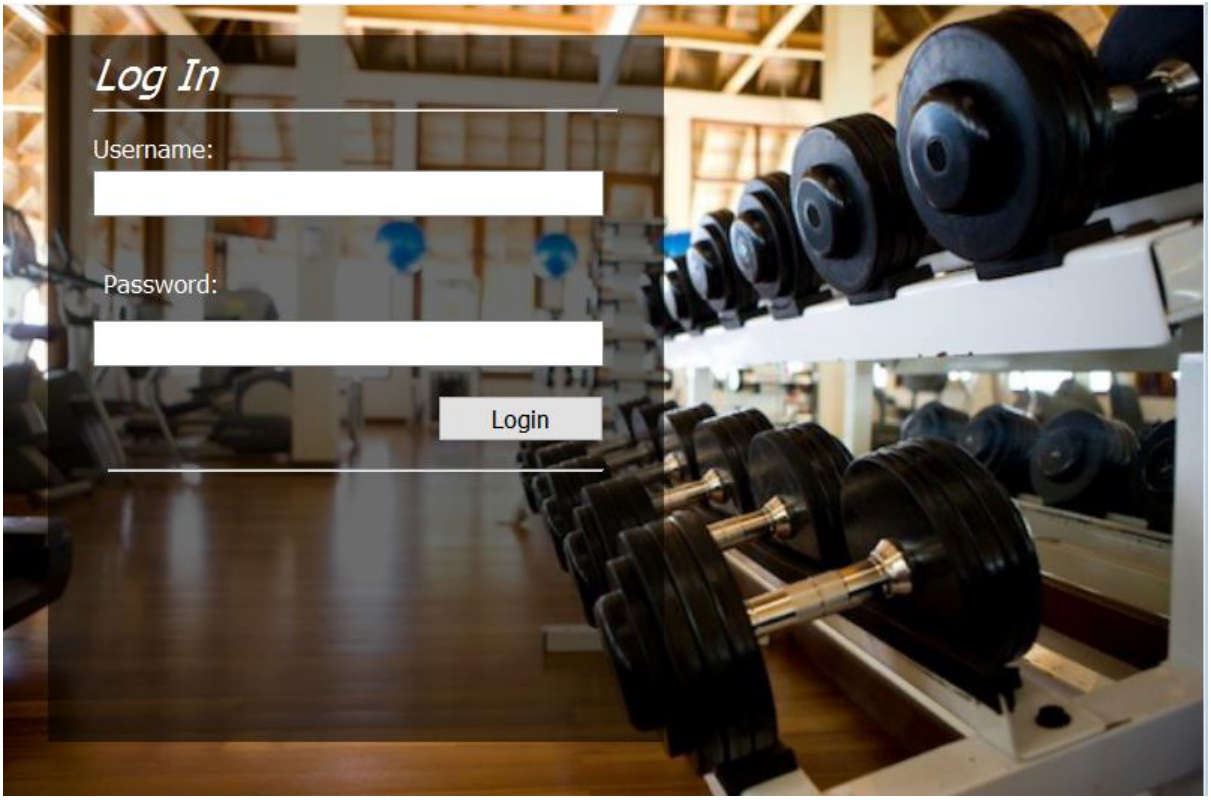

```
class SecondWindow(QtGui.QMainWindow, welcome_win):
    def __init__(self, parent=None): #Sets up the UI
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
```

I added this to my code:

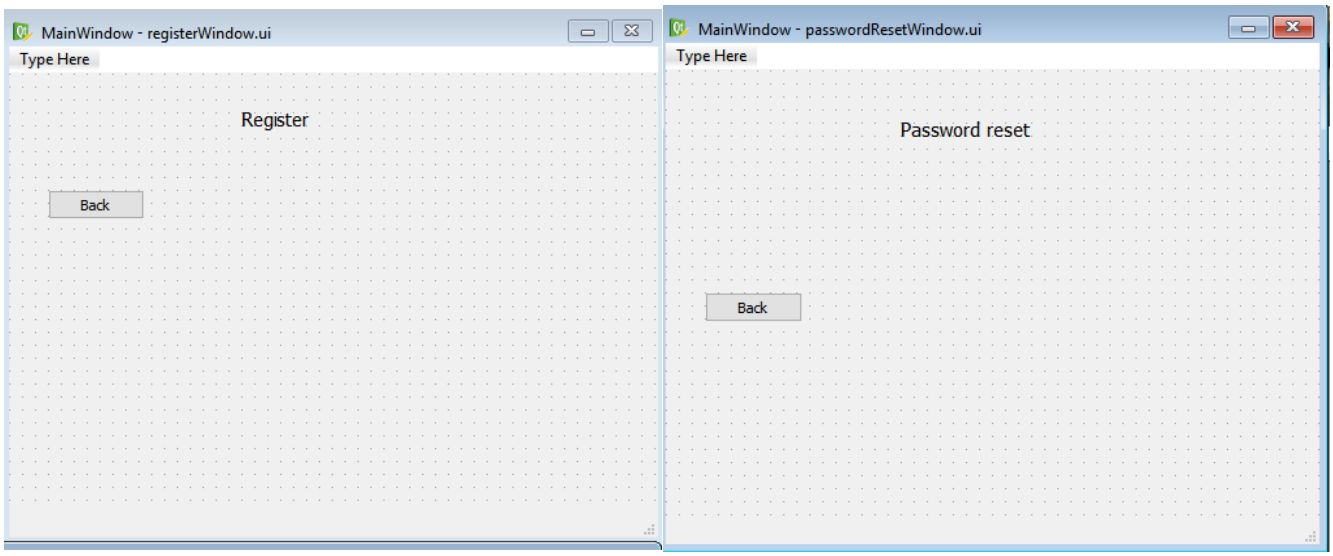
```
QtGui.QMessageBox.information(self, "Login", "Login successful")
self.hide()
welcome_window.show()
self.username_string.clear()
self.password_string.clear()
```

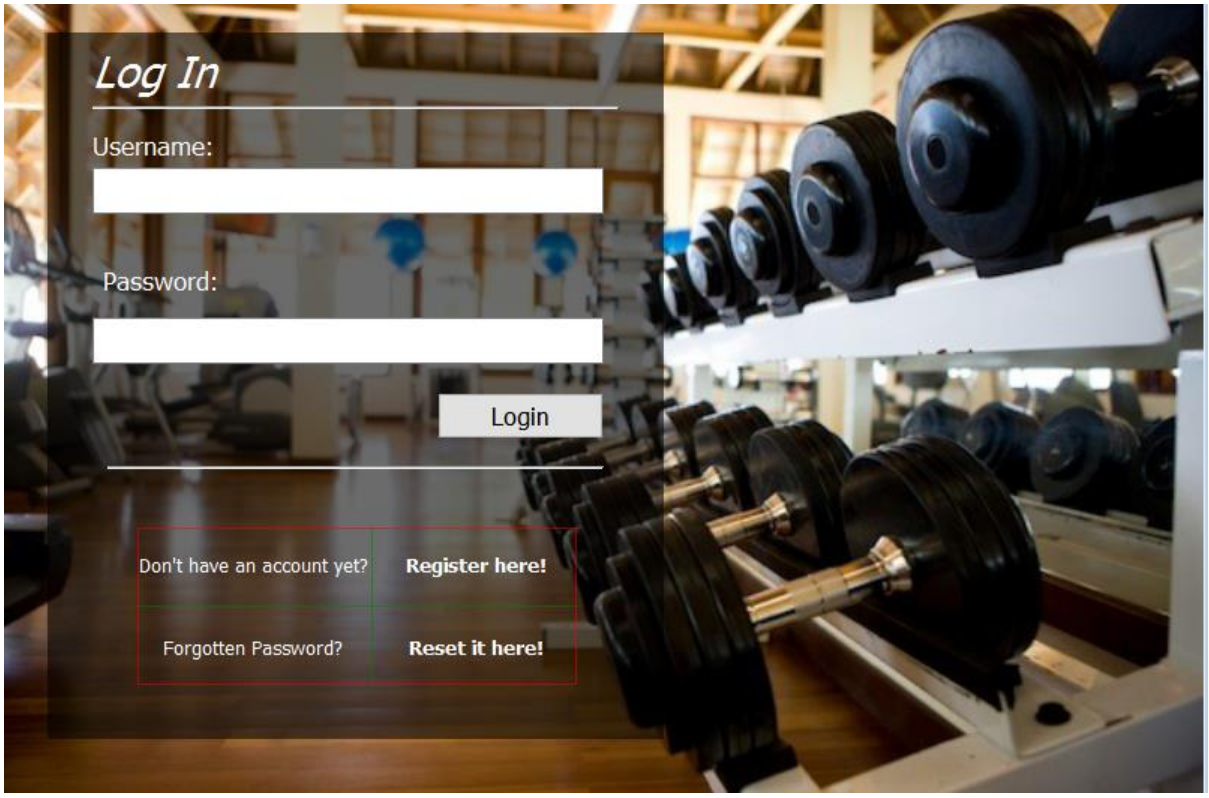
Now instead of the message being printed to the console, a message box will appear instead. The login window is then hidden; the line edits are cleared and the welcome window is shown. Once the button is pressed the current window is hidden and the welcome window is shown. The next step was to make the login window more visually appealing. I added a picture of weights into the window, a black box with 40% transparency on top of this picture and changed the font to a bigger size and white colour:

I also set the echoMode of the password line edit to password, so now only black dots appear when typing in the password. I then added labels – when clicked they will take us to the reset email screen or register screen



For this I had to create the register and the password reset windows. They include back buttons and will be developed later.





Setting them up in python:

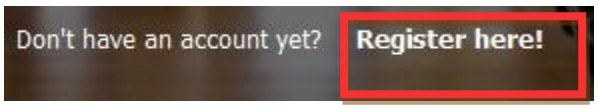
```
class ThirdWindow(QtGui.QMainWindow, register_win):
    def __init__(self, parent=None): #Sets up the UI
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)

class FourthWindow(QtGui.QMainWindow, passwordreset_win):
    def __init__(self, parent=None): #Sets up the UI
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
```

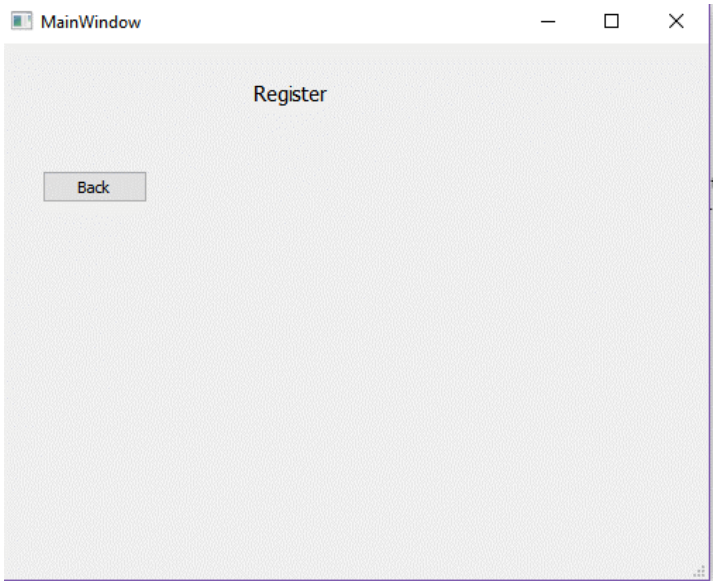
Now we must link it to a function. When the area of the label is pressed, it will execute a function

```
self.register_lbl.mousePressEvent = self.open_second
self.reset_lbl.mousePressEvent = self.open_reset
```

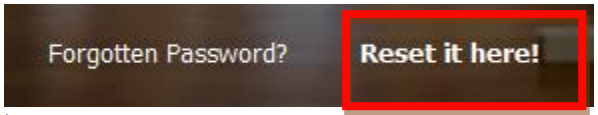
```
def open_second(self, event): #Opens the register window
    self.username_input.clear()
    self.password_input.clear()
    self.hide()
    register_window.show()

def open_reset(self, event): #Opens the password reset window
    self.username_input.clear()
    self.password_input.clear()
    self.hide()
    passwordreset_window.show()
```

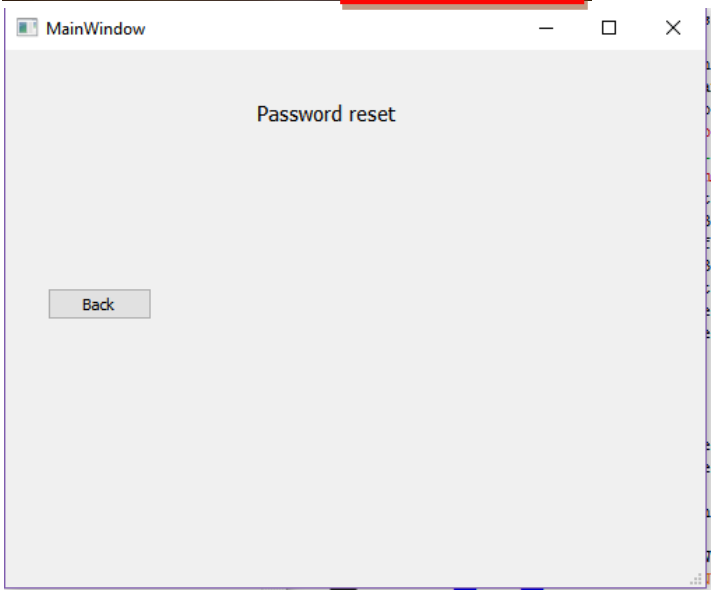
When the register label is clicked, the current window is hidden, the line edits are cleared and the register window is shown. When the reset label is pressed, the current window is hidden, the reset window is shown and the line edits are cleared. The red box represents the area of the label therefore the area of the input for that label.



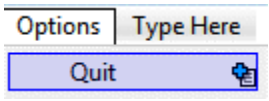


Then when the reset label is pressed:





Next I decided to add some extra functionality to the admin window. Firstly, a quit button. In my window in PyQt Designer I added an action button in the toolbar for a 'quit' action:

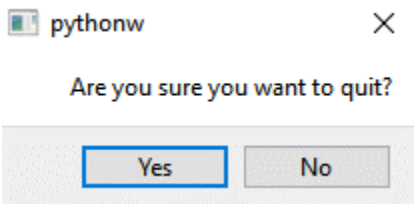


Then I added to my code the code for this action:

```
self.actionQuit.triggered.connect(self.quit) #Can quit using shortcut
self.actionQuit.setShortcut(QtGui.QKeySequence("ESC")) #Use ESC to quit
```

And the function that links the button to this action:

```
def quit(self): # When the ESC button is pressed this is the function that is run
    msgBox = QtGui.QMessageBox()
    msgBox.setText('Are you sure you want to quit?')
    msgBox.addButton(QtGui.QPushButton('Yes'), QtGui.QMessageBox.YesRole)
    msgBox.addButton(QtGui.QPushButton('No'), QtGui.QMessageBox.NoRole)
    ret = msgBox.exec_()
    if ret==0: # If the button with 'yes' is clicked, close current window
        self.close()
    else: # If the button with 'no' is clicked, close message box
        msgBox.hide()
```



When I pressed the 'esc' button in the login window, this message box was displayed:

If yes is pressed, the program is closed, and if no is pressed the message box hides and the login window is still open

Next, I decided to do a button to confirm login. I set the return button to confirm this action

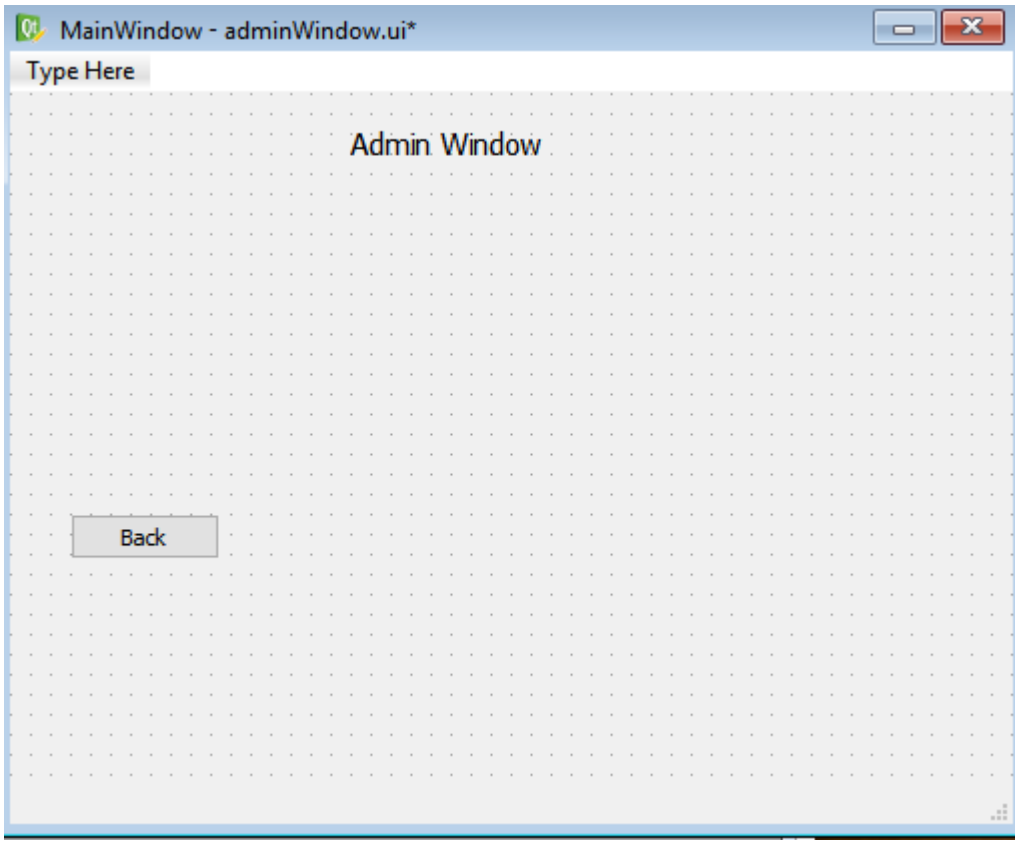

```
self.actionEnter.triggered.connect(self.login) #Can quit using shortcut
self.actionEnter.setShortcut(QtGui.QKeySequence("ENTER")) #Use Enter to login
```

I set it as the enter button. However, this was not working as it is the return button not the enter button.

```
self.actionEnter.triggered.connect(self.login) #Can quit using shortcut
self.actionEnter.setShortcut(QtGui.QKeySequence("RETURN")) #Use Enter to login
```

When the 'enter' button was pressed, depending on username and password, the correct message box was displayed.

The next part was to make sure an admin account would be able to log in. Firstly, I created an admin window. This will be developed later.



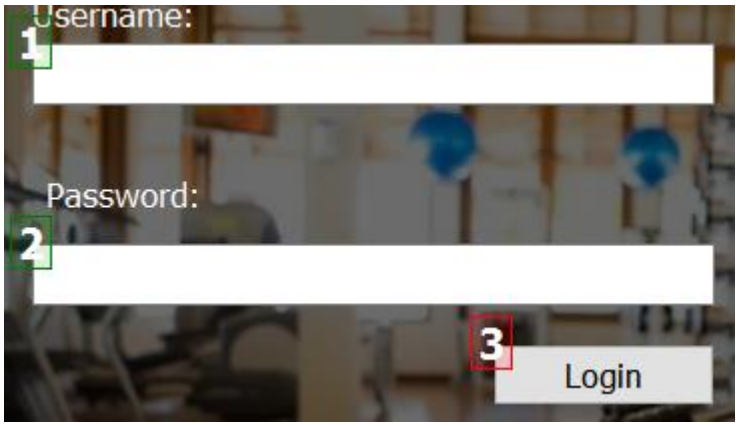
Then I added the code to setup the admin window in python

```
admin_win = uic.loadUiType("adminWindow.ui")[0]
class FifthWindow(QtGui.QMainWindow, admin_win):
    def __init__(self, parent=None): #Sets up the UI
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
admin_window = FifthWindow(None)
```

Almost the same code for checking the username and password inputs against the ones from the database but this time checking against the admin username and admin password.

```
 if self.username_string==admin_username: # If the input matches the admin password
     if self.password_string==admin_password:
         QtGui.QMessageBox.information(self, "Login", "Admin login successful")
         self.hide()
         admin_window.show()
         self.username_input.clear()
         self.password_input.clear()
     else:
         QtGui.QMessageBox.critical(self, "Error", "Username or password incorrect")
```

The next part in this window was to set the tabs in order in PyQt Designer so the user can tab through the line edits easily.



The window starts at the username, then when 'tab' is pressed, goes to the password line and finally the login button. The last step in the login window is to make sure we have the password input hashed and checked against the password in the database. First I imported a module name hashlib

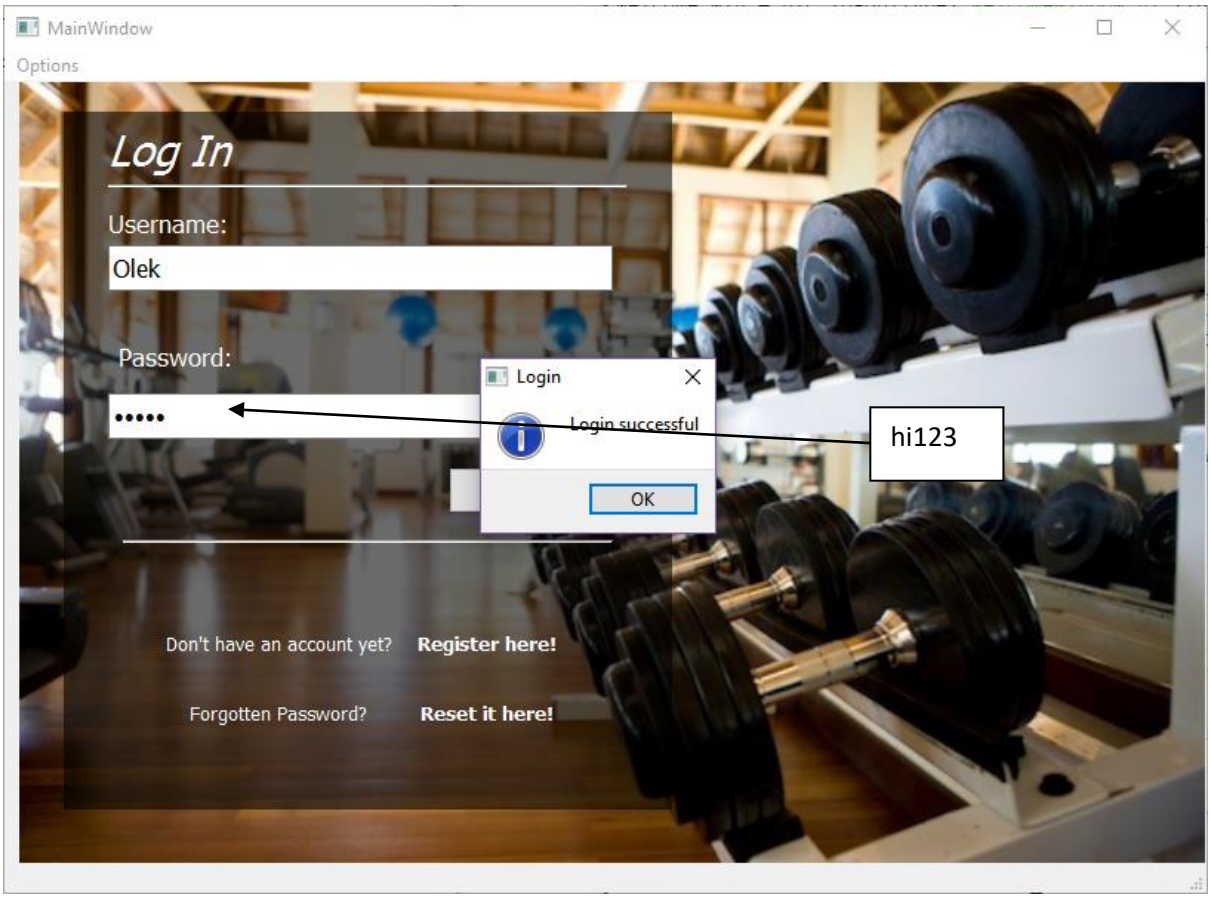
```
import hashlib
```

Then I set the password input and hashed this value. Then I compared this variable to the password in the database.

However, when I pressed the login button it did not log me in. This was because the password in the database was not hashed. I hashed the value 'hi123' and stored it in the database

```
import hashlib
password_original="hi123"
new_password=hash
print(new_passwor

30c2138619045a1dd
```

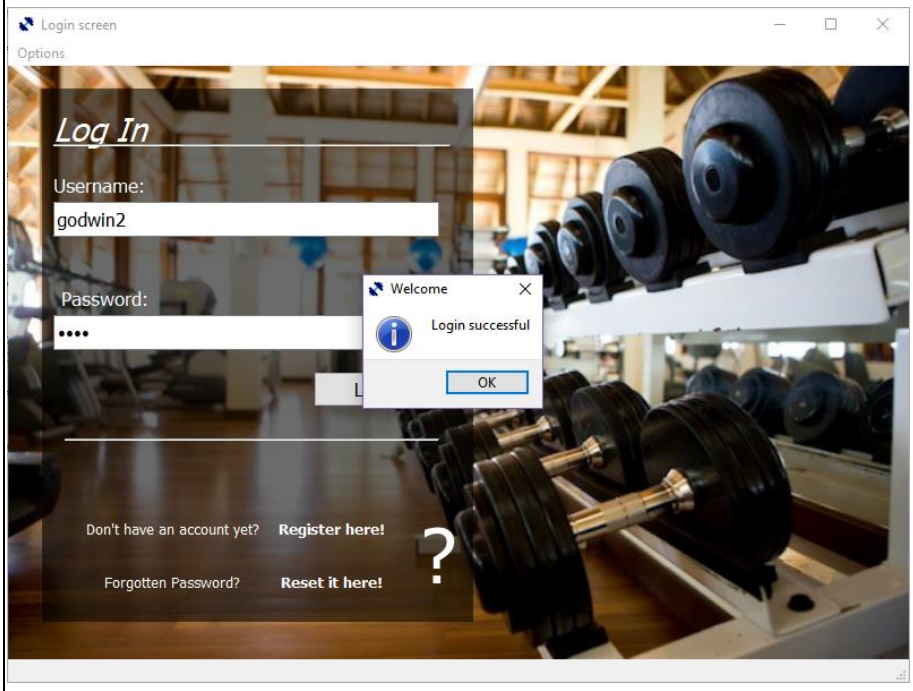
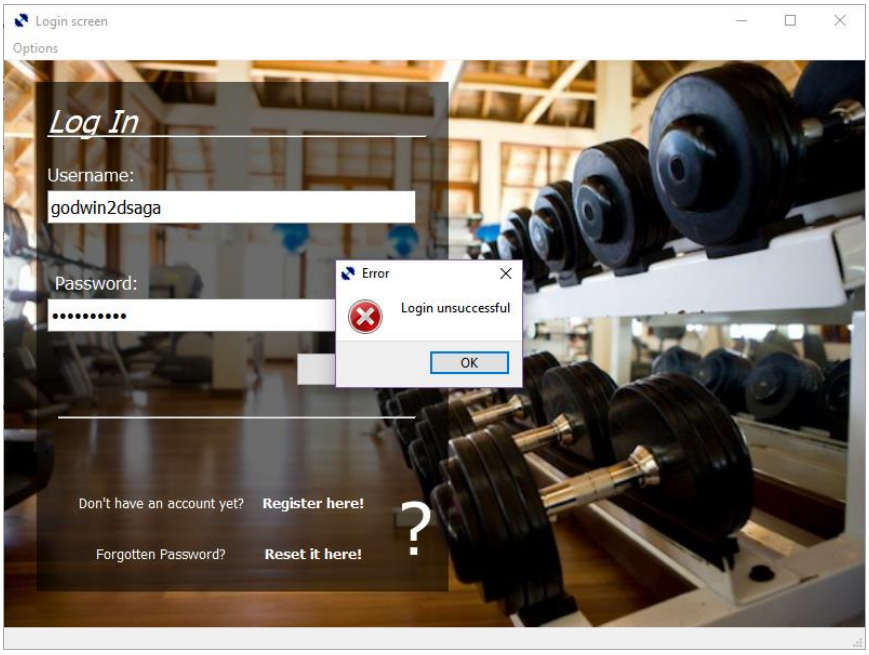
	CustomerID	Username	Password	Name
	Filter	Filter	Filter	Filter
1	1	Olek	30c2138619045a1dd4b1f6cb888f0d67	NULL



The password is still hi123 for the user and this is still what they type in, but this time this gets hashed and checked against the password in the database (not the hash of this) as it is already stored as a hash in the database.

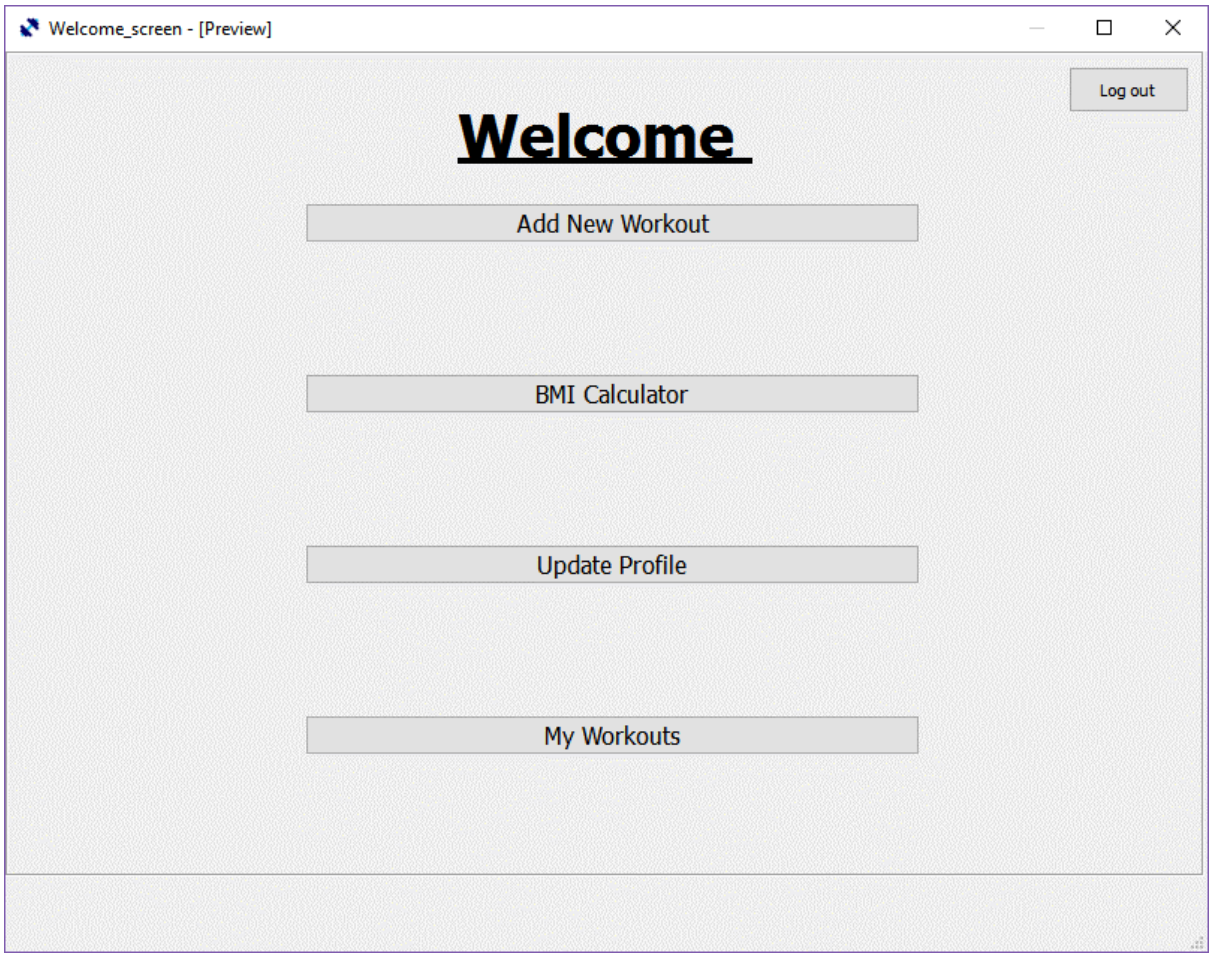
Testing the login window

What am I testing?	How will it be tested?	Success/Fail (action taken if fail)
Can the user login using username and password	Running the program and seeing if the pre-set variables match with the input	Success - Enter the usernameOlek Enter the passwordhi123 Login successful >>>
Can the user login using incorrect details	Running the program and inputting the wrong username and password	Success - Enter the usernameOlek13412 Enter the password1234124 Login failed >>>
Can the user login using incorrect details from the database	Running the program and logging in and seeing if details match the ones in database	Fail – I used try, except and else to correct this error. By doing this the program was first checking if it can read any details from the database before assigning the username from the list. This was a better solution than coming up with lots of elif statements as I would have had to fill each criteria – so this was is also more efficient Traceback (most recent call last): File "C:\Users\oleks\OneDrive\Documents\Coursework tables pyQt\Program.py ne 20, in login username=user_details_list[0] TypeError: 'NoneType' object is not subscriptable
Can the user log in using someone else's password	Have two accounts in db – use username for one and password for another account	Fail – when I first did this I was reading all the users from the database and then selecting the user where the username was found in the list. I could be reading different details e.g. if a username happens to be someone else's password. This would then read the wrong data. Therefore, I executed an SQL statement which selects the username and password from the database where the username is equal to the input. If it cannot find it an error message is displayed.

Can the user login inside the GUI window	Run the program and type in the details into the window	Success – 
Can the user login with random details inside the GUI window	Run the program and type in random input into password and username	Success – 

Welcome Window

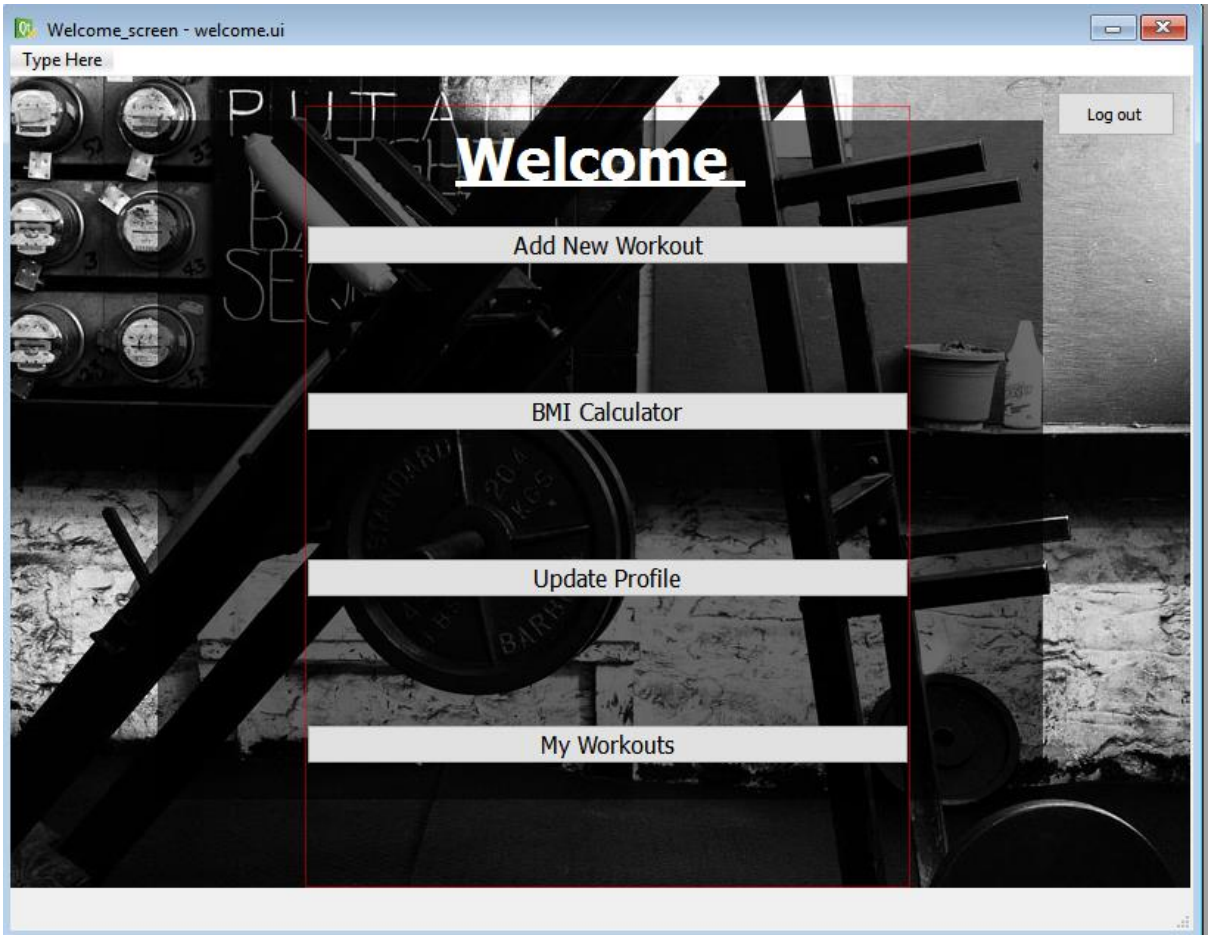
The next step was to develop the welcome window. We started this when we made the login window. I added the buttons which would link to each of the other window.



I then linked all the buttons to the windows and added this to the logout function I created earlier. This was all that had to be done for the welcome window

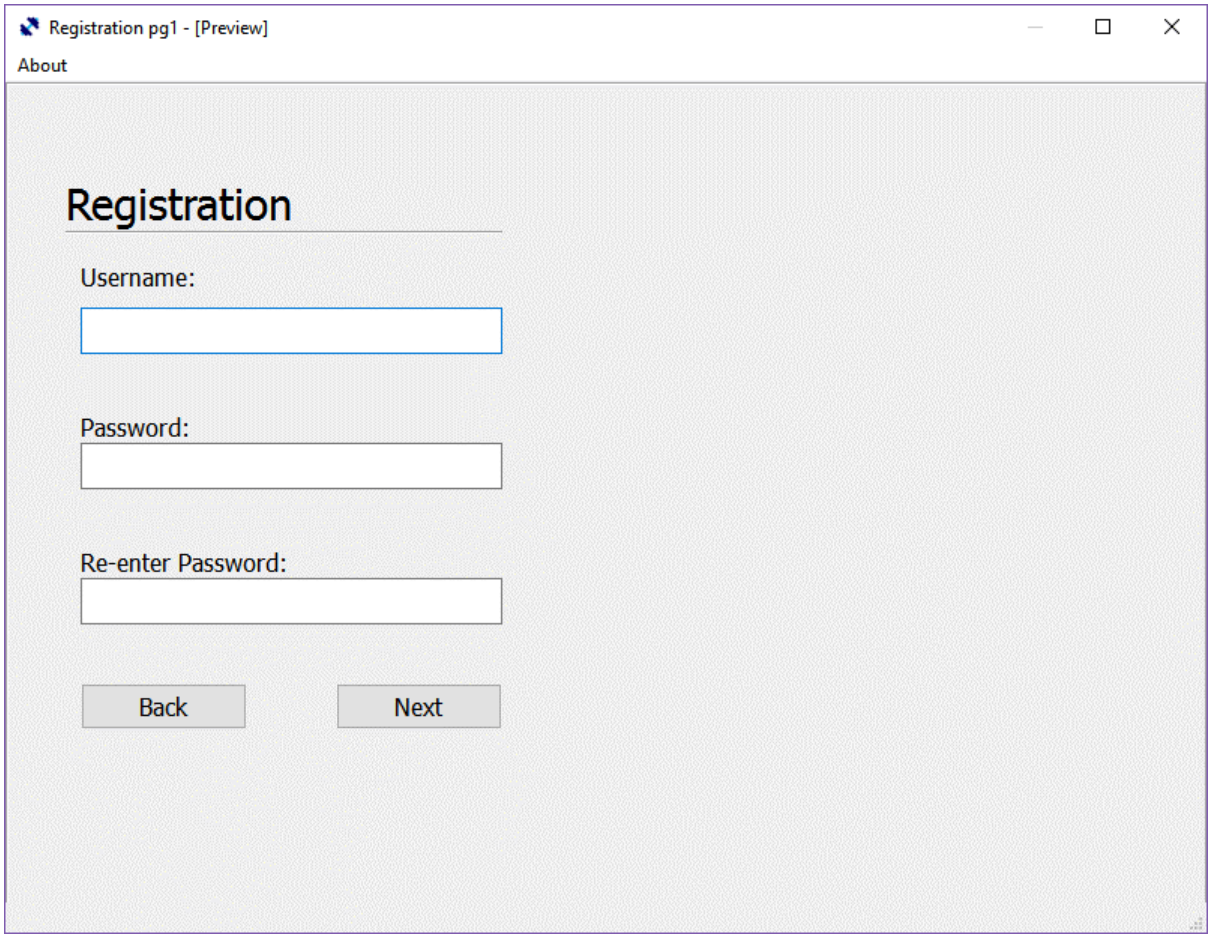
```
class ThirdWindow(QtGui.QMainWindow, welcome_win):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self,parent)
        self.setupUi(self)
        self.myworkoutsBtn.clicked.connect(self.gomyworkouts)
        self.workout_butt.clicked.connect(self.gonext)
        self.calorieBtn.clicked.connect(self.goBMI)
        self.updateProfileBtn.clicked.connect(self.goProfile)
        self.logoutBtn.clicked.connect(self.logout)
    def logout(self):
        myWindow.lineEdit_2.clear()
        myWindow.lineEdit.clear()
        msgBox = QtGui.QMessageBox()
        msgBox.setText('Are you sure you want to log out?')
        msgBox.addButton(QtGui.QPushButton('Yes'), QtGui.QMessageBox.YesRole)
        msgBox.addButton(QtGui.QPushButton('No'), QtGui.QMessageBox.NoRole)
        ret = msgBox.exec_()
        if ret==0:
            self.close()
            myWindow.show()
        else:
            msgBox.hide()
```

I added a picture of weights in the background and a black box with 40% transparency with white font for a professional look



Register Window

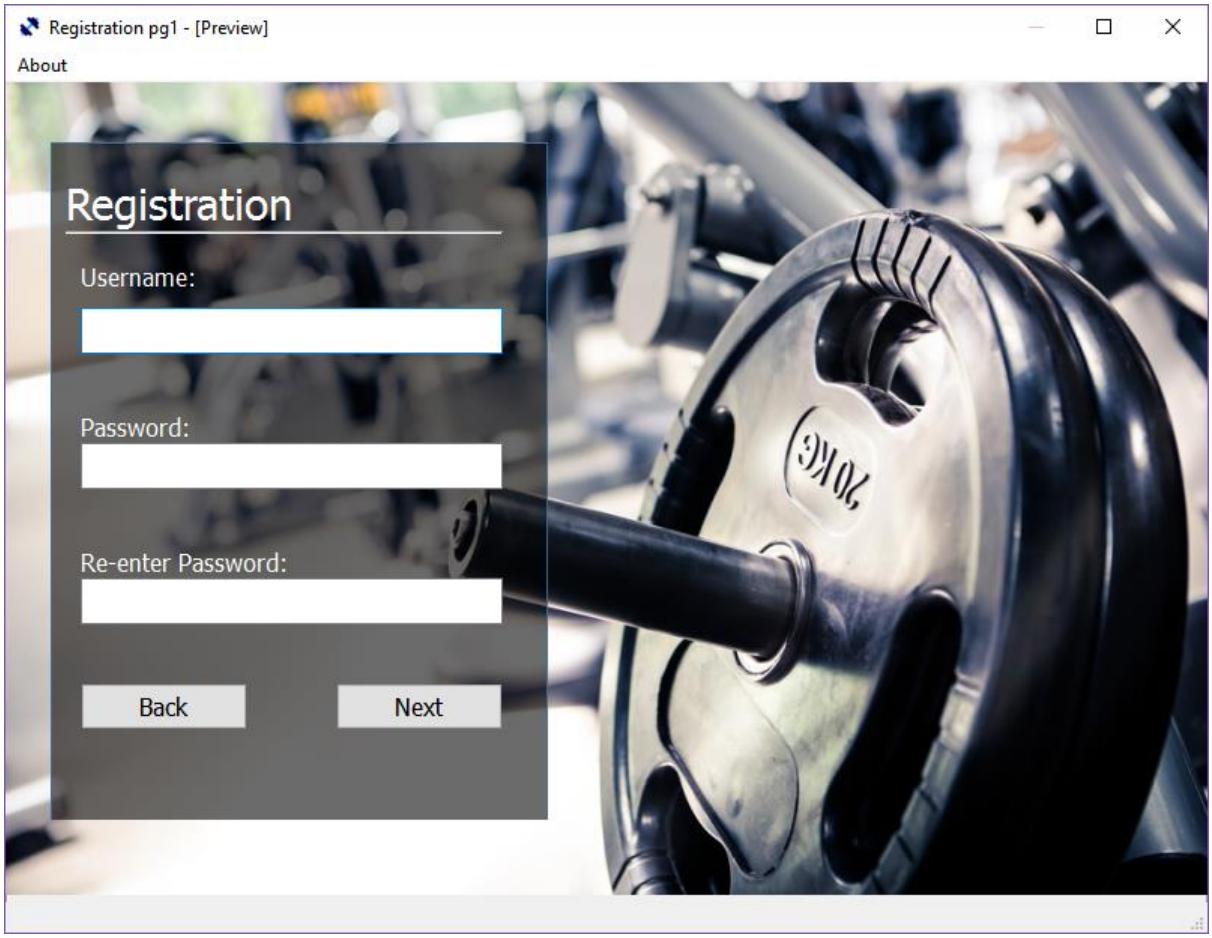
Firstly, I developed the register screen to have basic line edits



Firstly, I linked the buttons and set the line edits to 20 characters max.

```
class SecondWindow(QtGui.QMainWindow, register_pg1_win):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.lineEdit.setMaxLength(20)
        self.lineEdit_2.setMaxLength(20)
        self.lineEdit_3.setMaxLength(20)
        self.actionEnter.triggered.connect(self.next_page)
        self.actionEnter.setShortcut(QtGui.QKeySequence("RETURN"))
        self.home_butt.clicked.connect(self.goback)
        self.next_butt.clicked.connect(self.next_page)# Bind the event handlers
        #   to the buttons
```

I then added a picture of gym weights and black box with 40% transparency to make the register window stand out visually

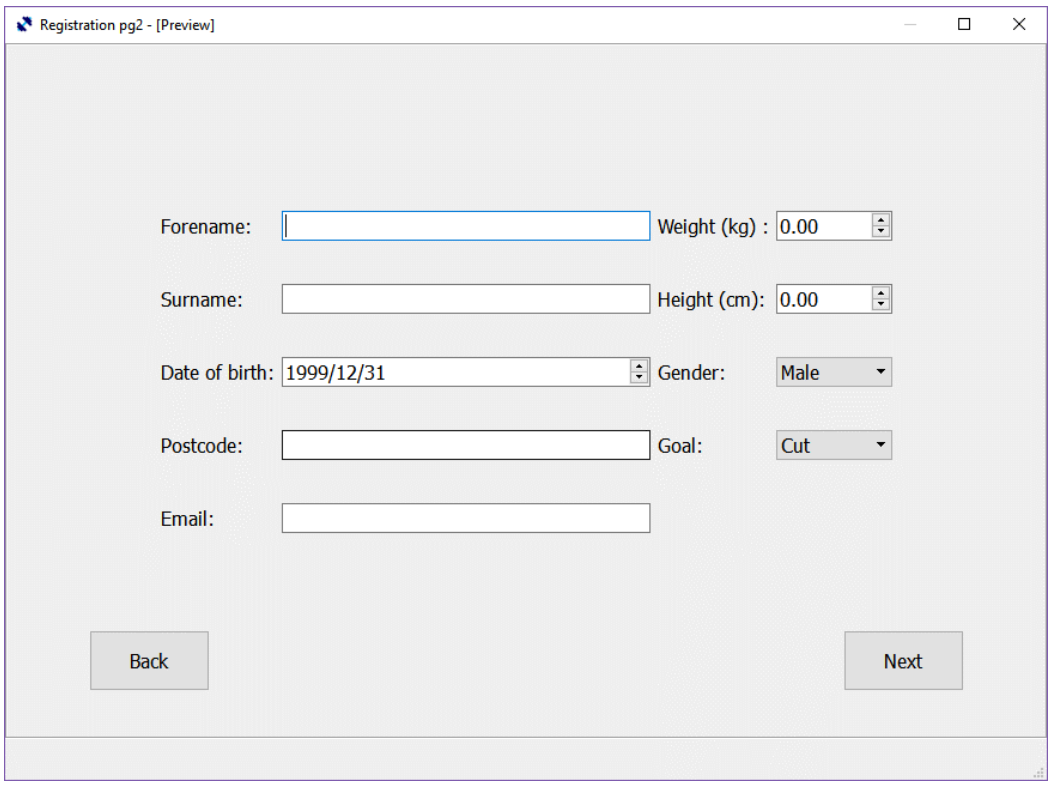


Then I added the validation for the password boxes and the username boxes:

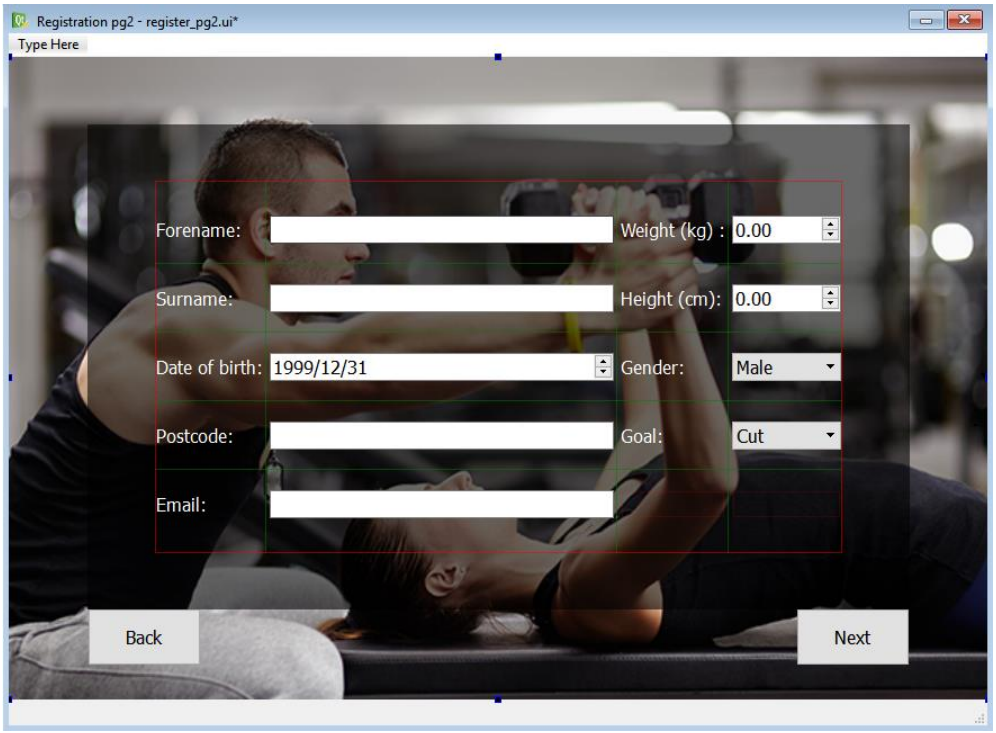
In this section, I'm testing for empty password, if the length is more than 4 characters, if the first password matches the re-enter password, if username is empty, if username is equal to password since there cannot be the same username as there is password and checking if there are spaces in the passwords. Also, I am checking to see if the username already exists in the database to make sure usernames are not repeated.

Register window 2

Firstly, I created a basic window for this screen:



Then I added a picture to the background and black box with 40% transparency:



I then set up this window in Python:

```
class FourthWindow(QtGui.QMainWindow, register_pg2_win):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.back_but.clicked.connect(self.goback)
        self.next_but.clicked.connect(self.gonext)
    def goback(self):
        self.hide()
        wind2.show()
```

I then set variables to all the inputs in the line edit

```
def gonext(self):
    self.forename=self.forename_edit.text()
    self.surname=self.surname_edit.text()
    self.email=self.email_edit.text()
    wind2.password=wind2.password=hashlib.md5(wind2.password.encode('utf-8')).hexdigest()
    self.postcode=self.postcode_edit.text()
    self.weight=self.weightEdit.text()
    self.height=self.heightEdit.text()
    self.gender=self.genderCB.currentText()
```

I also used wind2.password to inherit the previous windows password input so that it can be stored.
I then validated the inputs, some using regular expressions:

If the user's inputs were fine, then the values they input were written to the database file:

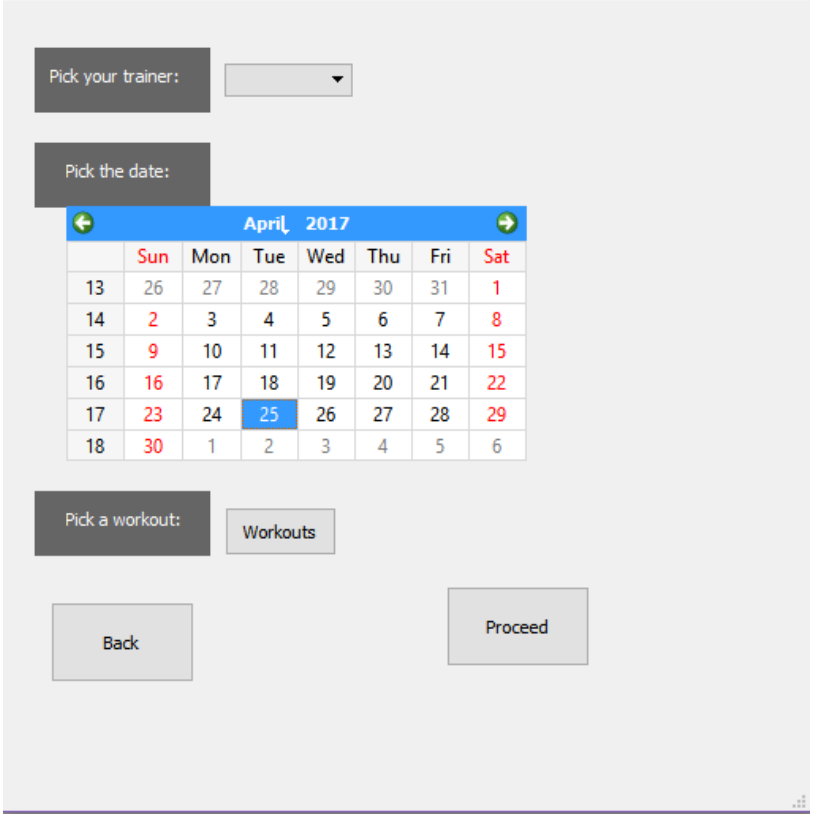
```
else:
    cur.execute("INSERT INTO Customers (Username, Password, Name, Surname, Postcode,
                (wind2.username, wind2.password, self.forename, self.surname, self.p
    conn.commit()
    self.close()
    wind3.show()
```

```
  Email, Weight, Height, Gender, GoalID) VALUES ( ?, ?, ?, ?, ?, ?, ?, ?, ?, ? )",
 ostcode, self.email, self.weight, self.height, self.gender, self.goal))
```

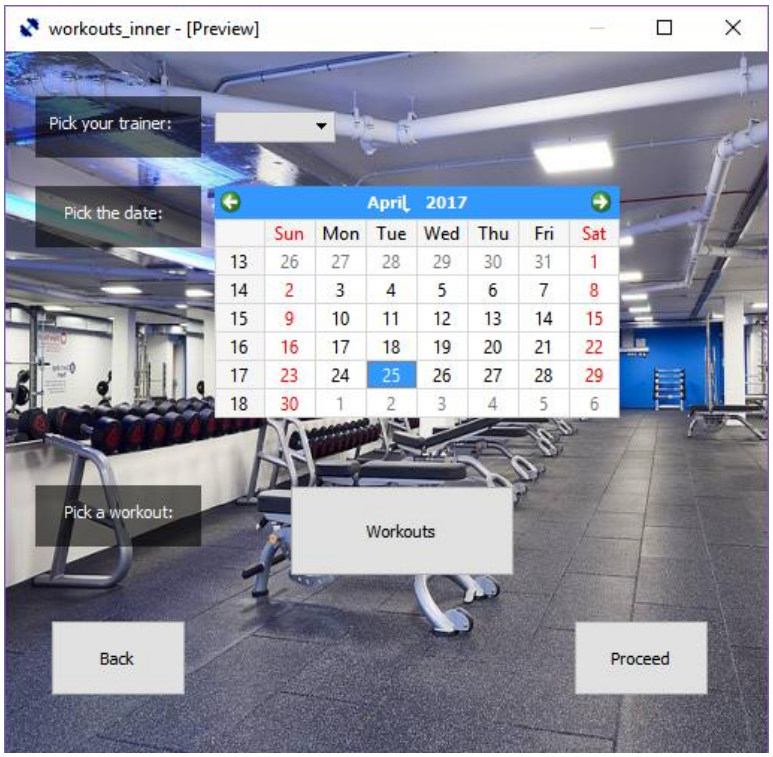
```
        if self.goalCB.currentText()=="Cut":
            self.goal=1
        elif self.goalCB.currentText()=="Build":
            self.goal=2
        elif self.goalCB.currentText()=="Transform":
            self.goal=3
        if not re.match("^[a-z]*$", self.forename.lower()):
            QtGui.QMessageBox.critical(self, "Error", "Only letters a-z in forename allowed")
        elif self.forename=="":
            QtGui.QMessageBox.critical(self, "Error", "Forename empty")
        elif not re.match("^[a-z]*$", self.surname.lower()):
            QtGui.QMessageBox.critical(self, "Error", "Only letters a-z in surname allowed")
        elif self.surname=="":
            QtGui.QMessageBox.critical(self, "Error", "Surname empty")
        elif not re.match("^[0-9]*$", self.mobile):
            QtGui.QMessageBox.critical(self, "Error", "Only numbers allowed")

        elif self.email=="":
            wind4.show()
```

Workouts window

The next window I decided to develop was the window to book and choose the workouts. This is the main part of my overall program. Firstly, I created a simple version of the window



I then added a picture to the background and added an icon to the window:



Next, I linked the windows in Python

```
workouts_inner = uic.loadUiType("workout_inner.ui")[0]
class FifthWindow(QtGui.QMainWindow, workouts_inner):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
```

I read all the details of the trainers from the database next.

	TrainerID	TrainerName	PaymentRates
	Filter	Filter	Filter
1	1001	Josh Taylor	7.2
2	1002	Hugh Mungus	6.9
3	1003	George Smith	6.55
4	1004	Sam Daniels	7.1
5	1005	Johnny Rogers	8.2
6	1006	Tim Michael	7
7	1007	James East	7.5
8	1008	Patrick Sun	7.3
9	1009	Jim Falcon	8.1

```
cur.execute("select * from Trainers")
trainers_data=cur.fetchall()
for each in trainers_data: #For each piece of data in the list
    list1.append(each[1])
```

After I read the details from the trainer's table, I added all the names of trainers (each [1]) to a list called list1. Then I added this list to the combo box after clearing it first to prevent multiple additions of the same list into one combo box

```
self.comboBox.clear()
self.comboBox.addItems(list1)
```

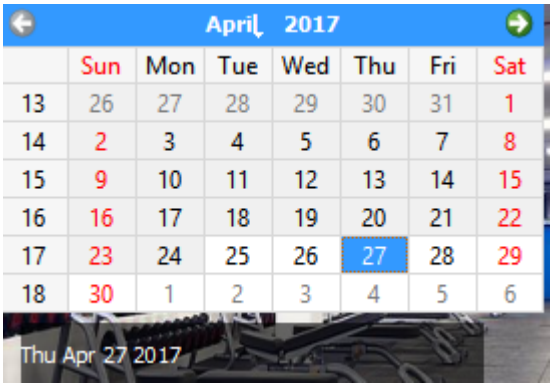
Next, I set a minimum date for my calendar to today's date, and the date that is selected is a global variable:

```
self.calendar.clicked[QtCore.QDate].connect(self.showDate)
global date
date = self.calendar.selectedDate()
self.date_lbl.setText(date.toString())
current_time=time.strftime("%d/%m/%Y")
self.calendar.setMinimumDate(QDate(date))
```

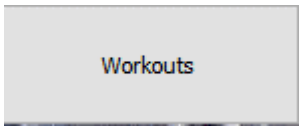
When the calendar is clicked, go to the showDate function. Global date is set as global variable so it can be used in the next window. The date is the date that is selected on the calendar by the user. Then a label is set to the date selected to show the user. toString makes it user friendly and displays the date as e.g. "Tuesday 25th November" instead of 25/11/2017. I also set the colour of the label to white on a black background as I could not do this in Designer. Therefore, I decided to code it myself.

```
palette = QtGui.QPalette()
palette.setColor(QtGui.QPalette.Foreground,QtCore.Qt.white)
self.date_lbl.setPalette(palette) #Sets the colour
```

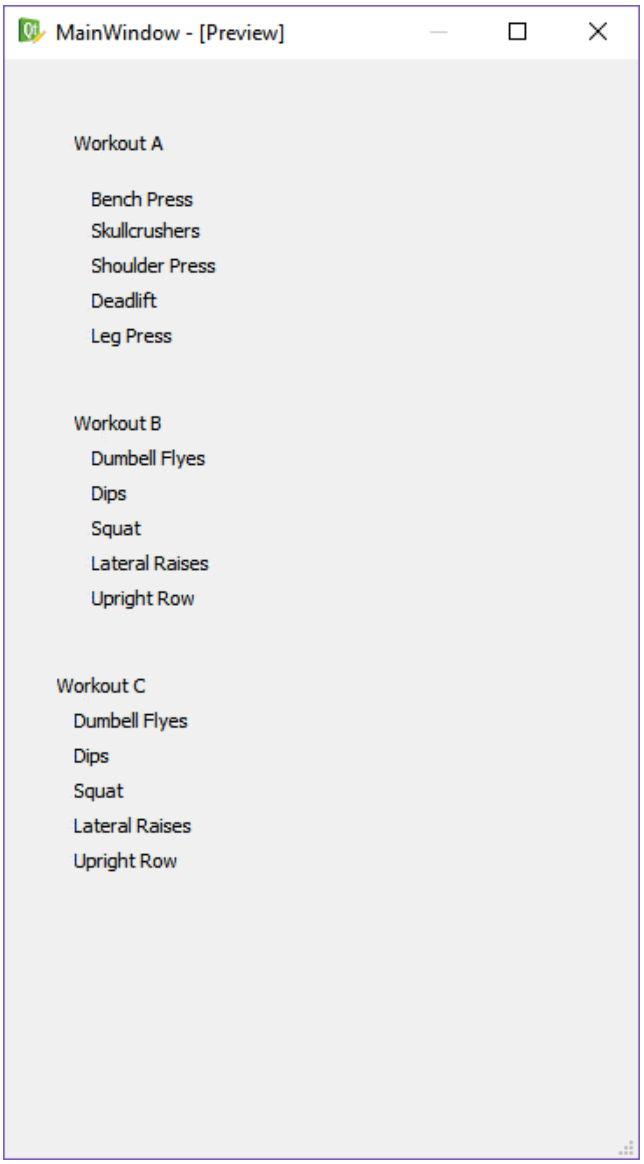
QPalette lets you pick from a range of colours. Then I set palette to have a white colour. Then I set the label to have the colour of the palette variable (white):



Next, I added a workouts button so the user can pick their routine:

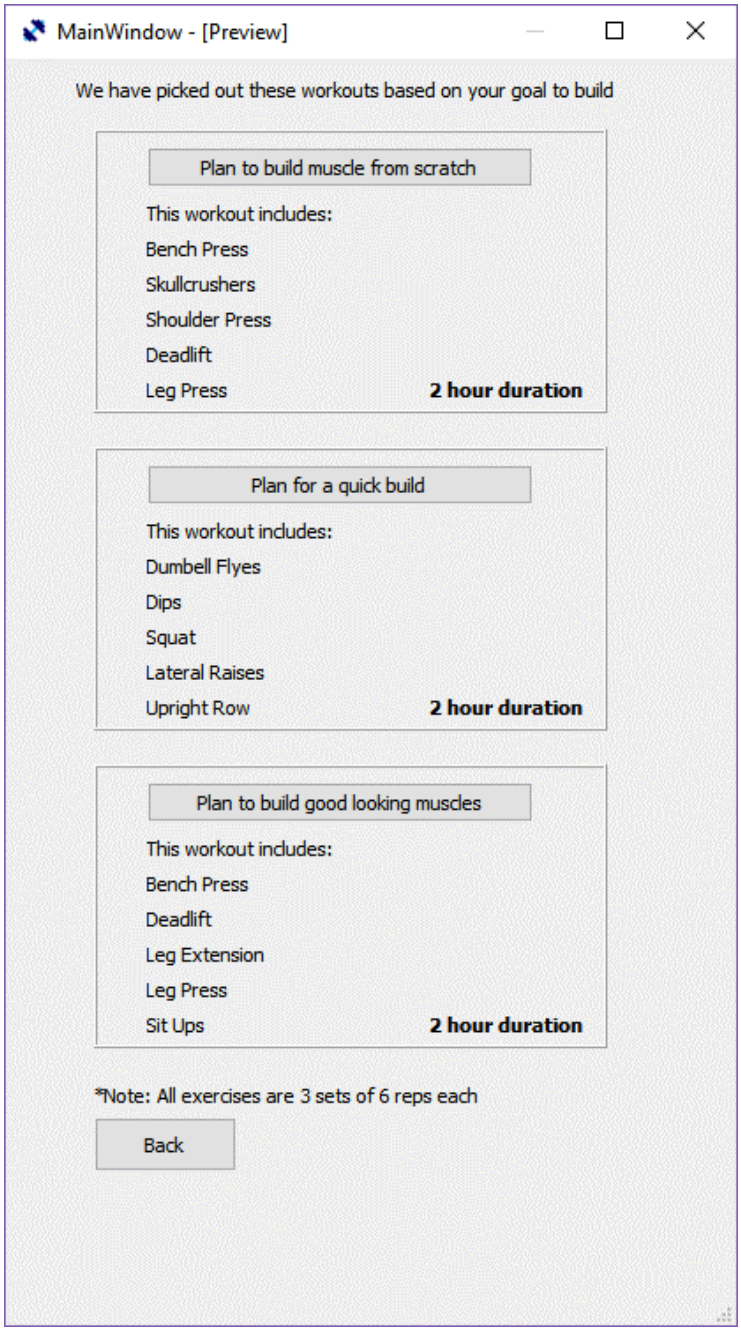


I then made routines for all the goals. Firstly, I made the one for the 'build goal':



This was a basic layout for the workouts, so I added sunken frames and a button for each workout along with a back button:

I added the duration of the workout also. When the button was pressed the exercise ID for that plan was read from the database



I connected this window to python and when each of the buttons was clicked, they would be taken to that function

```
class EleventhWindow(QtGui.QMainWindow, build_win):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        self.plan1Btn.clicked.connect(self.plan1)
        self.plan2Btn.clicked.connect(self.plan2)
        self.plan3Btn.clicked.connect(self.plan3)
```

```
def plan1(self):
    global planID
    planID=1004
    global duration
    duration=str(2)
    cur.execute('SELECT * FROM Exercises WHERE ExerciseID="1" OR ExerciseID="7" OR ExerciseID="15" OR ExerciseID="8" OR ExerciseID="10"')
    self.fetched=cur.fetchall()
    print(self.fetched)
    self.hide()
    wind5.show()
```

The exercise description was read from the database as well as the ID. Duration is the length of the workout (2 hours). This will be used later to calculate the total payment. PlanID is the plan ID unique to this routine. I did the same for the other 2 'build workouts'.

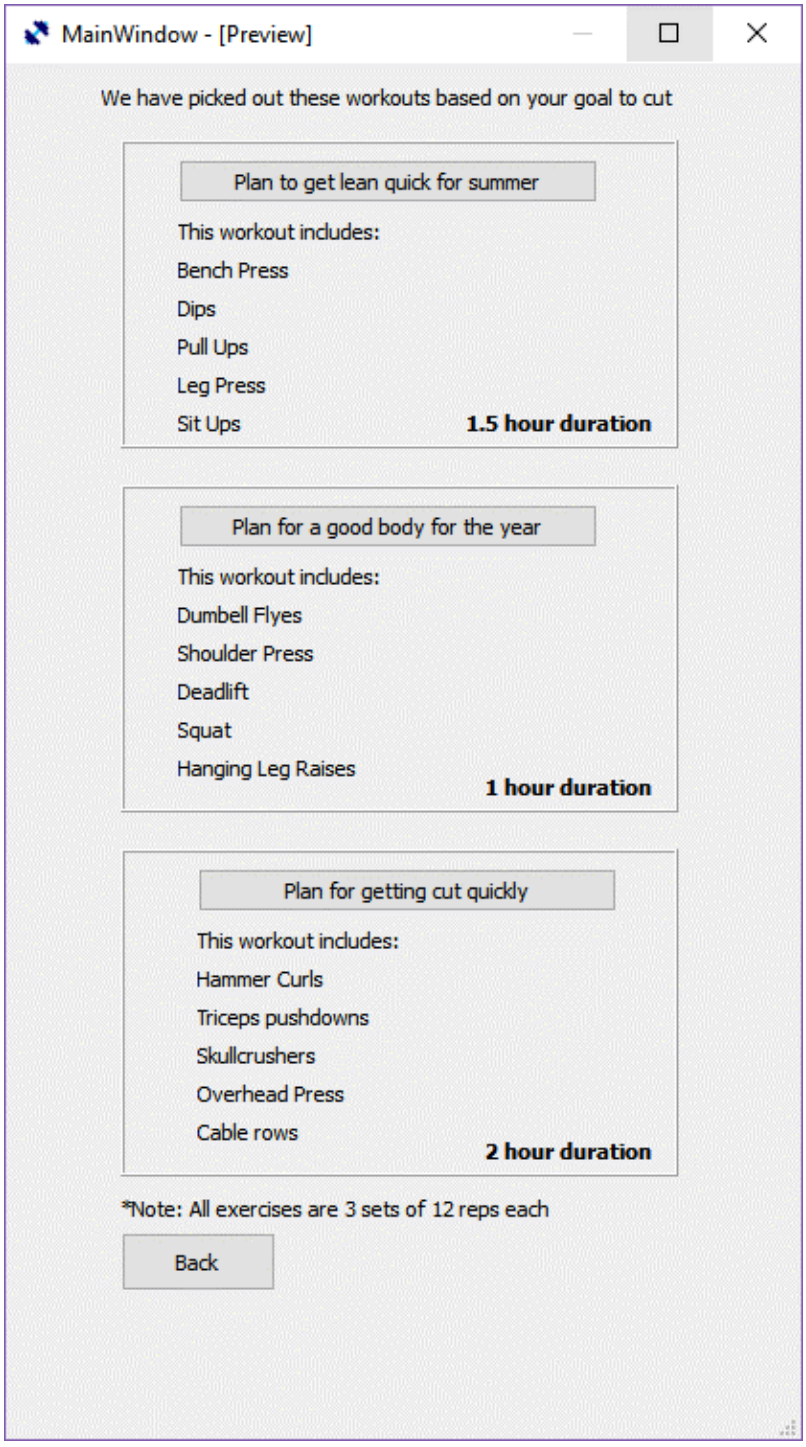
```
def plan2(self):
    global planID
    planID=1005
    global duration
    duration=str(2)
    cur.execute('SELECT * FROM Exercises WHERE ExerciseID="2" OR ExerciseID="3" OR ExerciseID="12" OR ExerciseID="16" OR ExerciseID="20"')
    self.fetched=cur.fetchall()
    print(self.fetched)
    self.hide()
    wind5.show()
def plan3(self):
    global planID
    planID=1006
    global duration
    duration=str(2)
    cur.execute('SELECT * FROM Exercises WHERE ExerciseID="1" OR ExerciseID="13" OR ExerciseID="10" OR ExerciseID="18" OR ExerciseID="15"')
    self.fetched=cur.fetchall()
    print(self.fetched)
    self.hide()
    wind5.show()
```

All I changed for the other plans was the planID and the exerciseIDs read from the database. PlanID is global as it needs to change depending on which workout is picked and therefore it must exist throughout the program. Duration is also global as it must change outside the function but needs to be named inside the function as the workouts can have different durations. Also, the function for going back to the previous window:

```
def goback(self):
    self.hide()
    wind5.show()
```

I then made the 'cut' goal routines window:

It has the same styling/design as the ‘build’ routines, but this time the exercises are changed, the duration is different and all the routines are 3 sets of 12 reps each, compared to 3 sets of 6 reps if the goal is to ‘build’.

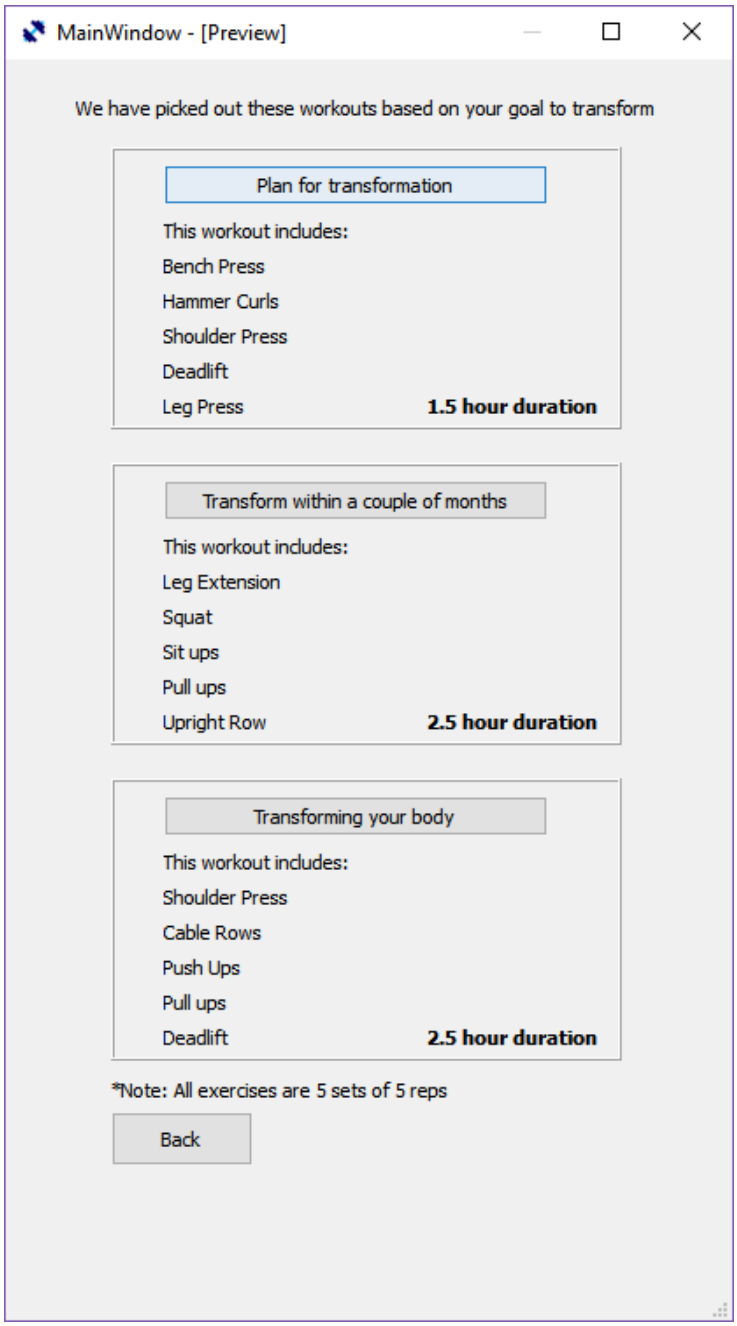


This was the code for the ‘cut’ routines:

```
class TenthWindow(QtGui.QMainWindow, cut_win):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        self.plan1Btn.clicked.connect(self.plan1)
        self.plan2Btn.clicked.connect(self.plan2)
        self.plan3Btn.clicked.connect(self.plan3)
    def plan1(self):
        global planID
        planID=1001
        global duration
        duration=str(1.5)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="1" OR ExerciseID="3" OR ExerciseID="18" OR ExerciseID="9" OR ExerciseID="15"')
        self.fetched=cur.fetchall()
        print(self.fetched)
        self.hide()
        wind5.show()

    def plan2(self):
        global planID
        planID=1002
        global duration
        duration=str(1)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="12" OR ExerciseID="8" OR ExerciseID="10" OR ExerciseID="19" OR ExerciseID="2"')
        self.fetched=cur.fetchall()
        print(self.fetched)
        self.hide()
        wind5.show()
    def plan3(self):
        global planID
        planID=1003
        global duration
        duration=str(2)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="11" OR ExerciseID="5" OR ExerciseID="7" OR ExerciseID="14" OR ExerciseID="6"')
        self.fetched=cur.fetchall()
        print(self.fetched)
        self.hide()
        wind5.show()
```

Again, the exercises from the database were changed, the planID, and the duration. Everything else was the same as the ‘build’ window. Finally, I made the transform window:

This was the same window but again with different durations and exercises, and the sets were 5 and 5 reps of each set.



The code for this window:

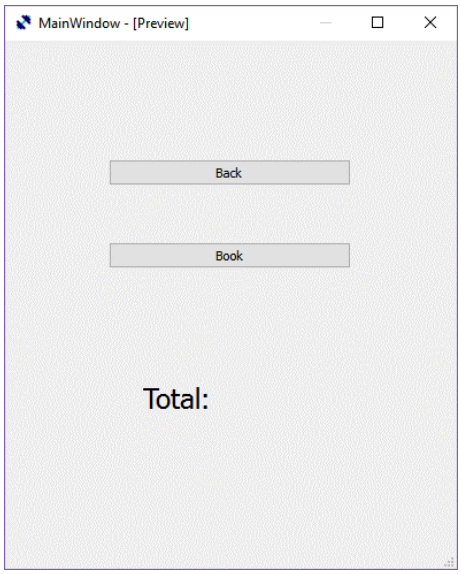
```
class TwelfthWindow(QtGui.QMainWindow, transform_win):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        self.plan1Btn.clicked.connect(self.plan1)
        self.plan2Btn.clicked.connect(self.plan2)
        self.plan3Btn.clicked.connect(self.plan3)
    def plan1(self):
        global planID
        planID=1007
        global duration
        duration=str(1.5)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="1" OR ExerciseID="8" OR ExerciseID="10" OR ExerciseID="15" OR ExerciseID="5"')
        self.fetched=cur.fetchall()
        print(self.fetched)
        self.hide()
        wind5.show()

    def plan2(self):
        global planID
        planID=1008
        global duration
        duration=str(2.5)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="2" OR ExerciseID="8" OR ExerciseID="10" OR ExerciseID="12" OR ExerciseID="19"')
        self.fetched=cur.fetchall()
        print(self.fetched)
        self.hide()
        wind5.show()
    def plan3(self):
        global planID
        planID=1009
        global duration
        duration=str(2.5)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="5" OR ExerciseID="6" OR ExerciseID="7" OR ExerciseID="11" OR ExerciseID="14"')
        self.fetched=cur.fetchall()
        print(self.fetched)
        self.hide()
```

Again, I only change the exercises, duration and planID – everything else stayed the same as the 'cut'/'build' window. When the routine in any of these window is clicked the duration and planID must be stored as we will be using this to store it in the database. For example, using the first workout from the 'cut' window printed this to the console – the planID and the exercises read. I then knew that the button was working and was executing the SQL query:

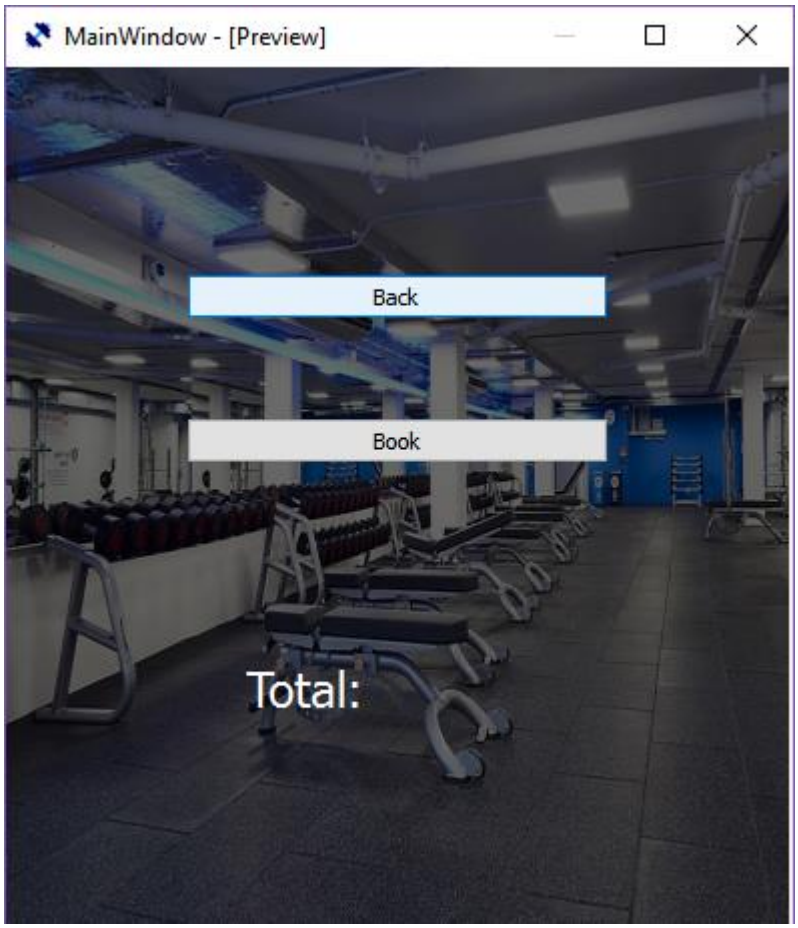
```
1001
[(1, 'Bench Press'), (10, 'Deadlift'), (13, 'Leg Extension'), (15, 'Leg Press'), (18, 'Sit Ups')]
```

Booking workout window

Next, I needed to make another window which confirms all the details. First a simple version:



Then I added a picture and had to change the font to white otherwise it was unreadable in black font. Also, I added a black box with 40% transparency to the background in front of the picture:



I had to read the details from the previous window like the duration of the workout and the payment per hour of that trainer and the exercises picked.

```
cur.execute('SELECT PaymentRates FROM Trainers WHERE TrainerName=?',(trainer_name,))
```

Trainer_name is the input from the combo_box and I was selecting a payment rate for that trainer from the database.

	TrainerID	TrainerName	PaymentRates
	Filter	Filter	Filter
1	1001	Josh Taylor	7.2
2	1002	Hugh Mungus	6.9
3	1003	George Smith	6.55
4	1004	Sam Daniels	7.1
5	1005	Johnny Rogers	8.2
6	1006	Tim Michael	7
7	1007	James East	7.5
8	1008	Patrick Sun	7.3
9	1009	Jim Falcon	8.1

Then I also read the TrainerID from the database so that it can be stored in the workouts table:

```
cur.execute('SELECT TrainerID FROM Trainers WHERE TrainerName=?',(trainer_name,))
```

I then calculated the price:

```
price=float(payment)*float(duration)
```

However, when I tried string concatenation, Python would not let me as price was still a float so I had to convert it to a string i.e. cannot add £ and 15.3, but can add £ and '15.3'.

```
Traceback (most recent call last):
  File "C:\Users\oleks\OneDrive\Documents\Coursework tables pyQt\elprogramerio.py", line 1022, in total
    price='£'+price
TypeError: Can't convert 'float' object to str implicitly
>>> ================================ RESTART ================================
```

The price is the payment e.g. 7.2 multiplied by the duration of the workout e.g. 2 hours.

```
price=str(price)
```

Then, I also added a 0 to make sure it is in price format. I could have rounded it using the round function in Python, but this way was quicker.

```
price='£'+ price +'0'
self.total_lbl.setText(price)
self.total_lbl.setStyleSheet('color: white')
self.total_lbl.resize(76, 78)
```

Next, I read the last WorkoutID from my database and added 1 to this value. This would be our new WorkoutID when it is written to the database:

```
cur.execute("SELECT WorkoutID FROM Workouts ORDER BY WorkoutID DESC LIMIT 1")
workout_id=cur.fetchone()
new_workout_id=workout_id[0]+1
```

I also read the CustomerID of the current user logged in:

Then, I wrote an SQL statement which takes all the variables created throughout the booking screen and stored them into the workouts table in my database:

- New_workout_id is the last workout id in the database + 1

```
cur.execute("SELECT CustomerID FROM Customers WHERE Username=?",(myWindow.username,))
user_id=cur.fetchone()
cur.execute("INSERT INTO Workouts (WorkoutID, ExercisePlanID, CustomerID, TrainerID, DateBooked, Price) VALUES ( ?, ?, ?, ?, ?, ? )",
            (new_workout_id, planID, actual_user_id, trainer_actual, new_date, price))
con.commit()
QtGui.QMessageBox.information(self, "Success", "Workout successfully booked!")
self.hide()
wind3.show()
```

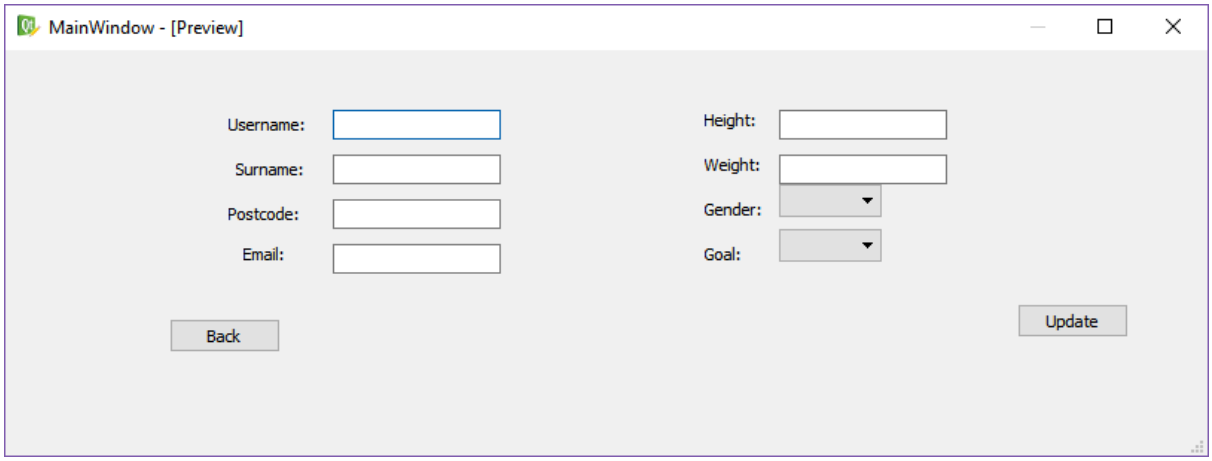
- planID is the global variable from the routine chosen by the user
- actual_user_id is the current user's unique ID
- trainer_actual is the chosen trainer's ID
- new_date is the date chosen of the workout
- price is the the total payment (duration * payment rate of trainer)

Here is an example of a booking stored in the database:

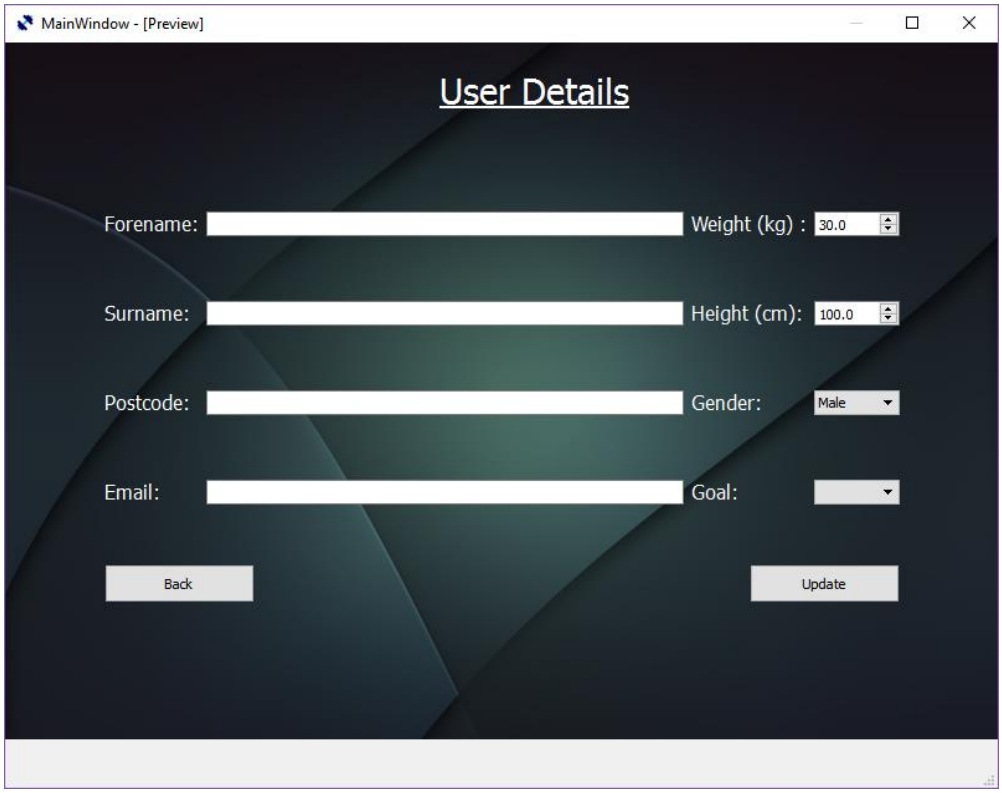
	WorkoutID	ExercisePlanID	CustomerID	TrainerID	DateBooked	Price
	Filter	Filter	Filter	Filter	Filter	Filter
1	9	1002	100003	1006	31/03/2017	7

User details window

The user details window is where the users should be able to view their details and edit them, including password. Firstly, a basic window with input boxes for each detail which can be changed. I also added combo boxes for the gender and goal as this is the same design that was in the registration screen.



I then added a picture to the background, put the widgets into a grid layout and added a change password button:



I set up the window in Python:

```
userDetails_win = uic.loadUiType("UserUpdate.ui")[0]
class  EighthWindow(QtGui.QMainWindow, userDetails_win):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
```

Then I read all the details of the user currently logged in:

```
cur.execute('SELECT Name, Surname, Postcode, Email, Weight, Height, Gender, PreviousWeight,
self.data = cur.fetchall()
    PreviousHeight, GoalID FROM Customers WHERE Username=?',(myWindow.username,))
```

Then I went through each item in the list and assigned a variable to each detail. I made them global so that they can be used throughout the window:

```
for each in self.data:
    global name
    global surname
    global Postcode
    global Email
    global Gender
    name=each[0]
    surname=each[1]
    Postcode=each[2]
    Email=each[3]
    global Weight
    global Height
    Weight=each[4]
    Height=each[5]
    Gender=each[6]
    global PreviousWeight
    global PreviousHeight
    PreviousWeight=each[7]
    PreviousHeight=each[8]
    global GoalID
    GoalID=each[9]
```

I printed these values to make sure they were correct before assigning them to the line edits:

```
current values:  Godwin Cherian HP13 6UW godwincherian@gmail.com 81 188 Male 80  2
```

Forename: Godwin
Surname: Cherian
Postcode: HP13 6UW
Email: godwincherian@gmail.com
Weight: 81
Height: 188
Gender: Male

Previous Weight: 80
GoalID: 2 (“Build”)

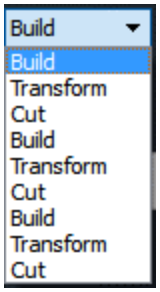
I then made sure that the goalID is not inserted into the line edit but the written format e.g. build

```
def insertvalues(self):
    if GoalID==1:
        Goal="Cut"
    elif GoalID==2:
        Goal="Build"
    elif GoalID==3:
        Goal="Transform"
```

I then inserted all the values into the line edits:

```
self.forename_edit.setText(name)
self.surname_edit.setText(surname)
self.goalCB.clear()
list2=["Build","Transform","Cut"]
self.goalCB.addItems(list2)
u=self.goalCB.findText(Goal)
self.goalCB.setCurrentIndex(u)
self.postcode_edit.setText(Postcode)
self.email_edit.setText(Email)
self.weightEdit_2.setValue(Weight)
self.heightEdit_2.setValue(Height)
```

I had to make sure I was clearing the combo box as before I did this, the combo box started repeating itself:



This was after going into the update screen 3 times – each goal is there 3 times. When the user was finished updating their details I read the values of the line edits and set them as ‘updated’ details:

```
self.updatedForename=self.forename_edit.text()
self.updatedSurname=self.surname_edit.text()
self.updatedPostcode=self.postcode_edit.text()
self.updatedEmail=self.email_edit.text()
self.updatedGoal=self.goalCB.currentText()
self.updatedGender=self.genderCB.currentText()
self.updatedHeight=self.heightEdit_2.text()
self.updatedWeight=self.weightEdit_2.text()
self.updatedWeight=int(float(self.updatedWeight))
```

I had to make sure the updatedGoal is being stored in the database as an ID and not the string:

```
if self.updatedGoal=="Cut":
    self.updatedGoal=1
elif self.updatedGoal=="Build":
    self.updatedGoal=2
elif self.updatedGoal=="Transform":
    self.updatedGoal=3
```

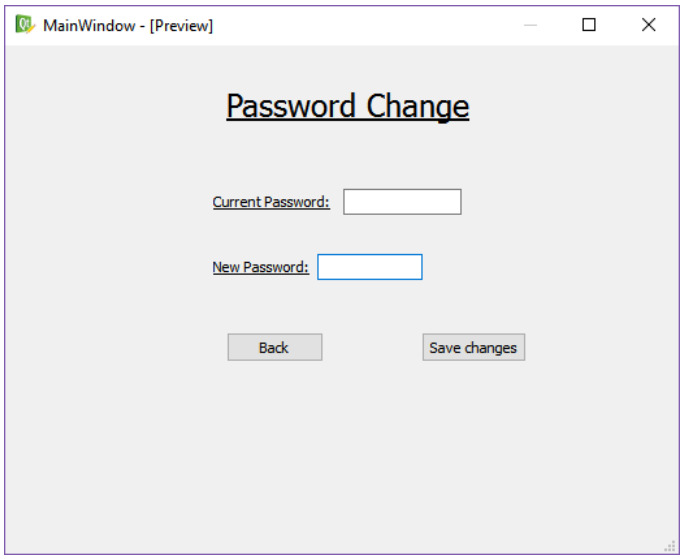
I had to make sure that if the updated weight is equal to the current weight, the current weight is overwritten with the updated weight and the previous weight is updated with the previous weight

```
if self.updatedWeight == Weight:
    cur.execute("""UPDATE Customers SET Name='%s',Surname='%s',Postcode='%s',Email='%s',Weight='%s',Height='%s',Gender='%s', GoalID='%s', PreviousWeight='%s' WHERE Username='%s'"""%
    (self.updatedForename,self.updatedSurname,self.updatedPostcode,self.updatedEmail,self.updatedWeight,self.updatedHeight,self.updatedGender,self.updatedGoal, PreviousWeight,myWindow.username))
    con.commit()
    self.loadDetails()
```

However, if it didn't, I would update the weight with the updated weight but this time the previous weight would be overwritten by the current weight

```
else:
    cur.execute("""UPDATE Customers SET Name='%s',Surname='%s',Postcode='%s',Email='%s',Weight='%s',Height='%s',Gender='%s', GoalID='%s',PreviousWeight='%s' WHERE Username='%s'"""%
    (self.updatedForename,self.updatedSurname,self.updatedPostcode,self.updatedEmail,self.updatedWeight,self.updatedHeight,self.updatedGender,self.updatedGoal, Weight,myWindow.username))
    con.commit()
    self.loadDetails()
```

For the password change I made an additional window separate to the registration window. Firstly, I made a basic version of the GUI:



Then I added a picture and added a grid layout. I change the font to white so it is easier to read.
I then linked the window in Python:

```
passwordreset1 = uic.loadUiType("PasswordReset1.ui")[0]
class FourteenthWindow(QtGui.QMainWindow, passwordreset1):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
```

Then I read the password from the database for the current user:

```
cur.execute('SELECT Password FROM Customers WHERE Username=?',(myWindow.username,))
self.fetched=cur.fetchall()
self.password1=self.password_edit.text()
hashed_password=self.fetched[0][0]
self.password=hashlib.md5(self.password1.encode('utf-8')).hexdigest()
```

The password was stored as self.fetched. Then self.password1 was the current password in the first line edit of the GUI. The hashed_password was then taken from the list to give the hash stored in the database. Then I hashed the current password input so that it can be checked against the hash from the database.
I then compared the hash from the database to the hash of the input of the user. If they matched, I updated the password of the user with the hash of the new password input. Con.commit() is used to save the changes to the database. If they didn't match an error is displayed:

```
if hashed_password==self.password:
    self.newpassword1=self.newpassword.text()
    newest=hashlib.md5(self.newpassword1.encode('utf-8')).hexdigest()
    cur.execute("""UPDATE Customers SET Password='%s' WHERE Username='%s'"""%(str(newest), myWindow.username))
    con.commit()

else:
    QtG                                                                    incorrect")
```

BMI Calculator window

Firstly, I tested a calculator using the console in Python.

```
Height=int(input("enter your height"))
Weight=int(input("enter your weight"))
BMI=Weight/(Height*Height)
```

I took input of the height and weight of the user as integers and calculated BMI using the BMI formula (Weight/(Height*Height)). Note that the height is in metres and the weight is in kilograms.

```
Height=Height/100
```

Since the height in the database is stored in centimetres I had to convert to metres Then I did 7 'if statements' and depending on the BMI, the status of the health of the user was printed:

```
if BMI <=19.5:
    print("Your BMI is:", BMI, "You are underweight")
elif BMI >19.5 and BMI<25:
    print("Your BMI is:", BMI, "You are healthy")
elif BMI >25 and BMI <30:
    print("Your BMI is:", BMI, "You are overweight")
elif BMI>30 and BMI <35:
    print("Your BMI is:", BMI, "You are obese")
elif BMI>35 and BMI <40:
    print("Your BMI is:", BMI, "You are severly obese")
elif BMI>40:
    print("Your BMI is:", BMI, "You are morbidly overweight")
```

Gave this output:

```
enter your height180
enter your weight20
Your BMI is: 6.17 You are underweight
```

I decided to round the BMI to 2 decimal places:

```
BMI=round((BMI),2)
```

Next, I decided to take the weight and height from the database instead of having to input this.

```
import sqlite3
conn = sqlite3.connect('bodybuildingv3.db')
cur = conn.cursor()
cur.execute("SELECT Weight, Height FROM Customers WHERE Username='godwin2'")
data = cur.fetchone()
print(data)
```

Gave this output:

godwin2	517b08dafd8a...	Godwin	Cherian	3/01/1999	HP13 6UW	godwincheria...	81	188


```
...
(81, 188)
>>>
```

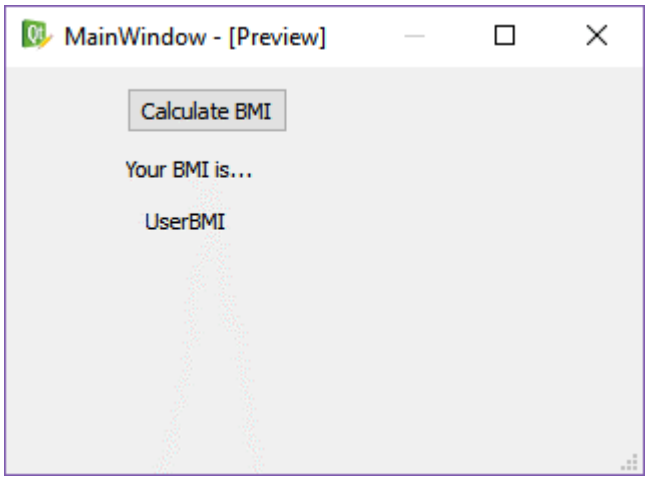
Then I assigned from the list the weight and height:

```
Weight=data[0]
Height=data[1]
print(Weight, Height)

81 188
>>>
```

```
Your BMI is: 22.92 You are healthy
>>>
```

And then the rest was calculated for the BMI

I then created a basic GUI for my BMI window:



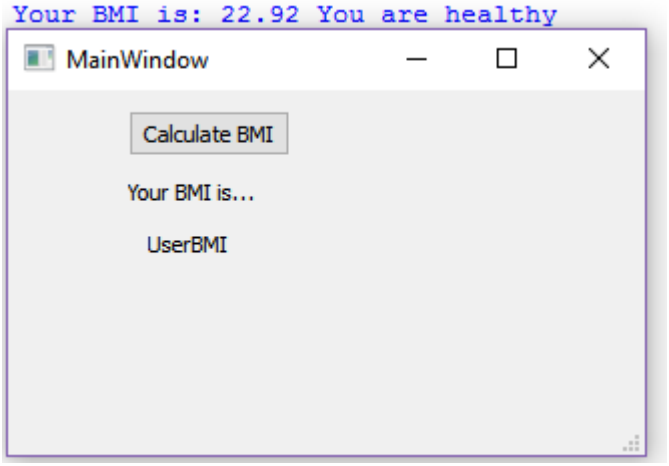
I then linked it in Python

```
BMI_win= uic.loadUiType("BMI.ui")[0]
class SeventhWindow(QtGui.QMainWindow, BMI_win):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
```

Then I linked the button to a function called 'calculate' when the button is pressed, and put the formula and the if statements inside this function:

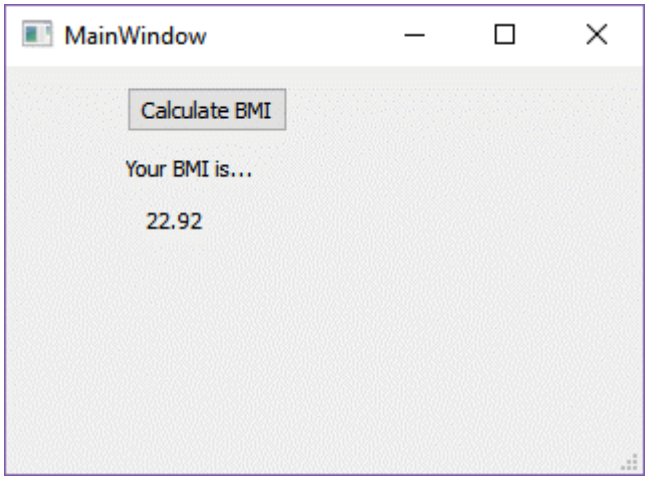
```
        self.calculateBtn.clicked.connect(self.calculate)
    def calculate(self):
        cur.execute("SELECT Weight, Height FROM Customers WHERE Username='godwin2'")
        data = cur.fetchone()
        Weight=data[0]
        Height=data[1]
        Height=Height/100
        BMI=Weight/(Height*Height)
        BMI=round((BMI),2)
        if BMI <=19.5:
            print("Your BMI is:", BMI, "You are underweight")
        elif BMI >19.5 and BMI<25:
            print("Your BMI is:", BMI, "You are healthy")
        elif BMI >25 and BMI <30:
            print("Your BMI is:", BMI, "You are overweight")
        elif BMI>30 and BMI <35:
            print("Your BMI is:", BMI, "You are obese")
        elif BMI>35 and BMI <40:
            print("Your BMI is:", BMI, "You are severly obese")
        elif BMI>40:
            print("Your BMI is:", BMI, "You are morbidly overweight")
```

```
Your BMI is: 22.92 You are healthy
```

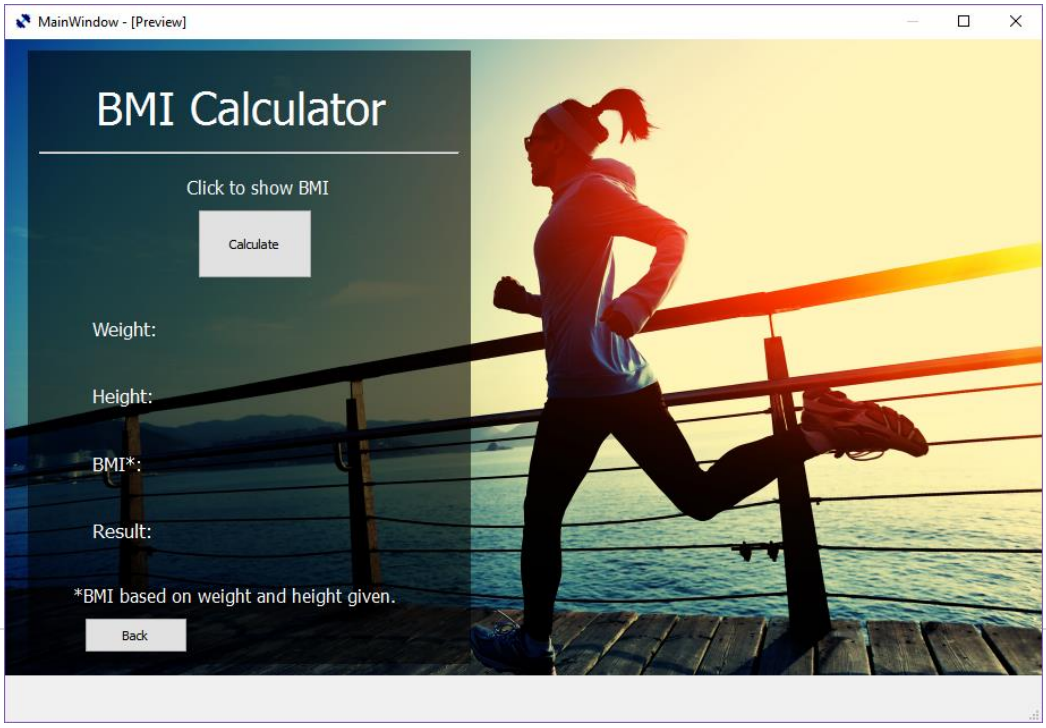


Next, I decided to display the BMI on the 'UserBMI' label instead of printing it to the console:

```
self.userBMI.setText(str(BMI))
```



I then decided to make the GUI more visually appealing:

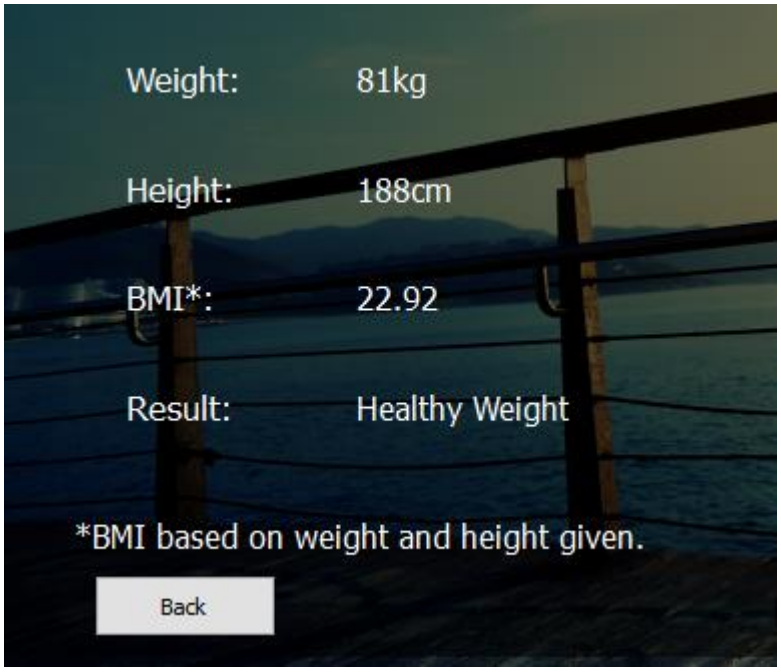


I then added the weight and height labels, as well as the status health label:

```
self.label_10.setText(str(dataWeight)+"kg")
self.label_10.setStyleSheet('color: white')
self.label_11.setText(str(dataHeight)+"cm")
self.label_11.setStyleSheet('color: white')
self.label_12.setText(str(self.BMI))
self.label_12.setStyleSheet('color: white')
```

This was dependent on the BMI i.e. different BMI gives different label

```
self.label_13.setText("Underweight")
```



Then I decided to make the graph comparing the previous weight and the current weight of the user:

I added a button and label.

Then, I read the previous weight and current weight of the user from the database

```
cur.execute('SELECT PreviousWeight, Weight, from Customers WHERE Username=?',(myWindow.username,))
self.data = cur.fetchall()
global previousweight
global weight
previousweight=self.data[0][0]
weight=self.data[0][1]
```

I then went to the first index of the list and stored this as the previous weight and the next index of the list as the current weight

I then found code in HTML for a bar graph on the internet:

```
                template='''<!DOCTYPE html>
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
    </style>
  </head>
  <body>
    <button onclick="test();">Launch</button>
    <p id="out"> </p>
    <canvas id="myCanvas" width="150" height="425"></canvas>

    <script>
       var canvas = document.getElementById('myCanvas');
      var context = canvas.getContext('2d');
      var a=new Array(5, 8);
function test(){

      var arrayLength = a.length;
for (var i = 0; i < arrayLength; i++) {

    //Do something
  draw_bar(50,i);
  //document.getElementById("out").innerHTML+=","+a[i];
}


      }
      function draw_bar(size,ind){

      context.beginPath();

      context.rect(size*ind, canvas.height-size/16*a[ind], size,size/8*a[ind]);
      context.fillStyle = 'yellow';
      context.fill();
      context.lineWidth = 4;
      context.strokeStyle = 'black';
      context.stroke();

         context.font = "20px Arial";
        context.fillStyle = "red";
        context.textAlign = "centre";
context.fillText(a[ind],size*ind+size/2-10,canvas.height-size/16*a[ind]-10);
      }
    </script>
</body>
</html> '''
```

I then replaced the new Array values of 5, 8 with the previous weight and current weight:

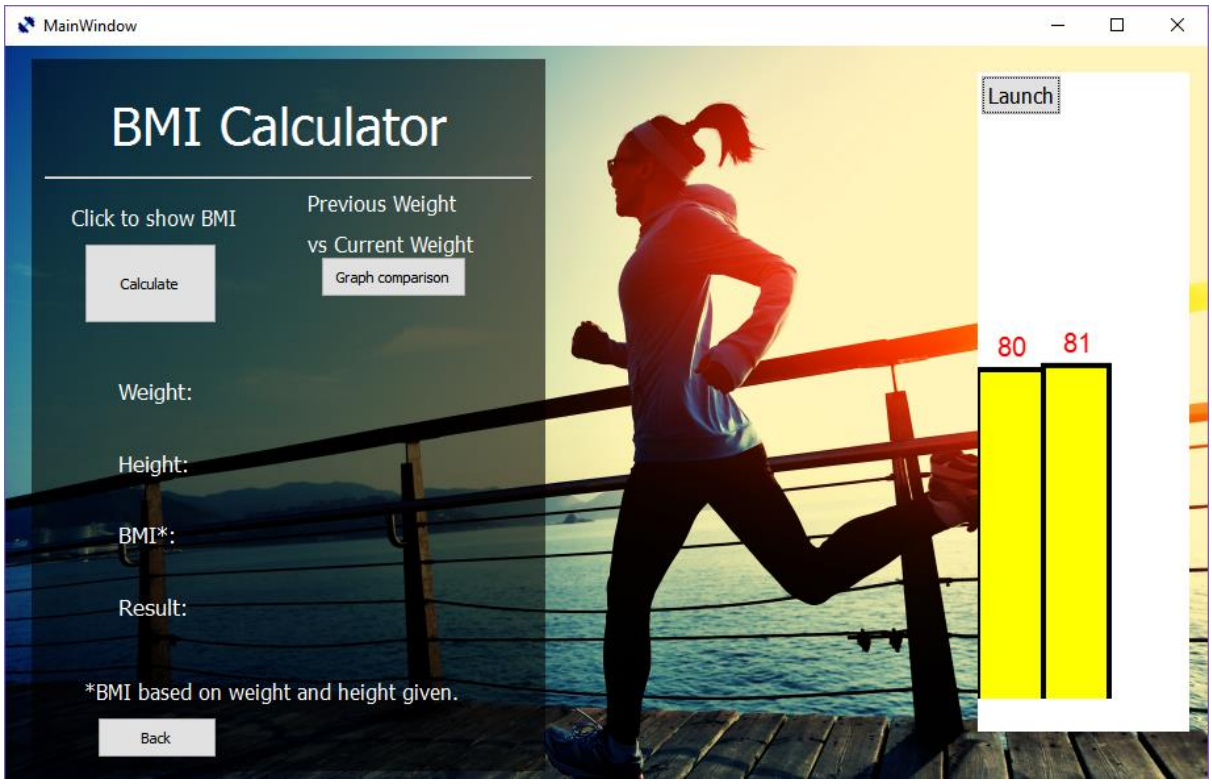
```
b=(str(previousweight)+", "+str(weight))
url=template.replace("5, 8",b)
print(url)
self.webView.setHtml(url)
```

I could not just replace 5, 8 with previous weight and current weight like this:

```
url=template.replace("5, 8", previousweight, weight)
```

This was because there was a comma between 5 and 8 and I had to set b as a string of the previous weight and current weight being joined together:

```
print(b)
 80, 81
```



This is a WebView in PyQt that I used to display the webpage that the graph was being displayed on:

Password reset window

The password reset was done by first deciding what details would be used to reset the password. I decided the username and the email of a user would be used as access to the email would be needed to check the reset password, therefore these 2 inputs would make for a secure solution. Firstly, I imported the libraries needed to connect to an email server.

```
import smtplib
from smtplib import SMTP
```

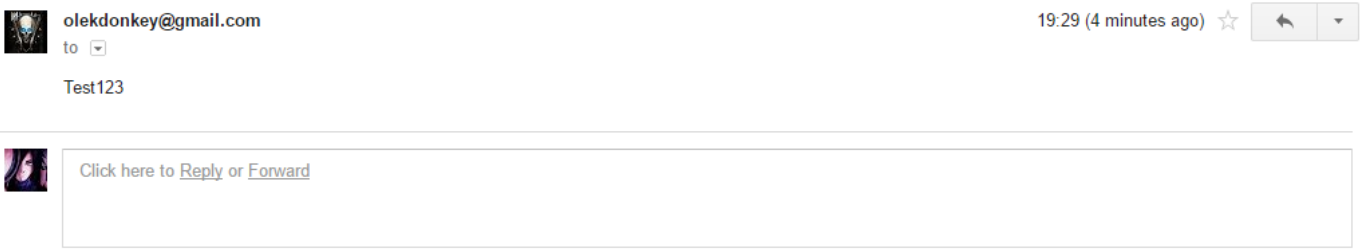
Then I connected to Google's SMTP email server:

```
server = smtplib.SMTP('smtp.gmail.com', 587)
```

Then I logged into my email olekdonkey@gmail.com with my password (hidden). Then I set the message as 'Test123'

```
server = smtplib.SMTP('smtp.gmail.com', 587)
server.starttls()
server.login("olekdonkey@gmail.com", password)
msg = 'Test123'
server.sendmail("olekdonkey@gmail.com", "boskiclown@gmail.com", msg)
print("email sent")
server.quit()
```

Then I sent the email using. sendmail using my email as the sender email and 'boskiclown@gmail.com as the recipient email and the message at the end. Then I printed email sent to confirm that the program had run up to this line and then I quit the server.



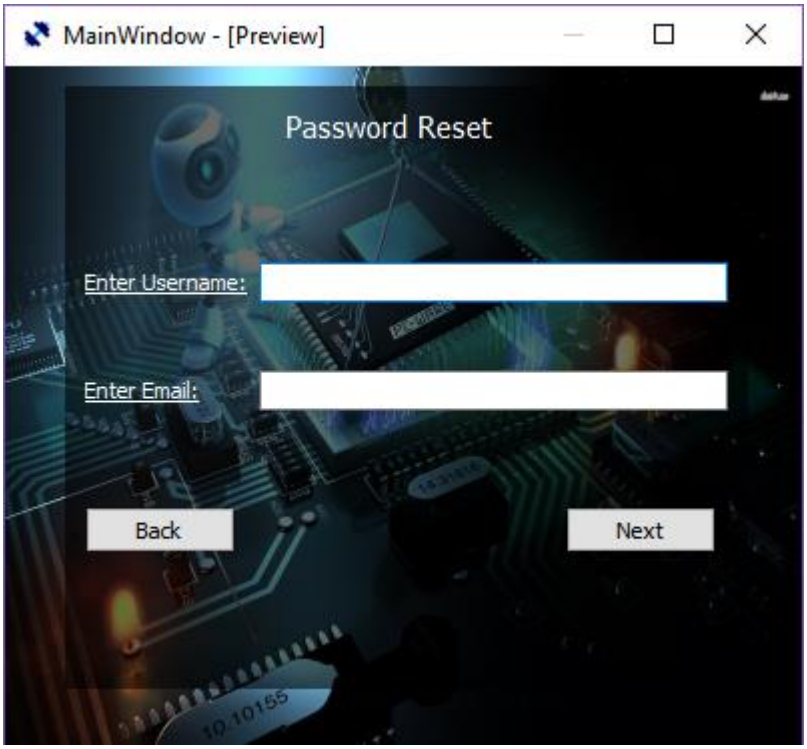
Next, I decided to create a random password to be used as the reset password for the user

```
string=""
x=("ABCDEFGHIJKLMNOPQRSTUVWXYZ123456789")
counter=0
while counter<10:
    k=random.choice(x)
    string=k+string
    x = x.replace(k, "")
    counter=counter+1
print(string)
```

First, string is created as an empty string. Then x is a string with the letters of the alphabet and number 1-9. Counter is set to 0. Random. choice chooses a random letter from 'x' – the letters and numbers. Then string is set to k (the random choice) + the string. Then whatever number or letter is chosen from 'x' is removed to prevent repetition of the same letters. The counter is incremented by 1 and the loop is run again until counter is equal to 10, and then the string is printed.

```
9B2USAP6ZX
>>>
```

The password reset window was then developed further from the basic GUI when I created the login window to this:



I added a picture to the background, and black box with 40% transparency. I also put the labels and line edits into a grid view for a more professional look.

I then assigned the line edit inputs to variables

```
self.username1=self.username.text()
self.email1=self.email.text()
```

Then I read the username and email from the customers table where the username was equal to the input of the user (self. username1).

```
cur.execute('SELECT Username, Email FROM Customers WHERE Username=?',(self.username1,))
details=cur.fetchall()
```

Then I checked if the email from the database was the same as the input email by the user:

```
if self.email1==self.email2:
```

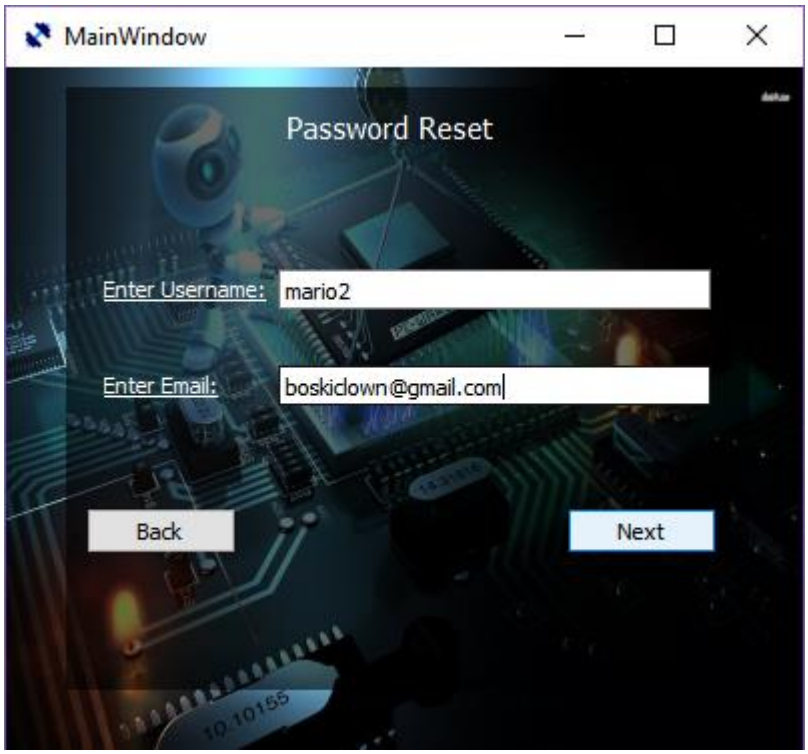
If they did I generated the random string for the reset password. Then I hashed this randomly generated string

```
hashed_string=hashlib.md5(string.encode('utf-8')).hexdigest()
```

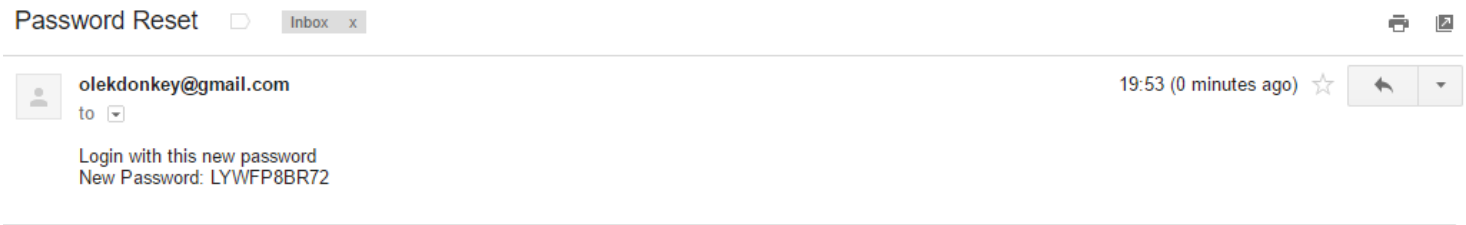
I then sent the email to the email given in the input in the line edit and update the user's password with the hashed string.

```
msg = 'Subject: {}\n\n{}'.format("Password Reset", "Login with this new password"+"\n"+"New Password: "+string)

        server.sendmail("olekdonkey@gmail.com", self.email2, msg)

cur.execute("UPDATE Customers SET Password='%s' WHERE Username='%s'"%(hashed_string, self.username1))
con.commit()
```

When the program was run, and this input was used for the username and email

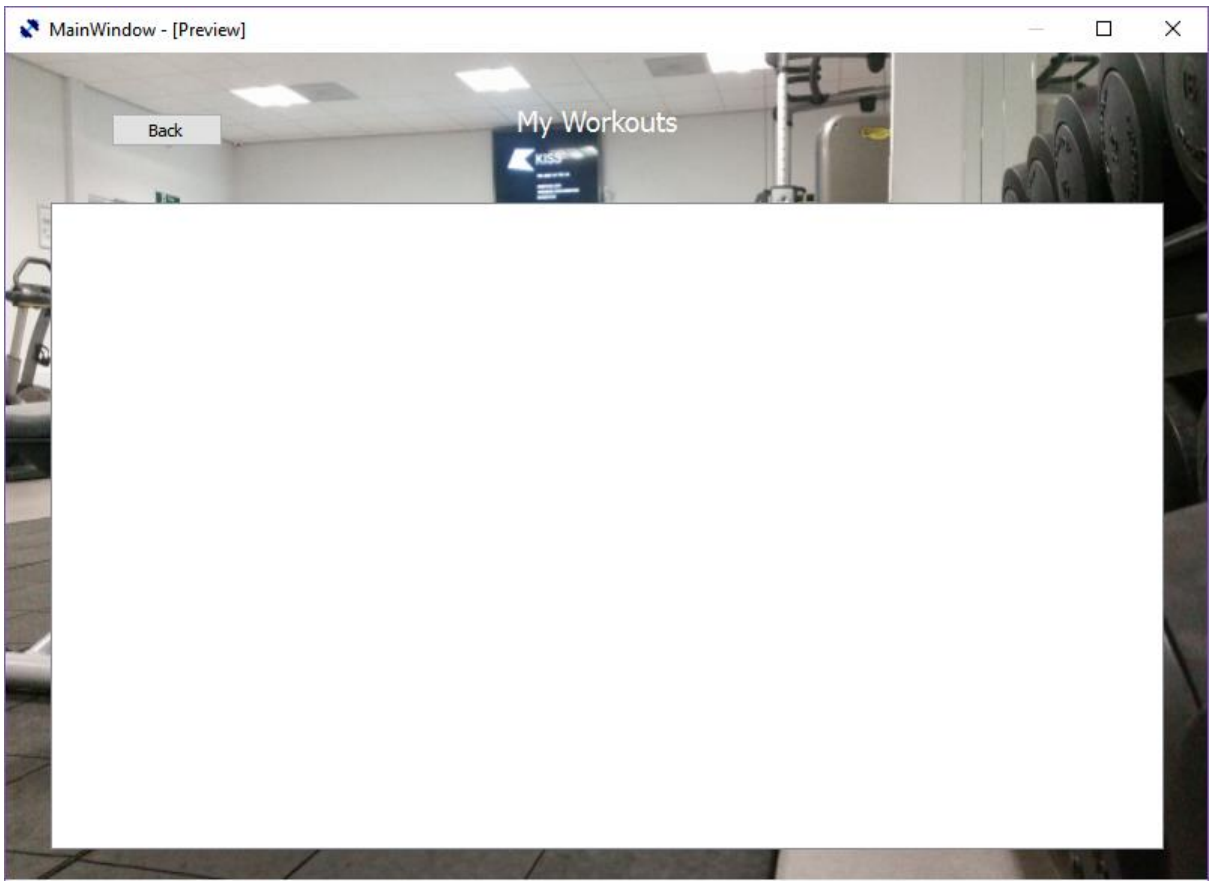


This email was shown in my inbox:



My workouts window

My workouts window displays the workouts for a user. First I created the GUI for this window



The square in the middle is the table view in PyQt which displays the database in table format using columns and rows.

I used code from the internet to help me display the details onto the table view. First I read the CustomerID of the currently logged in user. Then I read all the details from the workouts table where the CustomerID is equal to the current user's CustomerID:

```
cur.execute("""Select CustomerID from Customers WHERE Username=?""",(myWindow.username,)
cust_id=cur.fetchone()
cust_id1=cust_id[0]
cur.execute('select * from Workouts WHERE CustomerID=?',(cust_id1,))
```

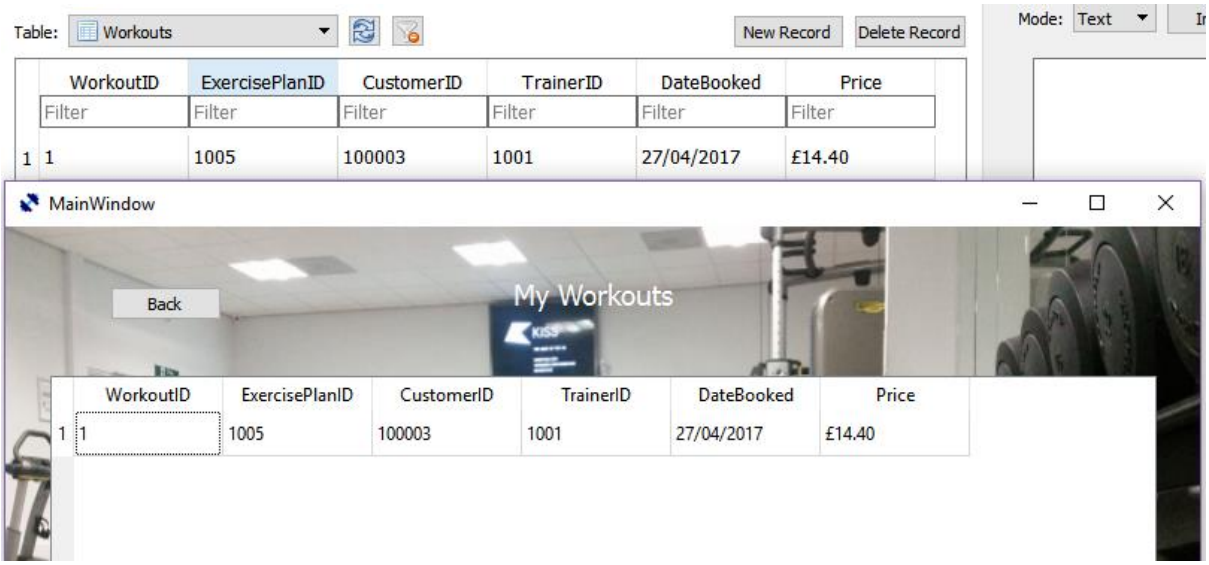
Then a loop was set to add each of the details into each individual row. This was to make sure the number of columns in the table view matches the number of columns in the database:

```
if len(self.data)>0:
    for i in range(len(self.data)-1,-1):
        self.data.pop(i)
        self.model.removeRow(index.row(self.data[i]))
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)
```

I then named the columns according to what data they were displaying:

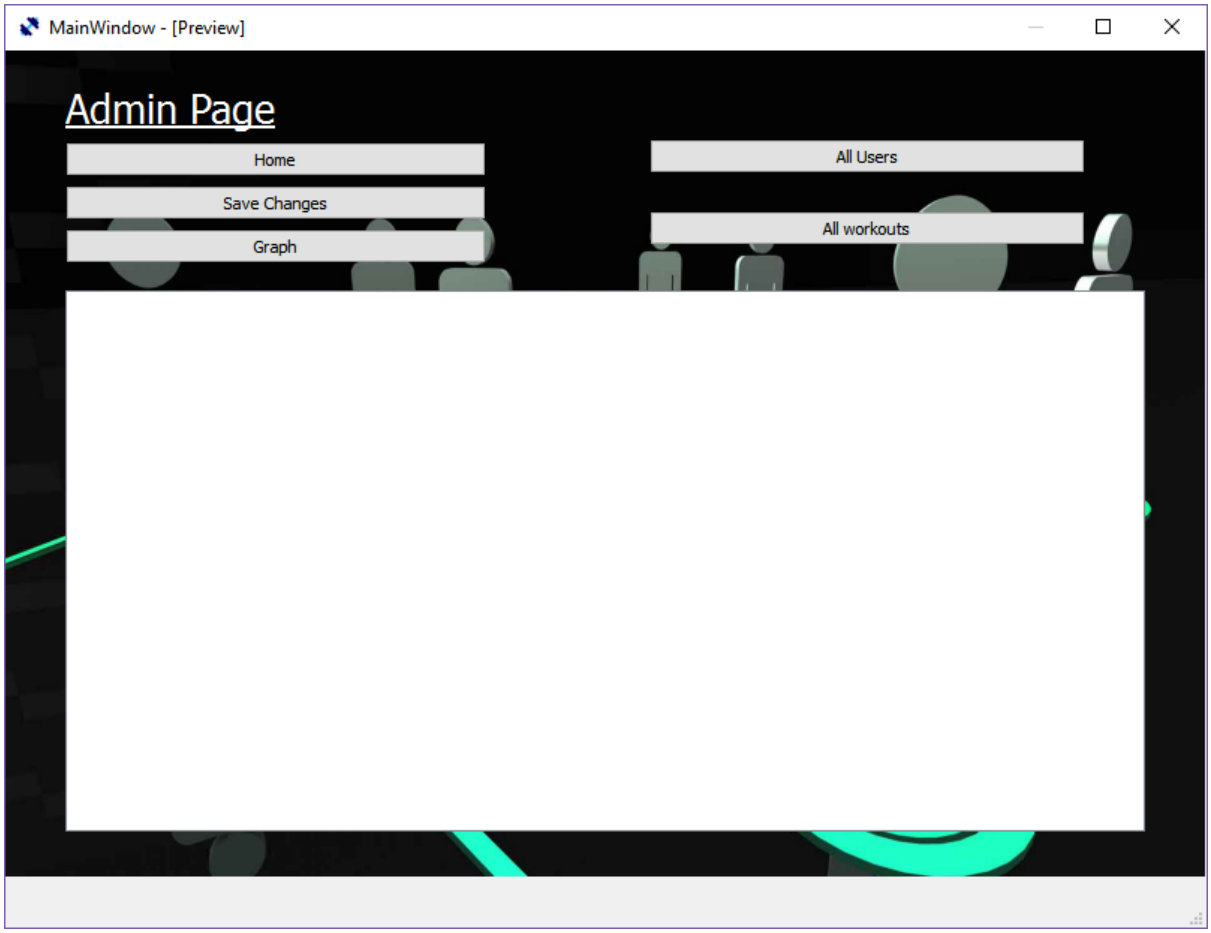
```
self.model.setHeaderData(0, QtCore.Qt.Horizontal, "WorkoutID")
self.model.setHeaderData(1, QtCore.Qt.Horizontal, "ExercisePlanID")
self.model.setHeaderData(2, QtCore.Qt.Horizontal, "CustomerID")
self.model.setHeaderData(3, QtCore.Qt.Horizontal, "TrainerID")
self.model.setHeaderData(4, QtCore.Qt.Horizontal, "DateBooked")
self.model.setHeaderData(5, QtCore.Qt.Horizontal, "Price")
```

This gave the following output:



Admin Window

The admin window was developed from the initial window made during the login screen



The buttons were not linked at this stage yet. Firstly, I set up the window in Python:

```
admin_win = uic.loadUiType("admin_pg.ui")[0]
class SixthWindow(QtGui.QMainWindow, admin_win):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
```

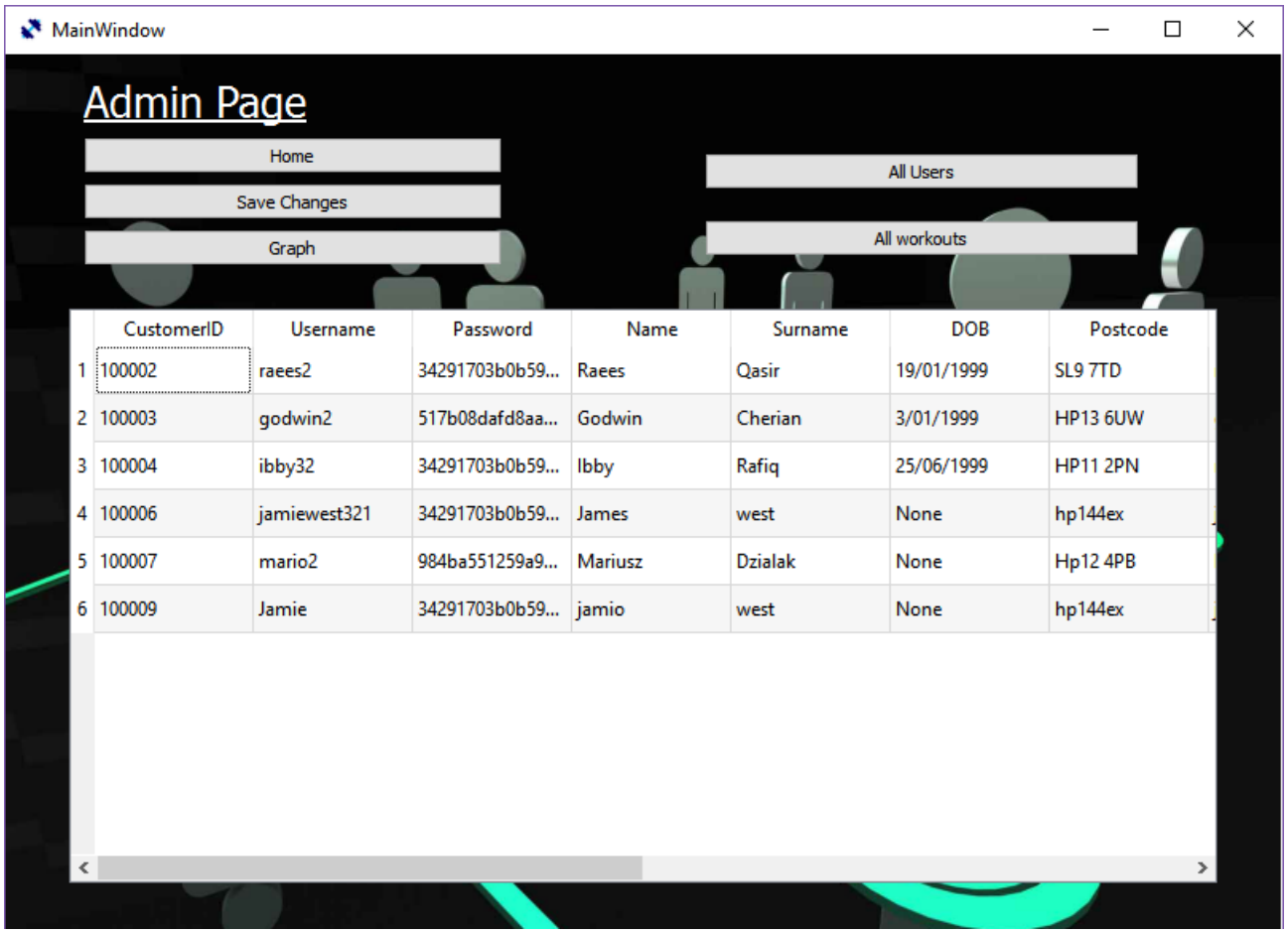
I then loaded the data in the same way that I loaded it into the table view in my workouts window, but this time I was selecting everything from the customers table

```
s='select * from Customers'
cur.execute(s)

if len(self.data)>0:
    for i in range(len(self.data)-1,-1):
        print("removing row",i)
        self.data.pop(i)
        self.model.removeRow(index.row(self.data[i]))
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)
```

I then named all the columns for the data:

```
self.model.setHeaderData(0, QtCore.Qt.Horizontal, "CustomerID")
self.model.setHeaderData(1, QtCore.Qt.Horizontal, "Username")
self.model.setHeaderData(2, QtCore.Qt.Horizontal, "Password")
self.model.setHeaderData(3, QtCore.Qt.Horizontal, "Name")
self.model.setHeaderData(4, QtCore.Qt.Horizontal, "Surname")
self.model.setHeaderData(5, QtCore.Qt.Horizontal, "DOB")
self.model.setHeaderData(6, QtCore.Qt.Horizontal, "Postcode")
self.model.setHeaderData(7, QtCore.Qt.Horizontal, "Email")
self.model.setHeaderData(8, QtCore.Qt.Horizontal, "Weight")
self.model.setHeaderData(9, QtCore.Qt.Horizontal, "Height")
self.model.setHeaderData(10, QtCore.Qt.Horizontal, "Gender")
self.model.setHeaderData(11, QtCore.Qt.Horizontal, "Goal")
self.model.setHeaderData(12, QtCore.Qt.Horizontal, "PreviousWeight")
self.model.setHeaderData(13, QtCore.Qt.Horizontal, "PreviousHeight")
```

This gave the following output:



	CustomerID	Username	Password	Name	Surname	DOB	Postcode
1	100002	raees2	34291703b0b59...	Raees	Qasir	19/01/1999	SL9 7TD
2	100003	godwin2	517b08dafd8aa...	Godwin	Cherian	3/01/1999	HP13 6UW
3	100004	ibby32	34291703b0b59...	Ibby	Rafiq	25/06/1999	HP11 2PN
4	100006	jamiewest321	34291703b0b59...	James	west	None	hp144ex
5	100007	mario2	984ba551259a9...	Mariusz	Dzialak	None	Hp12 4PB
6	100009	Jamie	34291703b0b59...	jamio	west	None	hp144ex

Table: Customers

New Record | Delete Record

	CustomerID	Username	Password	Name	Surname	DOB	Postcode	Email	Weight	Height	Gender	GoalID	PreviousWeight	PreviousHeight
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	100009	Jamie	34291703b0b...	jamio	west	None	hp144ex	jamiewest664...	81	187	Male	2	NULL	NULL
2	100007	mario2	984ba551259...	Mariusz	Dzialak	None	Hp12 4PB	boskiclown@g...	81	187	Male	1	90	
3	100006	jamiewest321	34291703b0b...	James	west	None	hp144ex	jamiewestt@...	75	185	Female	2		
4	100004	ibby32	34291703b0b...	Ibby	Rafiq	25/06/1999	HP11 2PN	madibby@gm...	93	177	Male	3	103	
5	100003	godwin2	517b08dafd8a...	Godwin	Cherian	3/01/1999	HP13 6UW	godwincheria...	81	188	Male	2	80	
6	100002	raees2	34291703b0b...	Raees	Qasir	19/01/1999	SL9 7TD	rqasir@hotma...	72	182	Female	1		

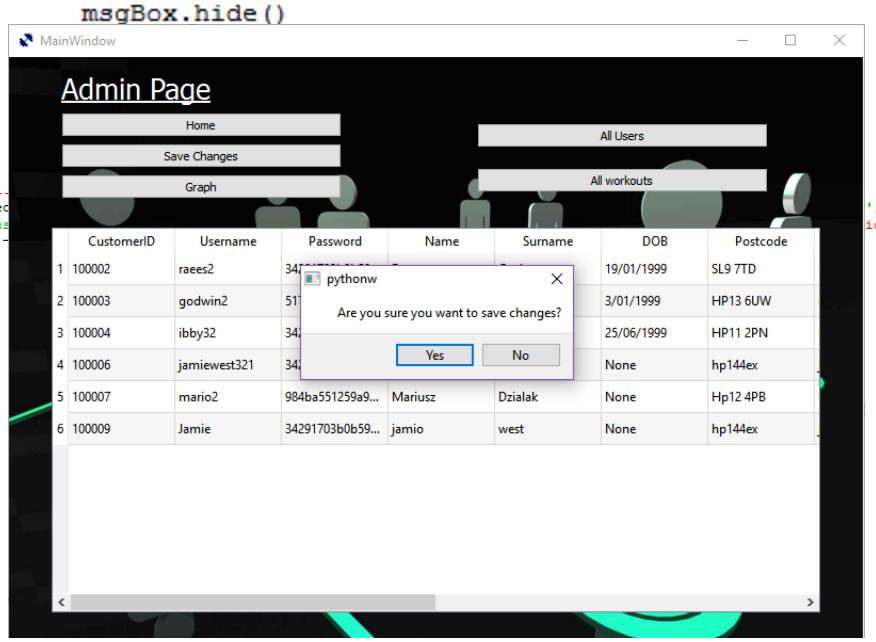
Then I made the updating for the admin, so that any change made to the data in this table view would be saved to the database.

```
index = self.tableView.currentIndex() #returns currently selected cell
r=index.row() #returns the row of the currently selected cell
c=index.column()#returns the column of the currently selected cell
cell_contents=index.data()#returns the contents of the currently selected cell
id=self.model.data(self.model.index(index.row(), 0))
print(id,self.model.data(self.model.index(index.row(), 1)))
name=self.model.data(self.model.index(index.row(), 1))
test1=self.model.data(self.model.index(index.row(), 2))
test2=self.model.data(self.model.index(index.row(), 3))
test3=self.model.data(self.model.index(index.row(), 4))
test4=self.model.data(self.model.index(index.row(), 5))
test5=self.model.data(self.model.index(index.row(), 6))
test6=self.model.data(self.model.index(index.row(), 7))
test7=self.model.data(self.model.index(index.row(), 8))
test8=self.model.data(self.model.index(index.row(), 9))
test9=self.model.data(self.model.index(index.row(), 10))
test10=self.model.data(self.model.index(index.row(), 11))
```

This code goes through each index in a row and assigns it to a variable. Then I ran an update query for these variables:

Then a message box to ask the admin if they want to save the changes they have made:

```
msgBox = QtGui.QMessageBox()
msgBox.setText('Are you sure you want to save changes?')
msgBox.addButton(QtGui.QPushButton('Yes'), QtGui.QMessageBox.YesRole)
msgBox.addButton(QtGui.QPushButton('No'), QtGui.QMessageBox.NoRole)
ret = msgBox.exec_()
if ret==0:
    con.commit()
    QtGui.QMessageBox.information(self, "Success", "Changes saved")
    con.close()
else:
    msgBox.hide()
```



Gave the following output when the save changes button was clicked:

Next I decided to make the graph which displays the percentage of male users compared to female users. First, I ran an SQL query which selected all unique data and gave the amount of times this appears in a certain column

```
cur.execute('select distinct Gender, count(Gender) as CountOf from Customers group by Gender')
```

```
[('Female', 2), ('Male', 4)]
```

Then I assigned the number of female and male accounts to a variable

```
total=(l[0][1])+(l[1][1])
print(total)
```

L [0][1] is the female count, and L [1][1] is the male count. I added them together to give me the total I needed for the male and female count. Then I imported the library needed to plot the graph:

```
import matplotlib.pyplot as plt
```

I then calculated the angle needed for each gender. The labels are Female (l[0][0]) and Male (l[1][0])

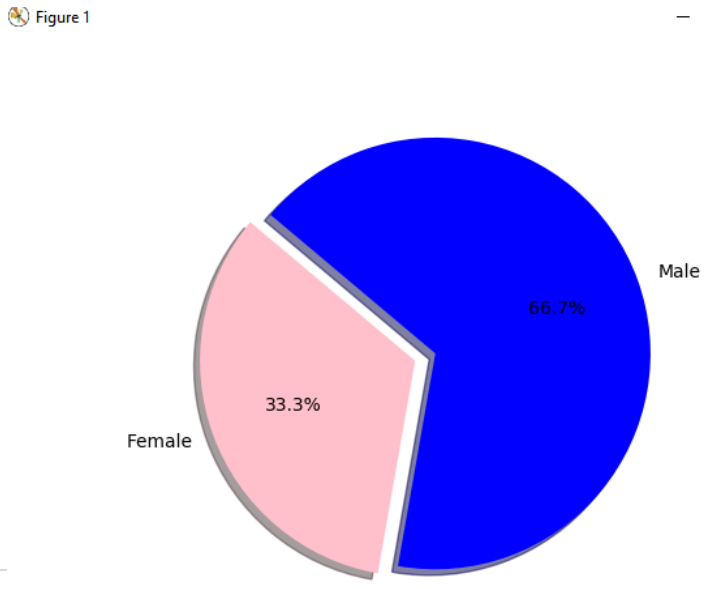
```
male_per=(l[1][1]/total)*360
female_per=(l[0][1]/total)*360
labels = l[0][0], l[1][0]
```

Then a list called 'sizes' is set with the male angle and the female angle. The colours are pink for female and blue for male. The first slice of the pie is exploded.

```
sizes = [female_per, male_per]
colors = ['pink', 'blue']
explode = (0.1, 0)  # explode 1st slice
```

Then the pie is plotted using the sizes list as the angles, exploded by 0.1 on the first slice, labels are Female and Male, colours are taken from the colours list, shadow is set to true i.e. show shadow and the start angle is 140 - right side of the pie. Plt.show() is used to display the graph.

```
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.show()
```



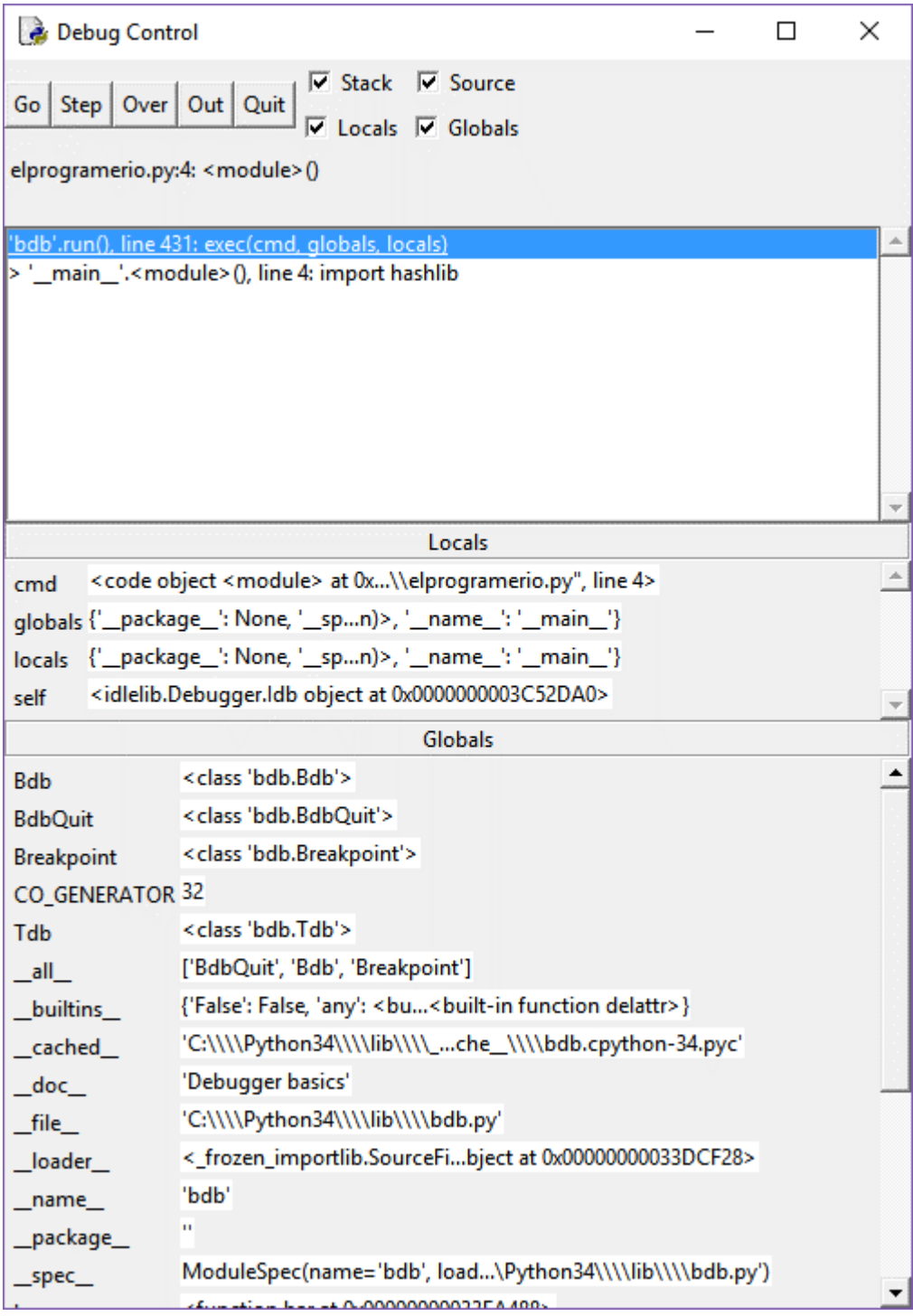
The last part of the admin window was to use an inner join SQL query to display all the workouts:

```
s='select Workouts.WorkoutID,Workouts.DateBooked, Customers.Name from Workouts inner join Customers on Workouts.CustomerID=Customers.CustomerID'
cur.execute(s)
```

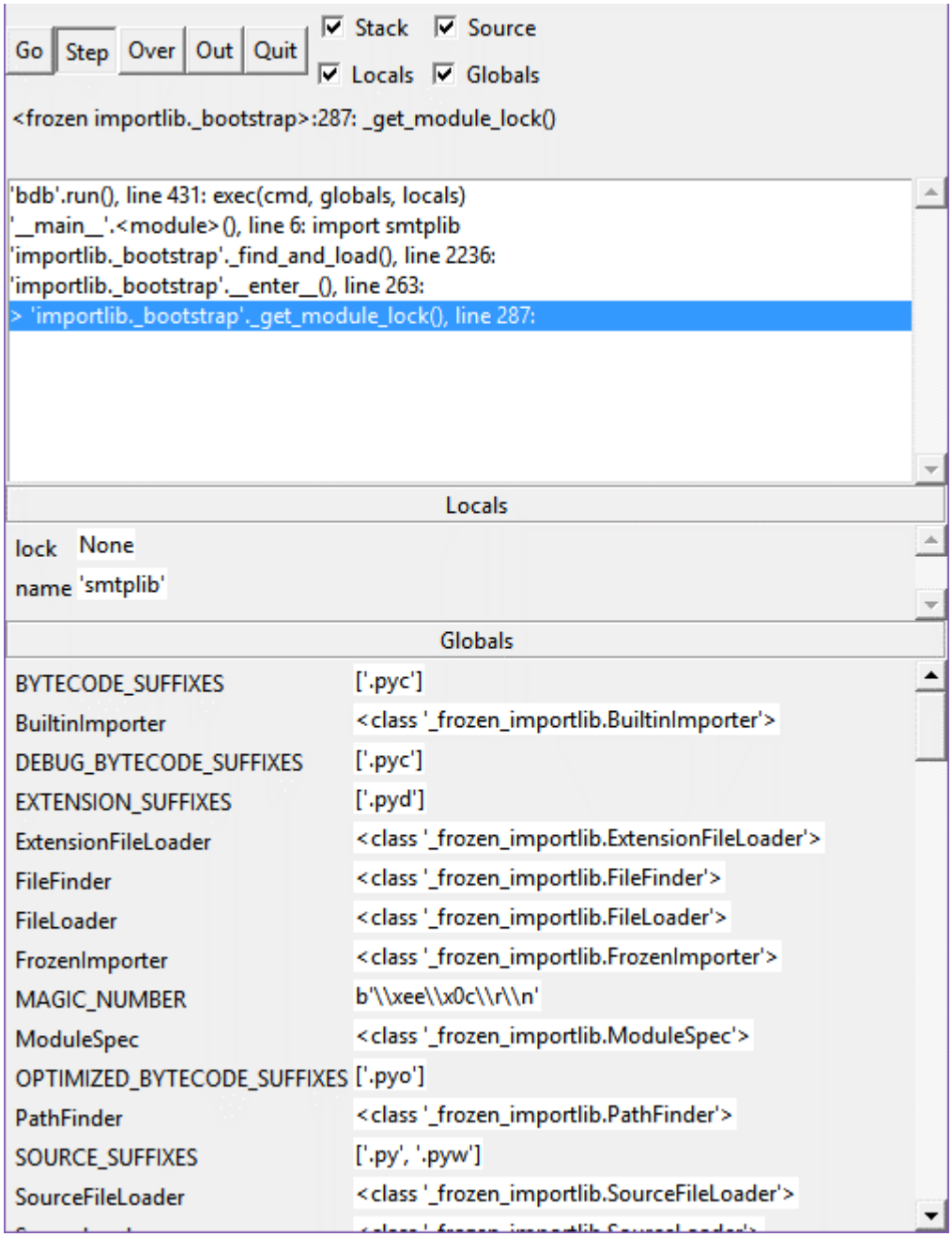
Then this is displayed in this format:

	WorkoutID	DateBooked	Name
1	1	27/04/2017	Godwin

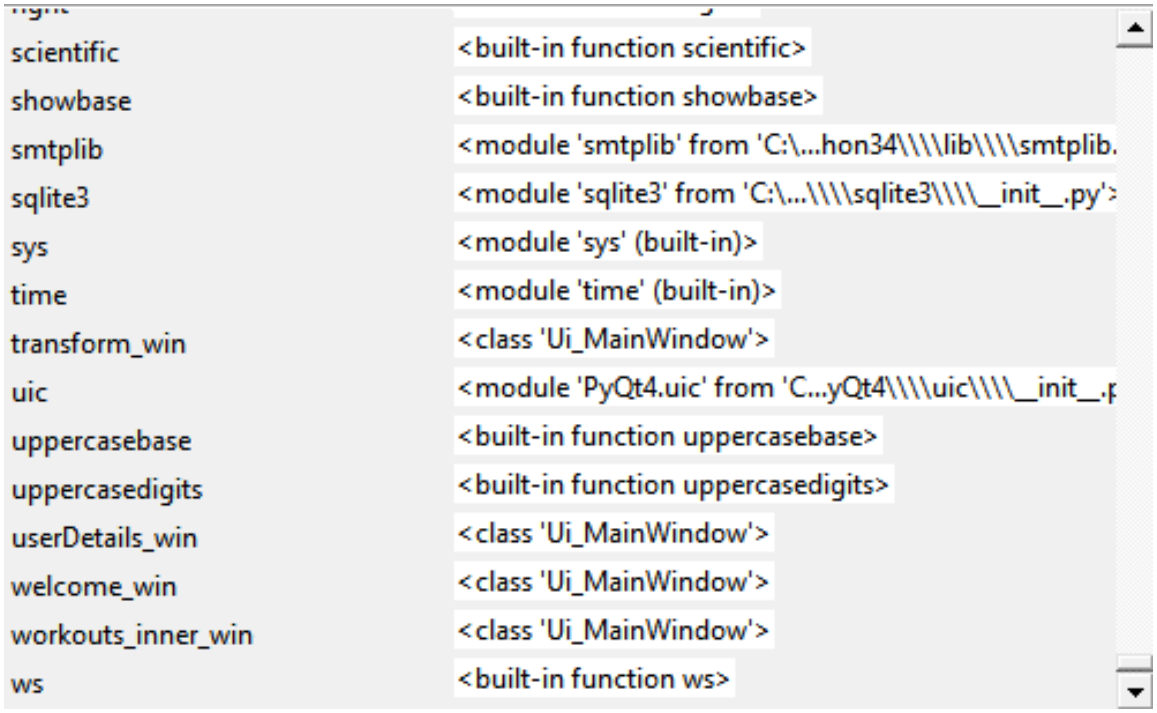
Using the debugger:

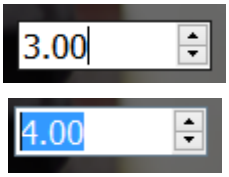
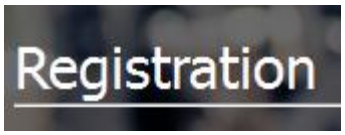
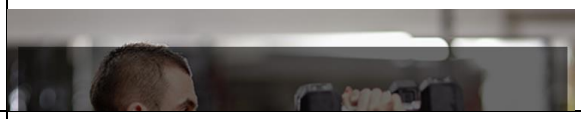
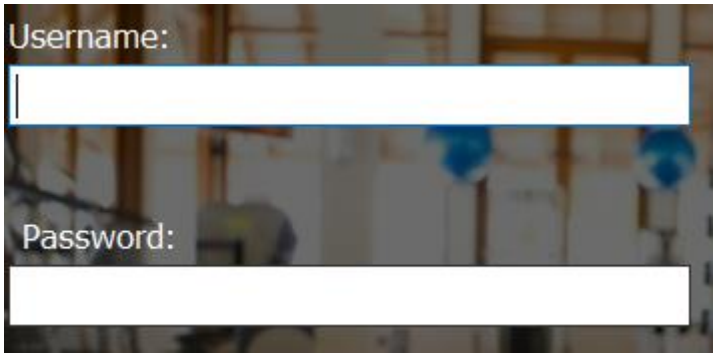


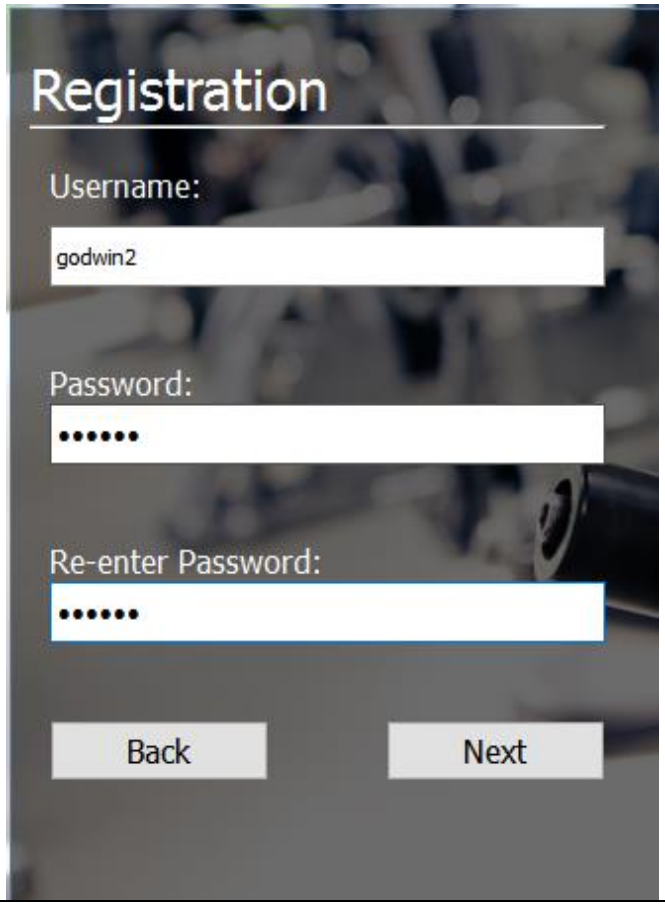
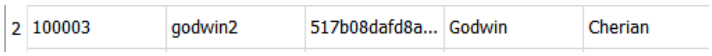
Stepping through line by line:

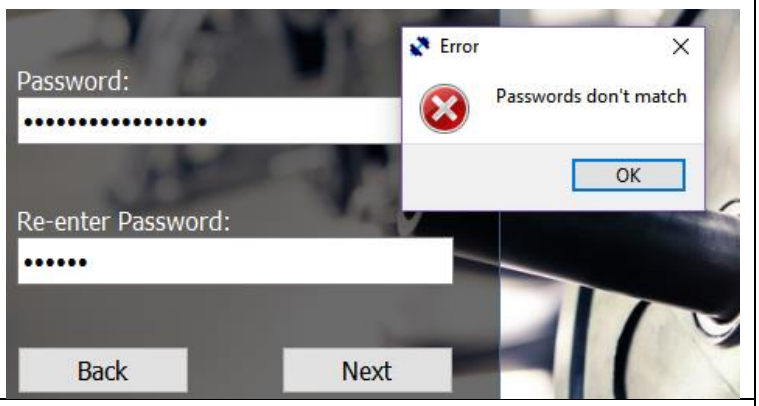
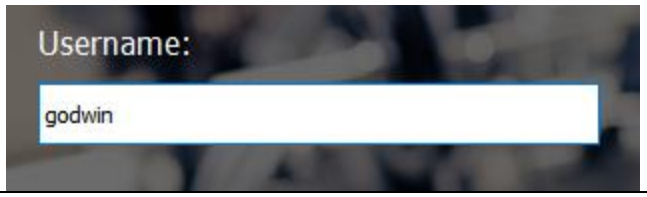
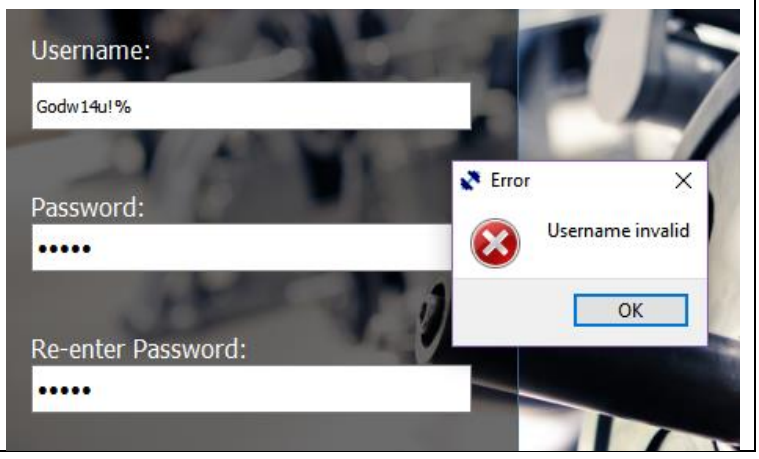


I was also able to see the stack array for my variables and variables from the imported modules:

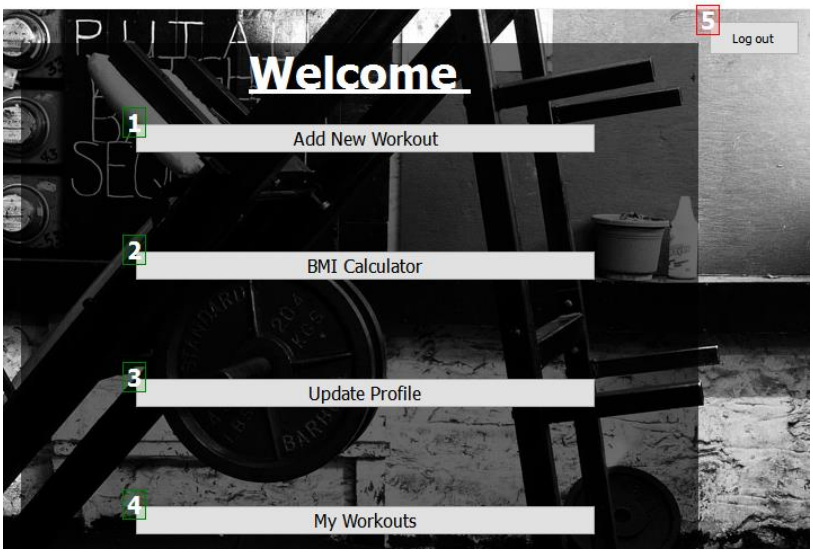
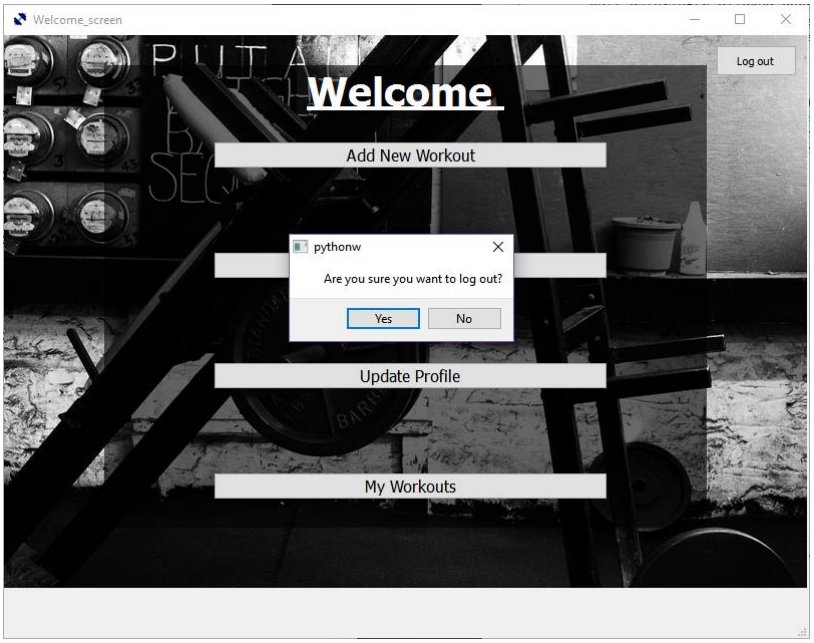


Test no.	Test data	How will it be tested	Success/Fail (action taken if failed)
11	Check the weight and height number box incremen ts when pressed (normal data)	Go into the register window 2 and increment through the numbers	Success- 
14 (1)	Check there is a label at the top of each window (normal data	Go into every screen made so far and see if there is a label at the top of the screen	Fail – Add the label of registration continued to the screen   
14 (2)	Check there is a label at the top of each window (normal data	Go into every screen made so far and see if there is a label at the top of the screen	Success- 
15	Check the font is large enough on any labels (normal data)	Go into the registration windows and ask Andrew Hersh if the labels are large enough for him	Success- 'Yes they are large enough to read' – 
16	Check all the buttons are	Go into the registration windows and check the buttons	Success - 

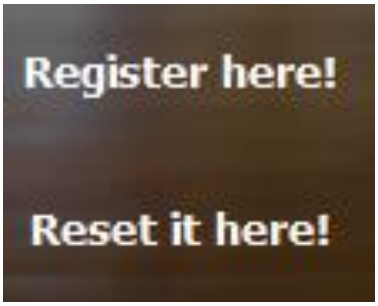
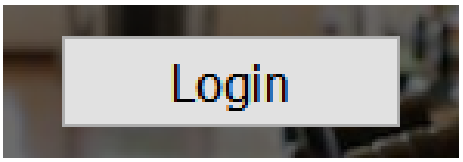
	labelled (normal data)	are labelled	
19	Register for new account (normal data)	Using the program and registration windows set up a new account	Success - 
21	Check if data has been written to the database after user has registere d	Go into the database and see if the data has been written to the database after registration	Success - 
42	Input password on register screen	Input different passwords and see if an error message comes up	Success –

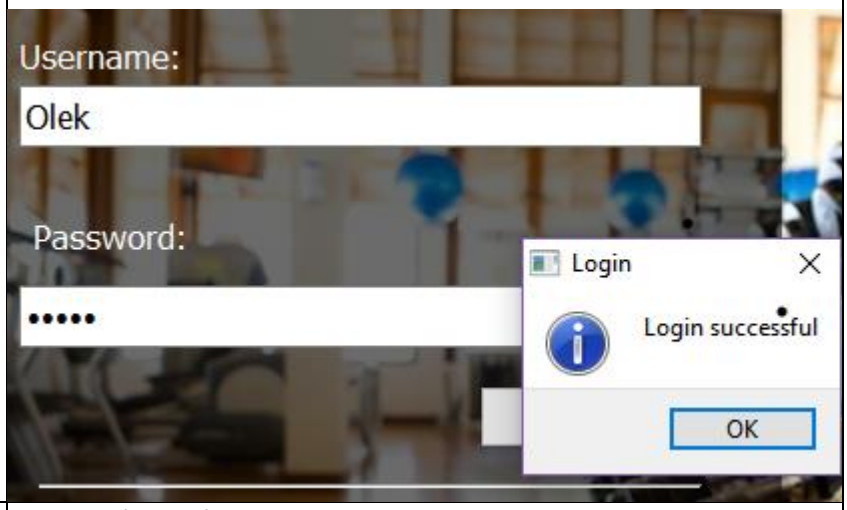
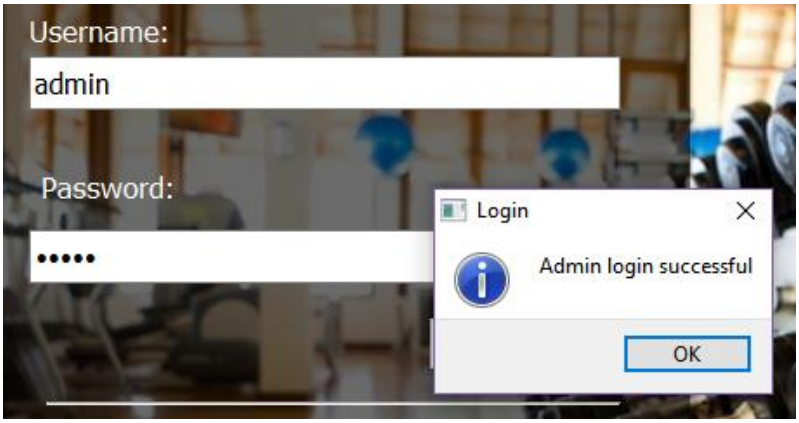
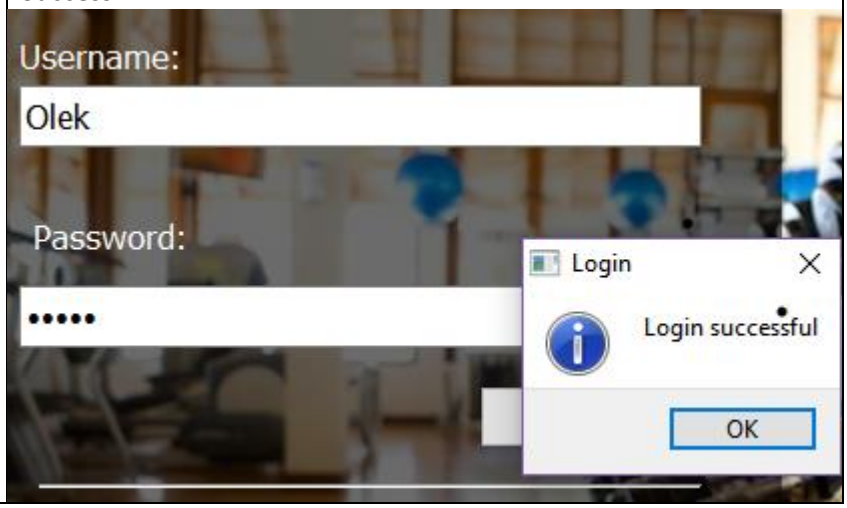
			
43	"godwin" entered into username box (normal data)	See if the program accepts this username when type into the line edit	Success – the program lets us proceed with this username 
44	"Godwin 65" entered into username box (extreme data)	See if the program accepts this username when next button is pressed	Success- the program lets us proceed 
45	"Godw14 u!%" entered into password box (erroneous data)	The program should reject this when type into the line edit and come up with an error message	Success – an error message is displayed when this is input 

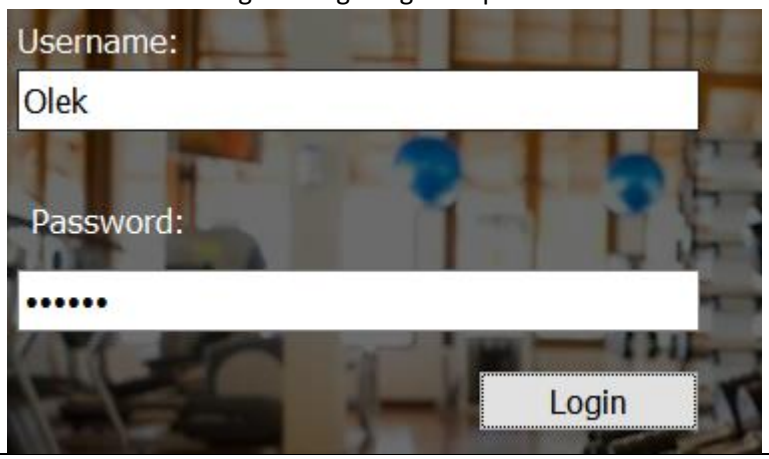
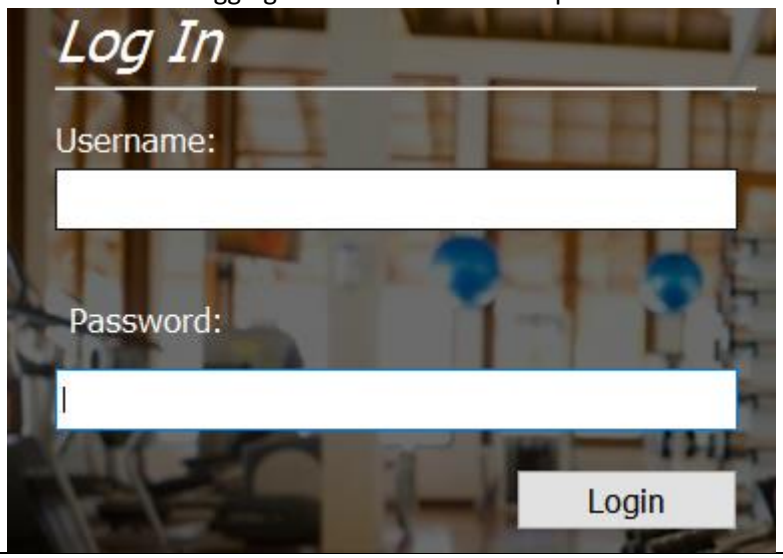
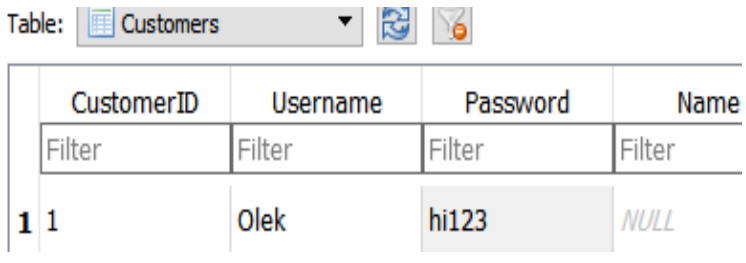
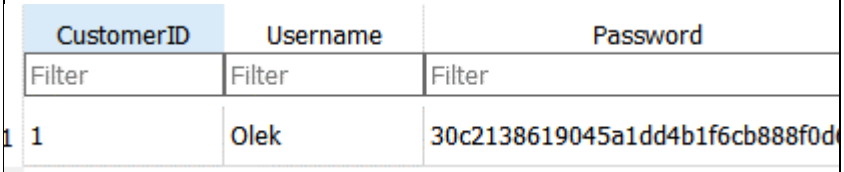
Test no.	Test data	How will it be tested	Success/Fail (action taken if failed)

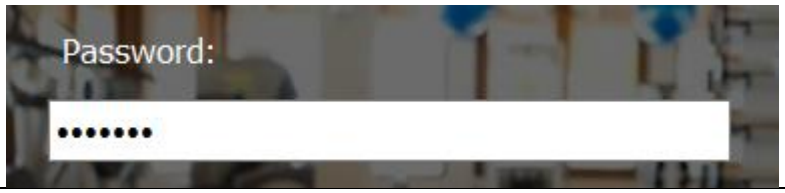
16	Check all the buttons are labelled (normal data)	Opening the welcome screen and checking if all buttons have a label	Success – 
39	Log out button pressed	Go into the welcome window and press the log out button	Success - 

Test no.	Test data	How will it be tested	Success/Fail (action taken if failed)

6	Check the text labels are in bold for registering a new account and resetting password (normal data)	Opening up the login window and checking that the labels are in bold	Success- 
16	Check all the buttons are labelled (normal data)	Open up the login window and check that all the buttons are correctly labelled	Success - 
17 (1)	Log in as admin and log in as a normal user (normal data)	Open up the login window and type in the details for a normal user – Olek, hi123. Then log in as admin – admin, admin	Fail - I asigned the data from the database to the username variable. Also, I used try, except and else to make sure empty usernames cannot be input and an error message is displayed. I knew that try, except and else would fix the empty username input error because if it did not read the username from the database, it would go to the 'except' part of the code, where the error message could be displayed Enter the usernameOlek Enter the passwordhi123 None Traceback (most recent call last): File "C:\Users\oleks\OneDrive\Documents\Coursework tables if username==username_string: #An if statement comparir NameError: name 'username' is not defined

17 (2)	Log in as admin and log in as a normal user (normal data)	Open up the login window and type in the details for a normal user – Olek, hi123. Then log in as admin – admin, admin	Success (normal user) - 
17 (2)	Log in as admin and log in as a normal user (normal data)	Open up the login window and type in the details for a normal user – Olek, hi123. Then log in as admin – admin, admin	Success (admin) - 
19 (1)	Log into the program as a user (normal data)	Open up the log in window and log in as a normal user	Success - 

20 (1)	Log into the program as a user using someone else's password (erroneous data)	Open up the log in window and use Olek and another user's password (godwin's 'league' password	Success – cannot log in using 'league as password' 
39	Check user can log out of the program	Log out button pressed	Success – after logging out the username and password are cleared 
40 (1)	Check passwords in database and see if hashed	Open up SQL database browser and check if passwords are hashed	Fail – I used the md5 hash module in python to hash this password 
40 (2)	Check passwords in database and see if hashed	Open up SQL database browser and check if passwords are hashed	Success – 
41 (1)	Input password	Check password is 'echoed	Success-

		out’	

Evidence of code

Type	Evidence

Comments for ever loop, selection or procedure	`self.loginBtn.clicked.connect(self.open_welcome_window) #If login button clicked go to open third function`
	`self.username_edit.setMaxLength(20)#Sets max length allowed in username`
	`self.password_edit.setMaxLength(20)#Sets max length allowed in password`
	`palette = QtGui.QPalette()  #Palette for the colours`
	`self.actionQuit.triggered.connect(self.quit) #Can quit using shortcut`
	`self.actionQuit.setShortcut(QtGui.QKeySequence("ESC")) #Use ESC to quit`
	`self.actionEnter.triggered.connect(self.open_welcome_window) #Once the button is triggered it goes to the function`
	`self.actionEnter.setShortcut(QtGui.QKeySequence("RETURN")) #Setting ENTER key to go to the next screen`
	`self.register_lbl.setPalette(palette) #Sets the colour`
	`self.help_lbl.mousePressEvent = self.see_help`
	`self.register_lbl.mousePressEvent = self.open_register_window`
	`self.reset_lbl.mousePressEvent = self.open_reset_window`
	
	`def open_register_window(self, event): #Hides current window and shows the register window`
	`self.hide()`
	`self.password_edit.clear()`
	`self.username_edit.clear()`
	`register_window.show()  #Opens up register page`
	`def open_welcome_window(self): #Opens the welcome window`
	`self.username_str=self.username_edit.text()`
	`self.password_str=self.password_edit.text()`
	`try: #Try and see if username and passwords match admin passwords`
	`if self.username_str==real_username: #If user input username is equal to admin username`
	`if self.password_str==real_password: #If user input password is equal to admin password`
	`self.hide()`
	`admin_window.show()`
	`self.username_edit.clear()`
	`self.password_edit.clear()`
	`else: #Else display an error`
	`QtGui.QMessageBox.critical(self, "Error", "Username or password incorrect")`
	`cur.execute("""SELECT Username, Password FROM Customers WHERE Username=?""",(self.username_str,))`
	`user_details_list=cur.fetchall()`
	`data_username_str=user_details_list[0][0]`
	`data_password_str=user_details_list[0][1]`
	`print(user_details_list)`
	`except: #If they dont match`
	`print("Error")`
	`else: #If any other input check the database data`
	
	`if self.username_str==data_username_str: #If username is equal to username from database`
	`self.password_str=hashlib.md5(self.password_str.encode('utf-8')).hexdigest()`
	`if self.password_str==data_password_str: #If password is equal to password from database - hide current window`
	`self.hide()`
	`welcome_window.show()`
	`self.username_edit.clear()`
	`self.password_edit.clear()`
	`else: #Otherwise display an error message`
	`QtGui.QMessageBox.critical(self, "Error", "Username or password wrong")`

Parameter passing	`def open_reset_window(self, event): #Hides the current window and opens the password reset wind`
	
	`def total(self): #Calculates the total for the workout`

Presence and data type check contained in resuable procedures	`QString=type("")`
	`global DURATION_str`
	`DURATION_str=str(1.5)`
	`PRICE_float=float(PAYMENT_str)*float(DURATION_str)`

Mix of functions and procdures	`if self.password_str==real_password: #If user input password is equal to admin passwo`
	`Success=True`
	`return Success`
	`self.hide()`
	`admin_window.show()`
	`self.username_edit.clear()`
	`self.password_edit.clear()`
	
	`def goback(self):`
	`self.hide() #Hides current window`
	`login_window.show() #Shows previous window`
	
	`class PasswordChangeWindow(QtGui.QMainWindow, passwordChange_win):`
	`def __init__(self, parent=None): #Sets up the UI and links the butt`
	`QtGui.QMainWindow.__init__(self, parent)`
	`self.setupUi(self)`
	`self.backBtn.clicked.connect(self.goback)`
	`self.saveChangesBtn.clicked.connect(self.changepass)`

Variables of different scopes: class, object, local, global	for each in self.details_list: #F global NAME_str global SURNAME_str global POSTCODE_str global EMAIL_str global GENDER_bool NAME_str=each[0] SURNAME_str=each[1] POSTCODE_str=each[2] EMAIL_str=each[3] global WEIGHT_float global HEIGHT_float WEIGHT_float=each[4] HEIGHT_float=each[5] GENDER_bool=each[6] global PREVIOUS_WEIGHT_float global PREVIOUS_HEIGHT_float PREVIOUS_WEIGHT_float=each[7] PREVIOUS_HEIGHT_float=each[8] global GOALID_int GOALID_int=each[9] print("current values: " ,NAME_st class UserDetailsWindow(QtGui.QMainWindow, userDetails_win): DATA_USERNAME_str=USER_DETAILS_list[0][0] DATA_PASSWORD_str=USER_DETAILS_list[0][1]
Use of constants with capitalised names	for each in self.details_list: #For each in the global NAME_str global SURNAME_str global POSTCODE_str global EMAIL_str global GENDER_bool NAME_str=each[0] SURNAME_str=each[1] POSTCODE_str=each[2] EMAIL_str=each[3] global WEIGHT_float global HEIGHT_float WEIGHT_float=each[4] HEIGHT_float=each[5] GENDER_bool=each[6] global PREVIOUS_WEIGHT_float global PREVIOUS_HEIGHT_float PREVIOUS_WEIGHT_float=each[7] PREVIOUS_HEIGHT_float=each[8] global GOALID_int GOALID_int=each[9] print("current values: " ,NAME_str, SURNAME_str,

Use of modules imported as needed	
Log in and out with hashed password data	
Two types of user accounts: admin and regular. Admin can do things to regular accounts.	

```
#Importing the modules needed
import hashlib
import random
import smtplib
from smtplib import SMTP
import sys, os
import time
import sqlite3 as lite
import re
import matplotlib.pyplot as plt
import csv
from PyQt4 import QtCore, QtGui, uic
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sqlite3
```

```
if self.username_str==data_username_str: #If username is equal to username from database
    self.password_str=hashlib.md5(self.password_str.encode('utf-8')).hexdigest()
    if self.password_str==data_password_str: #If password is equal to password from database - hide current window
        self.hide()
        welcome_window.show()
        self.username_edit.clear()
        self.password_edit.clear()
else: #Otherwise display an error message
    QtGui.QMessageBox.critical(self, "Error", "Username or password wrong")
```

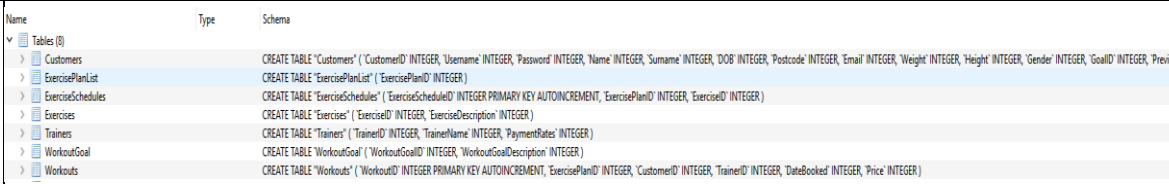
CustomerID	Username	Password
Filter	Filter	Filter
100009	Jamie	34291703b0b5937866e1a49f599d2c39

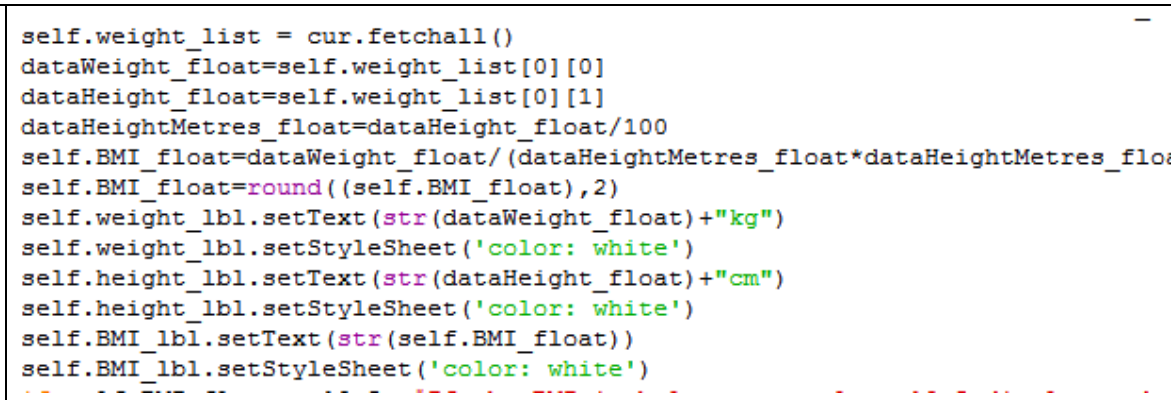
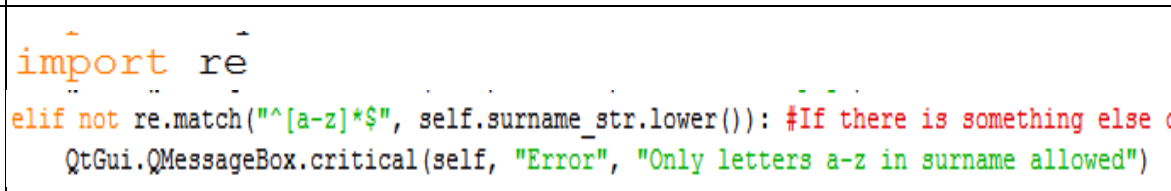
```
if self.username_str==data_username_str: #If username is equal to username from database
    self.password_str=hashlib.md5(self.password_str.encode('utf-8')).hexdigest()
    if self.password_str==data_password_str: #If password is equal to password from database - hide current wi
        self.hide()
        welcome_window.show()
        self.username_edit.clear()
        self.password_edit.clear()
else: #Otherwise display an error message
    QtGui.QMessageBox.critical(self, "Error", "Username or password wrong")

if self.username_str==real_username: #If user input username is equal to admin username
    if self.password_str==real_password: #If user input password is equal to admin passwo
        self.hide()
        admin_window.show()
        self.username_edit.clear()
        self.password_edit.clear()
    else: #Else display an error
        QtGui.QMessageBox.critical(self, "Error", "Username or password incorrect")
```

Passwords enforced/checked for strength	`if self.password_str=="": #If input password is empty`
	`  QtGui.QMessageBox.critical(self, "Error", "Password box(es) empty")`
	`elif len(self.username_str)<5: #If username is shorter than 5 in length`
	`  QtGui.QMessageBox.critical(self, "Error", "Username too short")`
	`elif len(self.password_str)<5: #If password is shorter than 5 in length`
	`  QtGui.QMessageBox.critical(self, "Error", "Password too short")`
	`elif self.password_str!=self.repassword_str: #If password and re-enter password dont ma`
	`  QtGui.QMessageBox.critical(self, "Error", "Passwords don't match")`
	`elif self.username_str=="": #If the username is empty`
	`  QtGui.QMessageBox.critical(self, "Error", "Username empty")`
	`elif self.username_str==self.password_str: #If the username is equal to the password`
	`  QtGui.QMessageBox.critical(self, "Error", "Username and password can't be the same"`
	`elif ' ' in self.username_str: #If there is a space in username`
	`  QtGui.QMessageBox.critical(self, "Error", "Username invalid")`
	`elif ' ' in self.password_str: #If there is a space in password`
	`  QtGui.QMessageBox.critical(self, "Error", "Spaces in password")`

SQL parameter queries (with question marks)	`cur.execute("INSERT INTO Customers (Username, Password, Name, Surname, Postcode, Email, Weight, Height, Gender,`
	`  (register_window.username, register_window.password, self.forename_str, self.surname_str, self.post`
	`con.commit()`
	`GoalID) VALUES ( ?, ?, ?, ?, ?, ?, ?, ?, ?, ? )",`
	`code_str, self.email_str, self.weight_float, self.height_float, self.gender_bool, self.Goal_int))`

Inner join or nested SQL queries	`s='select Workouts.WorkoutID,Workouts.DateBooked, Customers.Name from Workouts`
	`cur.execute(s)`
	
	`inner join Customers on Workouts.CustomerID=Customers.CustomerID'`

Relational database of at least 3 tables	

Hungarian notation	
Regular expressions	

All my code for further evidence (text + screenshots):

```
#tutorial reference http://pyqt.sourceforge.net/Docs/PyQt4/qwidget.html#x-prop

#Importing the modules needed
import hashlib
import random
import smtplib
from smtplib import SMTP
import sys, os
import time
import sqlite3 as lite
import re
import matplotlib.pyplot as plt
import csv
from PyQt4 import QtCore, QtGui, uic
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sqlite3

#Loading the database
con = sqlite3.connect('bodybuildingv3.db')
cur = con.cursor()

#Setting the admin login
real_username="Olek"
real_password="olek"

#Loading all the interface files
register_pg2_win = uic.loadUiType("register_pg2.ui")[0]  # Load the register page 2 UI
login_win = uic.loadUiType("login.ui")[0]  # Load the login UI
welcome_win = uic.loadUiType("welcome.ui")[0]  # Load the homepage UI
register_pg1_win = uic.loadUiType("register_pg1.ui")[0]  # Load the register page 1 UI
workouts_inner_win = uic.loadUiType("workout_inner.ui")[0] #Load the workouts inner page UI
admin_win = uic.loadUiType("admin_pg.ui")[0] #Load the admin page UI
BMI_win= uic.loadUiType("BMI.ui")[0] #Load the BMI UI
userDetails_win = uic.loadUiType("UserUpdate.ui")[0] #Load the user update UI
passwordChange_win = uic.loadUiType("passwordChange.ui")[0] #Load the password changeUI
cut_win= uic.loadUiType("cut.ui")[0] #Load the BMI UI
build_win = uic.loadUiType("build.ui")[0] #Load the build routine UI
transform_win = uic.loadUiType("transform.ui")[0] #Load the transform routine UI
finalscreen_win = uic.loadUiType("finalscreen.ui")[0] #Load the final screen UI
passwordreset1_win = uic.loadUiType("PasswordReset1.ui")[0] #Load the password reset UI
myworkouts_win = uic.loadUiType("myworkouts.ui")[0] #Load the my workouts UI

#Class - login window
class LoginWindow(QtGui.QMainWindow, login_win):
    def __init__(self, parent=None): #Loads the UI and sets it up
        QtGui.QMainWindow.__init__(self, parent)
        QString=type("")
        self.username_str=""
        self.password_str=""
        self.setupUi(self)
        self.loginBtn.clicked.connect(self.open_welcome_window) #If login button clicked go to open third function
        self.username_edit.setMaxLength(20)#Sets max length allowed in username
        self.password_edit.setMaxLength(20)#Sets max length allowed in password
        palette = QtGui.QPalette()  #Palette for the colours
        self.actionQuit.triggered.connect(self.quit) #Can quit using shortcut
        self.actionQuit.setShortcut(QtGui.QKeySequence("ESC")) #Use ESC to quit
        self.actionEnter.triggered.connect(self.open_welcome_window) #Once the button is triggered it goes to the function
        self.actionEnter.setShortcut(QtGui.QKeySequence("RETURN")) #Setting ENTER key to go to the next screen
        self.register_lbl.setPalette(palette) #Sets the colour
        self.help_lbl.mousePressEvent = self.see_help
```

```
    self.register_lbl.mousePressEvent = self.open_register_window
    self.reset_lbl.mousePressEvent = self.open_reset_window

def open_register_window(self, event): #Hides current window and shows the register window
    self.hide()
    self.password_edit.clear()
    self.username_edit.clear()
    register_window.show()  #Opens up register page
def open_welcome_window(self): #Opens the welcome window
    self.username_str=self.username_edit.text()
    self.password_str=self.password_edit.text()
    try: #Try and see if username and passwords match admin passwords
        if self.username_str==real_username: #If user input username is equal to admin username
            if self.password_str==real_password: #If user input password is equal to admin password
                self.hide()
                admin_window.show()
                self.username_edit.clear()
                self.password_edit.clear()
            else: #Else display an error
                QtGui.QMessageBox.critical(self, "Error", "Username or password incorrect")
        cur.execute("""SELECT Username, Password FROM Customers WHERE Username=?""",(self.username_str,))
        user_details_list=cur.fetchall()
        data_username_str=user_details_list[0][0]
        data_password_str=user_details_list[0][1]
        print(user_details_list)
    except: #If they dont match
        QtGui.QMessageBox.critical(self, "Error", "Login unsuccessful")
    else: #If any other input check the database data

        if self.username_str==data_username_str: #If username is equal to username from database
            self.password_str=hashlib.md5(self.password_str.encode('utf-8')).hexdigest()
            if self.password_str==data_password_str: #If password is equal to password from database - hide current window
                QtGui.QMessageBox.information(self, "Welcome", "Login successful")
                self.hide()
                welcome_window.show()
                self.username_edit.clear()
                self.password_edit.clear()
        else: #Otherwise display an error message
            QtGui.QMessageBox.critical(self, "Error", "Username or password wrong")

def see_help(self, event): #Function for the help button displaying help message
    QtGui.QMessageBox.information(self, "Help", "If you don't have an account, register for a free one")
def quit(self): #Function which asks user to input yes or no then quits or stays depending on choice
    msgBox = QtGui.QMessageBox()
    msgBox.setText('Are you sure you want to quit?')
    msgBox.addButton(QtGui.QPushButton('Yes'), QtGui.QMessageBox.YesRole)
    msgBox.addButton(QtGui.QPushButton('No'), QtGui.QMessageBox.NoRole)
    ret = msgBox.exec_()
    if ret==0: #If the answer is 'no'
        self.close()
    else: #If the answer is 'yes'
        msgBox.hide()

def open_reset_window(self, event): #Hides the current window and opens the password reset window
    self.username_edit.clear()
    self.password_edit.clear()
    self.hide()
    password_reset_window.show()
```

```
#Class - register window 1
class RegisterWindow1(QtGui.QMainWindow, register_pg1_win):
    def __init__(self, parent=None): #Sets up the UI and assign enter button for confirmation of input
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.username_reg_edit.setMaxLength(20)
        self.password_reg_edit.setMaxLength(20)
        self.password_reenter_edit.setMaxLength(20)
        self.actionEnter.triggered.connect(self.next_page)
        self.actionEnter.setShortcut(QtGui.QKeySequence("RETURN"))
        self.homeBtn.clicked.connect(self.goback)
        self.nextBtn.clicked.connect(self.next_page)
    def goback(self):
        self.hide() #Hides current window
        login_window.show() #Shows previous window
    def next_page(self): #Goes to the next page when button clicked
        self.username_str=self.username_reg_edit.text()
        self.password_str=self.password_reg_edit.text()
        self.repassword_str=self.password_reenter_edit.text()
        cur.execute("SELECT Username, Password FROM Customers WHERE Username=?",(self.username_str,))
        data_list=cur.fetchall()
        try: #Tries to print the first index of the fetched list
           print(data_list[0][0])
        except: #If it does read the list inputs checked against criteria
            if self.password_str=="": #If input password is empty
                QtGui.QMessageBox.critical(self, "Error", "Password box(es) empty")
            elif len(self.username_str)<5: #If username is shorter than 5 in length
                QtGui.QMessageBox.critical(self, "Error", "Username too short")
            elif len(self.password_str)<5: #If password is shorter than 5 in length
                QtGui.QMessageBox.critical(self, "Error", "Password too short")
            elif self.password_str!=self.repassword_str: #If password and re-enter password dont match
                QtGui.QMessageBox.critical(self, "Error", "Passwords don't match")
            elif self.username_str=="": #If the username is empty
                QtGui.QMessageBox.critical(self, "Error", "Username empty")
            elif self.username_str==self.password_str: #If the username is equal to the password
                QtGui.QMessageBox.critical(self, "Error", "Username and password can't be the same")
            elif ' ' in self.username_str: #If there is a space in username
                QtGui.QMessageBox.critical(self, "Error", "Username invalid")
            elif ' ' in self.password_str: #If there is a space in password
                QtGui.QMessageBox.critical(self, "Error", "Spaces in password")
            else: #Otherwise clear line edits and proceed to next page
                self.username_reg_edit.clear()
                self.password_reg_edit.clear()
                self.password_reenter_edit.clear()
                self.hide()
                register_window_page2.show()

        else: #Otherwise display error message because username exists
            QtGui.QMessageBox.critical(self, "Error", "That username already exists")
```

```
#Class - welcome window
class WelcomeWindow(QtGui.QMainWindow, welcome_win):
    def __init__(self, parent=None): #Sets up the UI
        QtGui.QMainWindow.__init__(self,parent)
        self.setupUi(self)
        self.myworkoutsBtn.clicked.connect(self.open_myworkouts_window)
        self.workoutsBtn.clicked.connect(self.open_workouts_window)
        self.calorieBtn.clicked.connect(self.open_BMI_window)
        self.updateProfileBtn.clicked.connect(self.open_profile_window)
        self.logoutBtn.clicked.connect(self.logout)
    def logout(self): #Logout function
        login_window.password_edit.clear()
        login_window.password_edit.clear()
        msgBox = QtGui.QMessageBox()
        msgBox.setText('Are you sure you want to log out?')
        msgBox.addButton(QtGui.QPushButton('Yes'), QtGui.QMessageBox.YesRole)
        msgBox.addButton(QtGui.QPushButton('No'), QtGui.QMessageBox.NoRole)
        ret = msgBox.exec_()
        if ret==0: #If the answer is 'yes' - log out
            self.close()
            login_window.show()
        else: #If the anwser is 'no' hide message box
            msgBox.hide()

    def open_profile_window(self): #Hides current window and opens user details window
        self.hide()
        user_details_window.show()
    def open_workouts_window(self): #Hides current window and opens workouts window
        self.hide()
        workouts_window.show()
    def open_BMI_window(self):  #Hides current window and opens bmi window
        self.hide()
        bmi_window.show()
    def open_myworkouts_window(self): #Hides current window and opens my workouts window
        self.hide()
        myworkouts_window.show()
```

```
#Class - register window page 2
class RegisterWindow2(QtGui.QMainWindow, register_pg2_win):
    def __init__(self, parent=None): #Sets up UI and link buttons to function
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        self.nextBtn.clicked.connect(self.gonext)
    def goback(self): #Hides current screen and opens register window
        self.hide()
        register_window.show()
    def gonext(self): #Checks criteria and then registers the user
        self.forename_str=self.forename_edit.text()
        self.surname_str=self.surname_edit.text()
        self.email_str=self.email_edit.text()
        register_window.password_str=hashlib.md5(register_window.password_str.encode('utf-8')).hexdigest()
        self.postcode_str=self.postcode_edit.text()
        self.weight_float=self.weight_edit.text()
        self.height_float=self.height_edit.text()
        self.gender_bool=self.gender_combo.currentText()
        self.goal_name_str=self.goal_combo.currentText()
        if self.goal_name_str=="Cut": #If the input in the combo box is 'Cut' set goal to 1
            self.goal_int=1
        elif self.goal_goal_name_str=="Build": #If the input in the combo box is 'Build' set goal to 2
            self.goal_int=2
        elif self.goal_goal_name_str()=="Transform": #If the input in the combo box is 'Transform' set goal to 3
            self.goal_int=3
        if not re.match("^[a-z]*$", self.forename_str.lower()): #If there is something else other than letters in forename display error message
            QtGui.QMessageBox.critical(self, "Error", "Only letters a-z in forename allowed")
        elif self.forename_str=="": #If forename is empty
            QtGui.QMessageBox.critical(self, "Error", "Forename empty")
        elif not re.match("^[a-z]*$", self.surname_str.lower()): #If there is something else other than letters in surname display error message
            QtGui.QMessageBox.critical(self, "Error", "Only letters a-z in surname allowed")
        elif self.surname_str=="": #If the surname is empty display error message
            QtGui.QMessageBox.critical(self, "Error", "Surname empty")
        else: #Else write all the details in to the database and hide current window + show the welcome window
            cur.execute("INSERT INTO Customers (Username, Password, Name, Surname, Postcode, Email, Weight, Height, Gender, GoalID) VALUES ( ?, ?, ?, ?, ?, ?, ?, ?, ?, ? )",
                        (register_window.username_str, register_window.password_str, self.forename_str, self.surname_str, self.postcode_str, self.email_str, self.weight_float, self.height_float, self.gender_bool, self.goal_int))
            con.commit()
            self.close()
            welcome_window.show()
```

```
#Class - workouts window
class WorkoutsWindow(QtGui.QMainWindow, workouts_inner_win):
    def __init__(self, parent=None): #Sets up UI and link buttons to function
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        trainer_name_list=[]
        self.backBtn.clicked.connect(self.goback)
        cur.execute("select * from Trainers")
        trainers_data_list=cur.fetchall()
        for each in trainers_data_list: #For each piece of data in the list add it to a seperate list
            trainer_name_list.append(each[1])
        self.trainer_name_combo.addItems(trainer_name_list)
        self.calendar.clicked[QtCore.QDate].connect(self.showDate)
        global selected_date
        selected_date = self.calendar.selectedDate()
        self.selected_date_lbl.setText(selected_date.toString())
        palette = QtGui.QPalette()
        palette.setColor(QtGui.QPalette.Foreground,QtCore.Qt.white) #Change colour here
        self.selected_date_lbl.setPalette(palette) #Sets the colour
        current_time_str=time.strftime("%d/%m/%Y")
        self.calendar.setMinimumDate(QDate(selected_date))
        self.workoutBtn.clicked.connect(self.goalread)
        self.proceedBtn.clicked.connect(self.proceed)
    def proceed(self): #When the procced button is clicked run this function
        self.calendar.clicked[QtCore.QDate].connect(self.showDate)
        selected_date = self.calendar.selectedDate()
        global new_date
        new_date = selected_date.toString("dd/MM/yyyy")
        global trainer_name
        trainer_name=self.trainer_name_combo.currentText()
        self.hide()
        booking_window.show()
    def goalread(self): #Once the workout button is clicked select goal id from customer
        cur.execute("select GoalID FROM Customers WHERE Username=?""",(login_window.username_str,))
        goal_list=cur.fetchone()
        for each in goal_list: #For each item in the goal list set goal int to each
            Goal_int=each
        if Goal_int==1: #If the goal is 1 then show the cut routines window
            self.hide()
            cut_window.show()
        elif Goal_int==2: #If the goal is 2 then show the build routines window
            self.hide()
            build_window.show()
        elif Goal_int==3: #If the goal is 3 then show the transform routines window
            self.hide()
            transform_window.show()

    def showDate(self, date): #Set the label to the date selected
        self.selected_date_lbl.setText(selected_date.toString())

    def goback(self): #Hides current window and goes to welcome window
        self.hide()
        welcome_window.show()
```

```
#Class - admin window
class AdminWindow(QtGui.QMainWindow, admin_win):
    def __init__(self, parent=None): #Sets up UI and link buttons to function
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.data=[]
        self.model = QtGui.QStandardItemModel(self)
        self.load_data()
        self.tableView.setModel(self.model)
        self.homeBtn.clicked.connect(self.gohome)
        self.updateBtn.clicked.connect(self.update)
        self.graphBtn.clicked.connect(self.graph)
        self.allworkoutsBtn.clicked.connect(self.allworkouts)
        self.allusersBtn.clicked.connect(self.load_data)
    def gohome(self): #If home button is pressed hide current window and show login window
        self.hide()
        login_window.show()
    def load_data(self): #Loads the data using the customers table data and assigns it to the table view
            s='select * from Customers'
            cur.execute(s)
            self.data_list = cur.fetchall() #query data comes back
            if len(self.data_list)>0: #If the length of the list is bigger than 0
                for i in range(len(self.data_list)-1,-1): #Whilst i is in the range of the length of the data list remove row
                    print("removing row",i)
                    self.data_list.pop(i)
                    self.model.removeRow(index.row(self.data_list[i]))
            self.model=QtGui.QStandardItemModel(self)
            self.tableView.setModel(self.model)
            for row in self.data_list:    #For each row in the self data list
                        items = [
                            QtGui.QStandardItem(str(field))
                            for field in row #For each field in row set the name for the column
                        ]
                        self.model.appendRow(items)
            self.model.setHeaderData(0, QtCore.Qt.Horizontal, "CustomerID")
            self.model.setHeaderData(1, QtCore.Qt.Horizontal, "Username")
            self.model.setHeaderData(2, QtCore.Qt.Horizontal, "Password")
            self.model.setHeaderData(3, QtCore.Qt.Horizontal, "Name")
            self.model.setHeaderData(4, QtCore.Qt.Horizontal, "Surname")
            self.model.setHeaderData(5, QtCore.Qt.Horizontal, "DOB")
            self.model.setHeaderData(6, QtCore.Qt.Horizontal, "Postcode")
            self.model.setHeaderData(7, QtCore.Qt.Horizontal, "Email")
            self.model.setHeaderData(8, QtCore.Qt.Horizontal, "Weight")
            self.model.setHeaderData(9, QtCore.Qt.Horizontal, "Height")
            self.model.setHeaderData(10, QtCore.Qt.Horizontal, "Gender")
            self.model.setHeaderData(11, QtCore.Qt.Horizontal, "Goal")
            self.model.setHeaderData(12, QtCore.Qt.Horizontal, "PreviousWeight")
            self.model.setHeaderData(13, QtCore.Qt.Horizontal, "PreviousHeight")

    def show_selected(self): #Shows the currently selected data
        index = self.tableView.currentIndex() #returns currently selected cell
        row_index=index.row() #returns the row of the currently selected cell
        column_index=index.column()#returns the column of the currently selected cell
        cell_contents=index.data()#returns the contents of the currently selected cell
    def update(self): #Updates the database with the changed data in the table view
            index = self.tableView.currentIndex() #returns currently selected cell
            row_index=index.row() #returns the row of the currently selected cell
            colum_index=index.column()#returns the column of the currently selected cell
            cell_contents=index.data()#returns the contents of the currently selected cell
            id=self.model.data(self.model.index(index.row(), 0))
```

```
        index_int1=self.model.data(self.model.index(index.row(), 1))
        index_int2=self.model.data(self.model.index(index.row(), 2))
        index_int3=self.model.data(self.model.index(index.row(), 3))
        index_int4=self.model.data(self.model.index(index.row(), 4))
        index_int5=self.model.data(self.model.index(index.row(), 5))
        index_int6=self.model.data(self.model.index(index.row(), 6))
        index_int7=self.model.data(self.model.index(index.row(), 7))
        index_int8=self.model.data(self.model.index(index.row(), 8))
        index_int9=self.model.data(self.model.index(index.row(), 9))
        index_int10=self.model.data(self.model.index(index.row(), 10))
        index_int11=self.model.data(self.model.index(index.row(), 11))
        self.model.data(self.model.index(4,1))
        #parameter query - get criteria from Python
        cur.execute("""UPDATE Customers SET Username='%s',Password='%s',Name='%s',Surname='%s',DOB='%s',Postcode='%s',Email='%s',Weight='%s',Height='%s',Gender='%s',GoalID='%s'
                    WHERE CustomerID='%s'"""%(index_int1,index_int2,index_int3,index_int4,index_int5,index_int6,index_int7,index_int8,index_int9,index_int10,index_int11,id)) #not case sensitive inside the string
        msgBox = QtGui.QMessageBox()
        msgBox.setText('Are you sure you want to save changes?')
        msgBox.addButton(QtGui.QPushButton('Yes'), QtGui.QMessageBox.YesRole)
        msgBox.addButton(QtGui.QPushButton('No'), QtGui.QMessageBox.NoRole)
        ret = msgBox.exec_()
        if ret==0: #If answer is 'yes' save the changes
            con.commit()
            QtGui.QMessageBox.information(self, "Success", "Changes saved")
        else: #If anwser is 'no' hide the message box
            msgBox.hide()
def graph(self): #Plots the graph dependent on number of male and female users in the database
        cur.execute('select distinct Gender, count(Gender) as CountOf from Customers group by Gender')
        global gender_list
        gender_list=cur.fetchall()
        print(gender_list)
        # Data to plot
        try: #Try the total of the both genders count
            total_int=(gender_list[0][1])+(gender_list[1][1])
            print(total_int)
        except: #If doesn't work total is equal to just one of those counts
            total_int=gender_list[0][1]
        else: #Else print error
            print("error")
        male_per=(gender_list[1][1]/total_int)*360
        female_per=(gender_list[0][1]/total_int)*360
        labels_str = gender_list[0][0], gender_list[1][0]
        sizes_list = [female_per, male_per]
        colors_list = ['pink', 'blue']
        explode_values = (0.1, 0)  # explode 1st slice

        # Plot
        plt.pie(sizes_list, explode=explode_values, labels=labels_str, colors=colors_list,
                autopct='%1.1f%%', shadow=True, startangle=140)

        plt.axis('equal')
        plt.show()


def allworkouts(self): #Displays all the workouts using an inner join function
        s='select Workouts.WorkoutID,Workouts.DateBooked, Customers.Name from Workouts inner join Customers on Workouts.CustomerID=Customers.CustomerID'
        cur.execute(s)
        self.distinct_list = cur.fetchall() #query data comes back
        if len(self.distinct_list)>0:#If the length of the list is bigger than 0
            for i in range(len(self.distinct_list)-1,-1):#Whilst i is in the range of the length of the data list remove row
                print("removing row",i)
```

```
                self.distinct_list.pop(i)
                self.model.removeRow(index.row(self.distinct_list[i]))
        self.model=QtGui.QStandardItemModel(self)
        self.tableView.setModel(self.model)
        for row in self.distinct_list: #For each row in the distinct list create an items list
                items = [
                    QtGui.QStandardItem(str(field))
                    for field in row #For each field in row set the name for the column
                ]
                self.model.appendRow(items)
        self.model.setHeaderData(0, QtCore.Qt.Horizontal, "WorkoutID")
        self.model.setHeaderData(1, QtCore.Qt.Horizontal, "DateBooked")
        self.model.setHeaderData(2, QtCore.Qt.Horizontal, "Name")

#Class - BMI window
class BMIWindow(QtGui.QMainWindow, BMI_win):
    def __init__(self, parent=None): #Sets up the UI and connects all the buttons to the functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.data=[]
        self.model = QtGui.QStandardItemModel(self)
        self.tableView.setModel(self.model)
        welcome_window.calorieBtn.clicked.connect(self.load_data)
        self.calculateBtn.clicked.connect(self.calculate)
        self.backBtn.clicked.connect(self.goback)
        self.graphBtn.clicked.connect(self.graph)
    def goback(self): #Resets the labels, hides the current window and goes back to the welcome window
        self.hide()
        self.height_lbl.setText("")
        self.weight_lbl.setText("")
        self.BMI_lbl.setText("")
        self.result_lbl.setText("")
        welcome_window.show()
    def calculate(self): #Calculates the BMI for the user and displays according to what the outcome is
        cur.execute('SELECT Weight, Height FROM Customers WHERE Username=?',(login_window.username_str,))
        self.weight_list = cur.fetchall()
        dataWeight_float=self.weight_list[0][0]
        dataHeight_float=self.weight_list[0][1]
        dataHeightMetres_float=dataHeight_float/100
        self.BMI_float=dataWeight_float/(dataHeightMetres_float*dataHeightMetres_float)
```

```
        self.BMI_float=round((self.BMI_float),2)
        self.weight_lbl.setText(str(dataWeight_float)+"kg")
        self.weight_lbl.setStyleSheet('color: white')
        self.height_lbl.setText(str(dataHeight_float)+"cm")
        self.height_lbl.setStyleSheet('color: white')
        self.BMI_lbl.setText(str(self.BMI_float))
        self.BMI_lbl.setStyleSheet('color: white')
        if self.BMI_float <=19.5: #If the BMI is below or equal to 19.5 display underweight
            self.result_lbl.setText("Underweight")
            self.result_lbl.setStyleSheet('color: white')
        elif self.BMI_float >19.5 and self.BMI_float<25:#If the BMI is larger than 19.5 and smaller than 25 display healthy weight
            self.result_lbl.setText("Healthy Weight")
            self.result_lbl.setStyleSheet('color: white')
        elif self.BMI_float >25 and self.BMI_float <30: #If the BMI is larger than 25 and smaller than 30 display overweight
            self.result_lbl.setText("Overweight")
            self.result_lbl.setStyleSheet('color: white')
        elif self.BMI_float >30 and self.BMI_float <35: #If the BMI is larger than 30 and smaller than 35 display obese
            self.result_lbl.setText("Obese")
            self.result_lbl.setStyleSheet('color: white')
        elif self.BMI_float >35 and self.BMI_float <40: #If the BMI is larger than 35 and smaller than 40 display severly obese
            self.result_lbl.setText("Severely Obese")
            self.result_lbl.setStyleSheet('color: white')
        elif self.BMI_float >40: #If the BMI is larger than 40 display morbidly obese
            self.result_lbl.setText("Morbidly Obese")
            self.result_lbl.setStyleSheet('color: white')
    def load_data(self): #Load the data and select previous weight and weight from customers table
        cur.execute('SELECT PreviousWeight, Weight from Customers WHERE Username=?',(login_window.username_str,))
        self.new_weight_list = cur.fetchall()
        global previousweight
        global weight
        self.previousweight_int=self.new_weight_list[0][0]
        self.weight_int=self.new_weight_list[0][1]


    def graph(self):
            template='''<!DOCTYPE html>
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
    </style>
  </head>
  <body>
    <button onclick="test();">Launch</button>
    <p id="out"> </p>
    <canvas id="myCanvas" width="150" height="425"></canvas>

    <script>
       var canvas = document.getElementById('myCanvas');
      var context = canvas.getContext('2d');
      var a=new Array(5, 8);
function test(){

      var arrayLength = a.length;
for (var i = 0; i < arrayLength; i++) {
```

```
    //Do something
  draw_bar(50,i);
  //document.getElementById("out").innerHTML+=","+a[i];
}

      }
      function draw_bar(size,ind){

      context.beginPath();

      context.rect(size*ind, canvas.height-size/16*a[ind], size,size/8*a[ind]);
      context.fillStyle = 'yellow';
      context.fill();
      context.lineWidth = 4;
      context.strokeStyle = 'black';
      context.stroke();

         context.font = "20px Arial";
        context.fillStyle = "red";
        context.textAlign = "centre";
context.fillText(a[ind],size*ind+size/2-10,canvas.height-size/16*a[ind]-10);
      }
    </script>
</body>
</html> '''
                b=(str(self.previousweight_int)+", "+str(self.weight_int))
                print(b)
                url=template.replace("5, 8", b)
                self.webView.setHtml(url)

#Class - User Details Window
class  UserDetailsWindow(QtGui.QMainWindow, userDetails_win):
    def __init__(self, parent=None): #Sets up the UI and links buttons to functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        welcome_window.updateProfileBtn.clicked.connect(self.both)
        self.updateBtn.clicked.connect(self.updateUser)
        self.change_passBtn.clicked.connect(self.change_pass)
    def goback(self): #Hides the current window and displays the welcome window
        self.hide()
        welcome_window.show()


    def loadDetails(self): #Loads the details of the currently logged in user
        cur.execute('SELECT Name, Surname, Postcode, Email, Weight, Height, Gender, PreviousWeight, PreviousHeight, GoalID FROM Customers WHERE Username=?',(login_window.username_str,))
        self.details_list = cur.fetchall()
        for each in self.details_list: #For each in the list of the users details assign each of the items to a variable
            global NAME_str
            global SURNAME_str
            global POSTCODE_str
            global EMAIL_str
            global GENDER_bool
            NAME_str=each[0]
            SURNAME_str=each[1]
            POSTCODE_str=each[2]
            EMAIL_str=each[3]
            global WEIGHT_float
            global HEIGHT_float
```

```
        WEIGHT_float=each[4]
        HEIGHT_float=each[5]
        GENDER_bool=each[6]
        global PREVIOUS_WEIGHT_float
        global PREVIOUS_HEIGHT_float
        PREVIOUS_WEIGHT_float=each[7]
        PREVIOUS_HEIGHT_float=each[8]
        global GOALID_int
        GOALID_int=each[9]
    print("current values: " ,NAME_str, SURNAME_str, POSTCODE_str, EMAIL_str , WEIGHT_float, HEIGHT_float, GENDER_bool, PREVIOUS_WEIGHT_float, PREVIOUS_HEIGHT_float, GOALID_int)
def insertvalues(self): #Insert these values from the textboxes as updated and into the database
    if GOALID_int==1: #If the goal id is 1 set the goal string to 'cut'
        GOAL_str="Cut"
    elif GOALID_int==2: #If the goal id is 2 set the goal string to 'build'
        GOAL_str="Build"
    elif GOALID_int==3: #If the goal id is 3 set the goal string to 'transform'
        GOAL_str="Transform"
    self.forename_edit.setText(NAME_str)
    self.surname_edit.setText(SURNAME_str)
    self.goal_combo.clear()
    GOAL_list=["Build","Transform","Cut"]
    self.goal_combo.addItems(GOAL_list)
    find_index_int=self.goal_combo.findText(GOAL_str)
    self.goal_combo.setCurrentIndex(find_index_int)
    self.postcode_edit.setText(POSTCODE_str)
    self.email_edit.setText(EMAIL_str)
    self.weightEdit_2.setValue(WEIGHT_float)
    self.heightEdit_2.setValue(HEIGHT_float)
def both(self): #Loads both of the loadDetails() and insertvalues() functions
    self.loadDetails()
    self.insertvalues()
def updateUser(self): #Function where the SQL query is run to update the user details
    self.updatedForename_str=self.forename_edit.text()
    self.updatedSurname_str=self.surname_edit.text()
    self.updatedPostcode_str=self.postcode_edit.text()
    self.updatedEmail_str=self.email_edit.text()
    self.updatedGoal_str=self.goal_combo.currentText()
    if self.updatedGoal_str=="Cut": #If the updated goal is cut then set the id to 1
        self.updatedGoal_str=1
    elif self.updatedGoal_str=="Build": #If the updated goal is build then set the id to 2
        self.updatedGoal_str=2
    elif self.updatedGoal_str=="Transform": #If the updated goal is transform then set the id to 3
        self.updatedGoal_str=3
    self.updatedGender_bool=self.gender_combo.currentText()
    self.updatedHeight_float=self.heightEdit_2.text()
    self.updatedWeight_float=self.weightEdit_2.text()
    self.updatedWeight_float=int(float(self.updatedWeight_float))
    if self.updatedWeight_float == WEIGHT_float: #If the updated weight is equal to the current weight then update all the details and previous weight with previous weight
        cur.execute("""UPDATE Customers SET Name='%s',Surname='%s',Postcode='%s',Email='%s',Weight='%s',Height='%s',Gender='%s', GoalID='%s', PreviousWeight='%s' WHERE Username='%s'"""%
        (self.updatedForename_str,self.updatedSurname_str,self.updatedPostcode_str,self.updatedEmail_str,self.updatedWeight_float,self.updatedHeight_float,self.updatedGender_bool,self.updatedGoal_str, PREVIOUS_WEIGHT_float,login_wi
        con.commit()
        self.loadDetails()
    else: #Else update the previous weight with the current weight
        cur.execute("""UPDATE Customers SET Name='%s',Surname='%s',Postcode='%s',Email='%s',Weight='%s',Height='%s',Gender='%s', GoalID='%s',PreviousWeight='%s' WHERE Username='%s'"""%
        (self.updatedForename_str,self.updatedSurname_str,self.updatedPostcode_str,self.updatedEmail_str,self.updatedWeight_float,self.updatedHeight_float,self.updatedGender_bool,self.updatedGoal_str, WEIGHT_float,login_window.user
        con.commit()
        self.loadDetails()
```

```
    def change_pass(self): #Hides the current window and shows the password change window
        self.hide()
        password_change_window.show()


#Class - Password Change Window
class  PasswordChangeWindow(QtGui.QMainWindow, passwordChange_win):
    def __init__(self, parent=None): #Sets up the UI and links the buttons to the functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        self.saveChangesBtn.clicked.connect(self.changepass)
    def goback(self): #Hides current window and shows the user details window
        self.hide()
        user_details_window.show()
    def changepass(self): #Reads the password from the database for the currently logged in user and updates their password with their new input
        cur.execute('SELECT Password FROM Customers WHERE Username=?',(login_window.username_str,))
        self.password_list=cur.fetchall()
        self.current_password_str=self.password_edit.text()
        hashed_password_str=self.password_list[0][0]
        self.current_password2_str=hashlib.md5(self.current_password_str.encode('utf-8')).hexdigest()
        if hashed_password_str==self.current_password2_str: #If the password from the db is equal to the hash of the current password input update the user's password
            self.new_password_str=self.newpassword_edit.text()
            newest_password_str=hashlib.md5(self.new_password_str.encode('utf-8')).hexdigest()
            QtGui.QMessageBox.information(self, "Success", "Password updated")
            cur.execute("""UPDATE Customers SET Password='%s' WHERE Username='%s'"""%(str(newest_password_str), login_window.username_str))
            con.commit()

        else: #Else display an error saying the current password is wrong
            QtGui.QMessageBox.critical(self, "Error", "Current password incorrect")


#Class - cut routines window
class CutWindow(QtGui.QMainWindow, cut_win):
    def __init__(self, parent=None): #Sets up the UI and links the buttons to the functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        self.plan1Btn.clicked.connect(self.plan1)
        self.plan2Btn.clicked.connect(self.plan2)
        self.plan3Btn.clicked.connect(self.plan3)
    def plan1(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1001
        global duration_str
        duration_str=str(1.5)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="1" OR ExerciseID="3" OR ExerciseID="18" OR ExerciseID="9" OR ExerciseID="15"')
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()

    def plan2(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1002
        global duration_str
        duration_str=str(1)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="12" OR ExerciseID="8" OR ExerciseID="10" OR ExerciseID="19" OR ExerciseID="2"')
```

```
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()
    def plan3(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1003
        global duration_str
        duration_str=str(2)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="11" OR ExerciseID="5" OR ExerciseID="7" OR ExerciseID="14" OR ExerciseID="6"')
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()
    def goback(self): #Hides current window and shows the workouts window
        self.hide()
        workouts_window.show()


#Class - build routines window
class BuildWindow(QtGui.QMainWindow, build_win):
    def __init__(self, parent=None): #Sets up the UI and links the buttons to the functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        self.plan1Btn.clicked.connect(self.plan1)
        self.plan2Btn.clicked.connect(self.plan2)
        self.plan3Btn.clicked.connect(self.plan3)
    def plan1(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1004
        global duration_str
        duration_str=str(2)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="1" OR ExerciseID="7" OR ExerciseID="15" OR ExerciseID="8" OR ExerciseID="10"')
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()

    def plan2(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1005
        global duration_str
        duration_str=str(2)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="2" OR ExerciseID="3" OR ExerciseID="12" OR ExerciseID="16" OR ExerciseID="20"')
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()
    def plan3(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1006
        global duration_str
        duration_str=str(2)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="1" OR ExerciseID="13" OR ExerciseID="10" OR ExerciseID="18" OR ExerciseID="15"')
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()
    def goback(self):
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()
```

```
#Class - transform routines window
class TransformWindow(QtGui.QMainWindow, transform_win):
    def __init__(self, parent=None): #Sets up the UI and links the buttons to the functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        self.plan1Btn.clicked.connect(self.plan1)
        self.plan2Btn.clicked.connect(self.plan2)
        self.plan3Btn.clicked.connect(self.plan3)
    def plan1(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1007
        global duration_str
        duration_str=str(1.5)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="1" OR ExerciseID="8" OR ExerciseID="10" OR ExerciseID="15" OR ExerciseID="5"')
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()

    def plan2(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1008
        global duration_str
        duration_str=str(2.5)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="2" OR ExerciseID="8" OR ExerciseID="10" OR ExerciseID="12" OR ExerciseID="19"')
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()
    def plan3(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1009
        global duration_str
        duration_str=str(2.5)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="5" OR ExerciseID="6" OR ExerciseID="7" OR ExerciseID="11" OR ExerciseID="14"')
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()
    def goback(self):
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()
#Class - booking window
class BookingWindow(QtGui.QMainWindow, finalscreen_win):
    def __init__(self, parent=None): #Sets up the UI and links the buttons to the functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        workouts_window.proceedBtn.clicked.connect(self.total)
        self.bookBtn.clicked.connect(self.gobook)

    def total(self): #Calculates the total for the workout
        cur.execute('SELECT PaymentRates FROM Trainers WHERE TrainerName=?',(trainer_name,))
        payment_str=cur.fetchone()
        for each in payment_str: #For each in the payment list make it equal to the number
            payment_str=each
        cur.execute('SELECT TrainerID FROM Trainers WHERE TrainerName=?',(trainer_name,))
        trainer_id_int=cur.fetchone()
        global trainer_actual_int
        trainer_actual_int=trainer_id_int[0]
        print(trainer_actual_int)
        global price_float
        price_float=float(payment_str)*float(duration_str)
```

```
        price_float=str(price_float)
        price_float='£'+ price_float +'0'
        self.total_lbl.setText(price_float)
        self.total_lbl.setStyleSheet('color: white')
        self.total_lbl.resize(76, 78)
    def goback(self):
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()
    def gobook(self): #Confirms the workout by writing it to the database
        cur.execute("SELECT WorkoutID FROM Workouts ORDER BY WorkoutID DESC LIMIT 1")
        workout_id_int=cur.fetchone()
        new_workout_id_int=workout_id_int[0]+1
        cur.execute("SELECT CustomerID FROM Customers WHERE Username=?",(login_window.username_str,))
        user_id_int=cur.fetchone()
        actual_user_id_int=user_id_int[0]
        cur.execute("INSERT INTO Workouts (WorkoutID, ExercisePlanID, CustomerID, TrainerID, DateBooked, Price) VALUES ( ?, ?, ?, ?, ?, ? )",
                    (new_workout_id_int, planID_int, actual_user_id_int, trainer_actual_int, new_date, price_float))
        con.commit()
        QtGui.QMessageBox.information(self, "Success", "Workout successfully booked!")
        self.hide()
        welcome_window.show()

#Class - password reset window
class PasswordResetWindow(QtGui.QMainWindow, passwordreset1_win):
    def __init__(self, parent=None): #Sets up the UI and links the buttons to the functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.nextBtn.clicked.connect(self.gonext)
        self.backBtn.clicked.connect(self.goBack)
    def goBack(self): #Hides the current window and shows the login screen
        self.hide()
        login_window.show()
    def gonext(self): #Generates a random string to be used as the 'reset password'
        self.username1_str=self.username.text()
        self.email1_str=self.email.text()
        cur.execute('SELECT Username, Email FROM Customers WHERE Username=?',(self.username1_str,))
        details_list=cur.fetchall()
        self.email2_str=details_list[0][1]
        if self.email1_str==self.email2_str: #If the email from the db is equal to the email input by user generate random string
            string=""
            x=("ABCDEFGHIJKLMNOPQRSTUVWXYZ123456789")
            counter=0
            while counter<10: #Whilst the counter is less than 10 keep adding the random choice to the string
                k=random.choice(x)
                string=k+string
                x = x.replace(k, "")
                counter=counter+1
            hashed_string_str=hashlib.md5(string.encode('utf-8')).hexdigest()
            server = smtplib.SMTP('smtp.gmail.com', 587)
            server.starttls()
            server.login("olekdonkey@gmail.com", "olek2511")
            msg = 'Subject: {}\n\n{}'.format("Password Reset", "Login with this new password"+"\n"+"New Password: "+string)
            try: #Try sending an email with the reset password
                server.sendmail("olekdonkey@gmail.com", self.email2_str, msg)
                print("email sent, with code: ", string)
                server.quit()
                cur.execute("UPDATE Customers SET Password='%s' WHERE Username='%s'"%(hashed_string_str, self.username1_str))
                con.commit()
                self.hide()
                login_window.show()
```

```
                login_window.show()
            except: #Error if email cannnot be found
                print("email not found")
        else: #Error if email cannnot be found
            print("email not found")
#Class - my workouts window
class MyWorkoutsWindow(QtGui.QMainWindow, myworkouts_win):
    def __init__(self, parent=None): #Sets up the UI and links the buttons to the functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        self.data=[]
        self.model = QtGui.QStandardItemModel(self)
        welcome_window.myworkoutsBtn.clicked.connect(self.load_data)
        self.tableView.setModel(self.model)
    def load_data(self): #Loads the customer ID data
        cur.execute("""Select CustomerID from Customers WHERE Username=?""",(login_window.username_str,))
        cust_id_list=cur.fetchone()
        cust_id1_int=cust_id_list[0]
        cur.execute('select * from Workouts WHERE CustomerID=?',(cust_id1_int,))
        self.data_list = cur.fetchall() #query data comes back
        #as a python 2D tuple  (read-only list)
        #clear all previus data, so that our model doesn't get longer and longer duplicating the records
        if len(self.data_list)>0: #If the length of the list is bigger than 1
            for i in range(len(self.data_list)-1,-1): #While the range of the length of the list keep going through the list
                self.data_list.pop(i)
                self.model.removeRow(index.row(self.data_list[i]))
        self.model=QtGui.QStandardItemModel(self)
        self.tableView.setModel(self.model)
        for row in self.data_list:    #For each row in the data create a items list
                    items = [
                        QtGui.QStandardItem(str(field))
                        for field in row #For each field in row set a name for the columm
                    ]
                    self.model.appendRow(items)
        self.model.setHeaderData(0, QtCore.Qt.Horizontal, "WorkoutID")
        self.model.setHeaderData(1, QtCore.Qt.Horizontal, "ExercisePlanID")
        self.model.setHeaderData(2, QtCore.Qt.Horizontal, "CustomerID")
        self.model.setHeaderData(3, QtCore.Qt.Horizontal, "TrainerID")
        self.model.setHeaderData(4, QtCore.Qt.Horizontal, "DateBooked")
        self.model.setHeaderData(5, QtCore.Qt.Horizontal, "Price")

    def show_selected(self):
        index = self.tableView.currentIndex() #returns currently selected cell
        r=index.row() #returns the row of the currently selected cell
        c=index.column()#returns the column of the currently selected cell
        cell_contents=index.data()#returns the contents of the currently selected cell

    def goback(self): #Hides the current window and shows the welcome window
        self.hide()
        welcome_window.show()
```

```
#Linking each window
app = QtGui.QApplication(sys.argv)
#Linking the windows the to classes
login_window = LoginWindow(None)
register_window = RegisterWindow1(None)
welcome_window = WelcomeWindow(None)
register_window_page2 = RegisterWindow2(None)
workouts_window = WorkoutsWindow(None)
admin_window = AdminWindow(None)
bmi_window = BMIWindow(None)
user_details_window = UserDetailsWindow(None)
password_change_window = PasswordChangeWindow(None)
cut_window = CutWindow(None)
build_window = BuildWindow(None)
transform_window = TransformWindow(None)
booking_window = BookingWindow(None)
password_reset_window = PasswordResetWindow(None)
myworkouts_window = MyWorkoutsWindow(None)
##starting the application
login_window.show()
app.exec_()
```

```
#tutorial reference http://pyqt.sourceforge.net/Docs/PyQt4/qwidget.html#x-prop

#Importing the modules needed
import hashlib
import random
import smtplib
from smtplib import SMTP
import sys, os
import time
import sqlite3 as lite
import re
import matplotlib.pyplot as plt
import csv
from PyQt4 import QtCore, QtGui, uic
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sqlite3

#Loading the database
con = sqlite3.connect('bodybuildingv3.db')
cur = con.cursor()

#Setting the admin login
real_username="Olek"
real_password="olek"

#Loading all the interface files
register_pg2_win = uic.loadUiType("register_pg2.ui")[0]  # Load the register page 2 UI
login_win = uic.loadUiType("login.ui")[0]  # Load the login UI
welcome_win = uic.loadUiType("welcome.ui")[0]  # Load the homepage UI
register_pg1_win = uic.loadUiType("register_pg1.ui")[0]  # Load the register page 1 UI
```

```
workouts_inner_win = uic.loadUiType("workout_inner.ui")[0] #Load the workouts inner page UI
admin_win = uic.loadUiType("admin_pg.ui")[0] #Load the admin page UI
BMI_win= uic.loadUiType("BMI.ui")[0] #Load the BMI UI
userDetails_win = uic.loadUiType("UserUpdate.ui")[0] #Load the user update UI
passwordChange_win = uic.loadUiType("passwordChange.ui")[0] #Load the password changeUI
cut_win= uic.loadUiType("cut.ui")[0] #Load the BMI UI
build_win = uic.loadUiType("build.ui")[0] #Load the build routine UI
transform_win = uic.loadUiType("transform.ui")[0] #Load the transform routine UI
finalscreen_win = uic.loadUiType("finalscreen.ui")[0] #Load the final screen UI
passwordreset1_win = uic.loadUiType("PasswordReset1.ui")[0] #Load the password reset UI
myworkouts_win = uic.loadUiType("myworkouts.ui")[0] #Load the my workouts UI

#Class - login window
class LoginWindow(QtGui.QMainWindow, login_win):
    def __init__(self, parent=None): #Loads the UI and sets it up
        QtGui.QMainWindow.__init__(self, parent)
        QString=type("")
        self.username_str=""
        self.password_str=""
        self.setupUi(self)
        self.loginBtn.clicked.connect(self.open_welcome_window) #If login button clicked go to open third function
        self.username_edit.setMaxLength(20)#Sets max length allowed in username
        self.password_edit.setMaxLength(20)#Sets max length allowed in password
        palette = QtGui.QPalette()  #Palette for the colours
        self.actionQuit.triggered.connect(self.quit) #Can quit using shortcut
        self.actionQuit.setShortcut(QtGui.QKeySequence("ESC")) #Use ESC to quit
        self.actionEnter.triggered.connect(self.open_welcome_window) #Once the button is triggered it goes to the function
        self.actionEnter.setShortcut(QtGui.QKeySequence("RETURN")) #Setting ENTER key to go to the next screen
        self.register_lbl.setPalette(palette) #Sets the colour
```

```
        self.help_lbl.mousePressEvent = self.see_help
        self.register_lbl.mousePressEvent = self.open_register_window
        self.reset_lbl.mousePressEvent = self.open_reset_window


    def open_register_window(self, event): #Hides current window and shows the register window
        self.hide()
        self.password_edit.clear()
        self.username_edit.clear()
        register_window.show()  #Opens up register page
    def open_welcome_window(self): #Opens the welcome window
        self.username_str=self.username_edit.text()
        self.password_str=self.password_edit.text()
        try: #Try and see if username and passwords match admin passwords
            if self.username_str==real_username: #If user input username is equal to admin username
                if self.password_str==real_password: #If user input password is equal to admin password
                    self.hide()
                    admin_window.show()
                    self.username_edit.clear()
                    self.password_edit.clear()
                else: #Else display an error
                    QtGui.QMessageBox.critical(self, "Error", "Username or password incorrect")
            cur.execute("""SELECT Username, Password FROM Customers WHERE
Username=?""",(self.username_str,))
            user_details_list=cur.fetchall()
            data_username_str=user_details_list[0][0]
            data_password_str=user_details_list[0][1]
            print(user_details_list)
        except: #If they dont match
            QtGui.QMessageBox.critical(self, "Error", "Login unsuccessful")
        else: #If any other input check the database data
```

```
        if self.username_str==data_username_str: #If username is equal to username from database
            self.password_str=hashlib.md5(self.password_str.encode('utf-8')).hexdigest()
            if self.password_str==data_password_str: #If password is equal to password from database
- hide current window
                QtGui.QMessageBox.information(self, "Welcome", "Login successful")
                self.hide()
                welcome_window.show()
                self.username_edit.clear()
                self.password_edit.clear()
        else: #Otherwise display an error message
            QtGui.QMessageBox.critical(self, "Error", "Username or password wrong")


    def see_help(self, event): #Function for the help button displaying help message
        QtGui.QMessageBox.information(self, "Help", "If you don't have an account, register for a free
one")
    def quit(self): #Function which asks user to input yes or no then quits or stays depending on choice
        msgBox = QtGui.QMessageBox()
        msgBox.setText('Are you sure you want to quit?')
        msgBox.addButton(QtGui.QPushButton('Yes'), QtGui.QMessageBox.YesRole)
        msgBox.addButton(QtGui.QPushButton('No'), QtGui.QMessageBox.NoRole)
        ret = msgBox.exec_()
        if ret==0: #If the answer is 'no'
            self.close()
        else: #If the answer is 'yes'
            msgBox.hide()


    def open_reset_window(self, event): #Hides the current window and opens the password reset
window
        self.username_edit.clear()
        self.password_edit.clear()
        self.hide()
        password_reset_window.show()
```

```
#Class - register window 1
class RegisterWindow1(QtGui.QMainWindow, register_pg1_win):
    def __init__(self, parent=None): #Sets up the UI and assign enter button for confirmation of input
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.username_reg_edit.setMaxLength(20)
        self.password_reg_edit.setMaxLength(20)
        self.password_reenter_edit.setMaxLength(20)
        self.actionEnter.triggered.connect(self.next_page)
        self.actionEnter.setShortcut(QtGui.QKeySequence("RETURN"))
        self.homeBtn.clicked.connect(self.goback)
        self.nextBtn.clicked.connect(self.next_page)
    def goback(self):
        self.hide() #Hides current window
        login_window.show() #Shows previous window
    def next_page(self): #Goes to the next page when button clicked
        self.username_str=self.username_reg_edit.text()
        self.password_str=self.password_reg_edit.text()
        self.repassword_str=self.password_reenter_edit.text()
        cur.execute("SELECT Username, Password FROM Customers WHERE
Username=?",(self.username_str,))
```

```
data_list=cur.fetchall()
try: #Tries to print the first index of the fetched list
  print(data_list[0][0])
except: #If it does read the list inputs checked against criteria
   if self.password_str=="": #If input password is empty
      QtGui.QMessageBox.critical(self, "Error", "Password box(es) empty")
   elif len(self.username_str)<5: #If username is shorter than 5 in length
      QtGui.QMessageBox.critical(self, "Error", "Username too short")
   elif len(self.password_str)<5: #If password is shorter than 5 in length
      QtGui.QMessageBox.critical(self, "Error", "Password too short")
   elif self.password_str!=self.repassword_str: #If password and re-enter password dont match
      QtGui.QMessageBox.critical(self, "Error", "Passwords don't match")
   elif self.username_str=="": #If the username is empty
      QtGui.QMessageBox.critical(self, "Error", "Username empty")
   elif self.username_str==self.password_str: #If the username is equal to the password
      QtGui.QMessageBox.critical(self, "Error", "Username and password can't be the same")
   elif ' ' in self.username_str: #If there is a space in username
      QtGui.QMessageBox.critical(self, "Error", "Username invalid")
   elif ' ' in self.password_str: #If there is a space in password
      QtGui.QMessageBox.critical(self, "Error", "Spaces in password")
   else: #Otherwise clear line edits and proceed to next page
      self.username_reg_edit.clear()
      self.password_reg_edit.clear()
      self.password_reenter_edit.clear()
      self.hide()
      register_window_page2.show()

else: #Otherwise display error message because username exists
   QtGui.QMessageBox.critical(self, "Error", "That username already exists")
```

```
#Class - welcome window
class WelcomeWindow(QtGui.QMainWindow, welcome_win):
    def __init__(self, parent=None): #Sets up the UI
        QtGui.QMainWindow.__init__(self,parent)
        self.setupUi(self)
        self.myworkoutsBtn.clicked.connect(self.open_myworkouts_window)
        self.workoutsBtn.clicked.connect(self.open_workouts_window)
        self.calorieBtn.clicked.connect(self.open_BMI_window)
        self.updateProfileBtn.clicked.connect(self.open_profile_window)
        self.logoutBtn.clicked.connect(self.logout)
    def logout(self): #Logout function
        login_window.password_edit.clear()
        login_window.password_edit.clear()
        msgBox = QtGui.QMessageBox()
        msgBox.setText('Are you sure you want to log out?')
        msgBox.addButton(QtGui.QPushButton('Yes'), QtGui.QMessageBox.YesRole)
        msgBox.addButton(QtGui.QPushButton('No'), QtGui.QMessageBox.NoRole)
        ret = msgBox.exec_()
        if ret==0: #If the answer is 'yes' - log out
            self.close()
            login_window.show()
        else: #If the anwser is 'no' hide message box
            msgBox.hide()
```

```
    def open_profile_window(self): #Hides current window and opens user details window
        self.hide()
        user_details_window.show()
    def open_workouts_window(self): #Hides current window and opens workouts window
        self.hide()
        workouts_window.show()
    def open_BMI_window(self):  #Hides current window and opens bmi window
        self.hide()
        bmi_window.show()
    def open_myworkouts_window(self): #Hides current window and opens my workouts window
        self.hide()
        myworkouts_window.show()
```

```
#Class - register window page 2
class RegisterWindow2(QtGui.QMainWindow, register_pg2_win):
    def __init__(self, parent=None): #Sets up UI and link buttons to function
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
```

```
        self.backBtn.clicked.connect(self.goback)
        self.nextBtn.clicked.connect(self.gonext)
    def goback(self): #Hides current screen and opens register window
        self.hide()
        register_window.show()
    def gonext(self): #Checks criteria and then registers the user
        self.forename_str=self.forename_edit.text()
        self.surname_str=self.surname_edit.text()
        self.email_str=self.email_edit.text()
        register_window.password_str=hashlib.md5(register_window.password_str.encode('utf-8')).hexdigest()
        self.postcode_str=self.postcode_edit.text()
        self.weight_float=self.weight_edit.text()
        self.height_float=self.height_edit.text()
        self.gender_bool=self.gender_combo.currentText()
        self.goal_name_str=self.goal_combo.currentText()
        if self.goal_name_str=="Cut": #If the input in the combo box is 'Cut' set goal to 1
            self.goal_int=1
        elif self.goal_goal_name_str=="Build": #If the input in the combo box is 'Build' set goal to 2
            self.goal_int=2
        elif self.goal_goal_name_str()=="Transform": #If the input in the combo box is 'Transform' set goal to 3
            self.goal_int=3
        if not re.match("^[a-z]*$", self.forename_str.lower()): #If there is something else other than letters in forename display error message
            QtGui.QMessageBox.critical(self, "Error", "Only letters a-z in forename allowed")
        elif self.forename_str=="": #If forename is empty
            QtGui.QMessageBox.critical(self, "Error", "Forename empty")
        elif not re.match("^[a-z]*$", self.surname_str.lower()): #If there is something else other than letters in surname display error message
            QtGui.QMessageBox.critical(self, "Error", "Only letters a-z in surname allowed")
        elif self.surname_str=="": #If the surname is empty display error message
```

```
            QtGui.QMessageBox.critical(self, "Error", "Surname empty")
        else: #Else write all the details in to the database and hide current window + show the welcome window
            cur.execute("INSERT INTO Customers (Username, Password, Name, Surname, Postcode, Email, Weight, Height, Gender, GoalID) VALUES ( ?, ?, ?, ?, ?, ?, ?, ?, ?, ? )",
                        (register_window.username_str, register_window.password_str, self.forename_str, self.surname_str, self.postcode_str, self.email_str, self.weight_float, self.height_float, self.gender_bool, self.goal_int))
            con.commit()
            self.close()
            welcome_window.show()



#Class - workouts window
class WorkoutsWindow(QtGui.QMainWindow, workouts_inner_win):
    def __init__(self, parent=None): #Sets up UI and link buttons to function
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        trainer_name_list=[]
        self.backBtn.clicked.connect(self.goback)
        cur.execute("select * from Trainers")
        trainers_data_list=cur.fetchall()
        for each in trainers_data_list: #For each piece of data in the list add it to a seperate list
            trainer_name_list.append(each[1])
        self.trainer_name_combo.addItems(trainer_name_list)
        self.calendar.clicked[QtCore.QDate].connect(self.showDate)
        global selected_date
        selected_date = self.calendar.selectedDate()
        self.selected_date_lbl.setText(selected_date.toString())
        palette = QtGui.QPalette()
```

```
        palette.setColor(QtGui.QPalette.Foreground,QtCore.Qt.white) #Change colour here
        self.selected_date_lbl.setPalette(palette) #Sets the colour
        current_time_str=time.strftime("%d/%m/%Y")
        self.calendar.setMinimumDate(QDate(selected_date))
        self.workoutBtn.clicked.connect(self.goalread)
        self.proceedBtn.clicked.connect(self.proceed)
    def proceed(self): #When the procced button is clicked run this function
        self.calendar.clicked[QtCore.QDate].connect(self.showDate)
        selected_date = self.calendar.selectedDate()
        global new_date
        new_date = selected_date.toString("dd/MM/yyyy")
        global trainer_name
        trainer_name=self.trainer_name_combo.currentText()
        self.hide()
        booking_window.show()
    def goalread(self): #Once the workout button is clicked select goal id from customer
        cur.execute("select GoalID FROM Customers WHERE
Username=?""",(login_window.username_str,))
        goal_list=cur.fetchone()
        for each in goal_list: #For each item in the goal list set goal int to each
            Goal_int=each
        if Goal_int==1: #If the goal is 1 then show the cut routines window
            self.hide()
            cut_window.show()
        elif Goal_int==2: #If the goal is 2 then show the build routines window
            self.hide()
            build_window.show()
        elif Goal_int==3: #If the goal is 3 then show the transform routines window
            self.hide()
            transform_window.show()
```

```
    def showDate(self, date): #Set the label to the date selected
        self.selected_date_lbl.setText(selected_date.toString())

    def goback(self): #Hides current window and goes to welcome window
        self.hide()
        welcome_window.show()
```

```
#Class - admin window
class AdminWindow(QtGui.QMainWindow, admin_win):
    def __init__(self, parent=None): #Sets up UI and link buttons to function
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.data=[]
        self.model = QtGui.QStandardItemModel(self)
        self.load_data()
        self.tableView.setModel(self.model)
        self.homeBtn.clicked.connect(self.gohome)
        self.updateBtn.clicked.connect(self.update)
```

```
        self.graphBtn.clicked.connect(self.graph)
        self.allworkoutsBtn.clicked.connect(self.allworkouts)
        self.allusersBtn.clicked.connect(self.load_data)
    def gohome(self): #If home button is pressed hide current window and show login window
        self.hide()
        login_window.show()
    def load_data(self): #Loads the data using the customers table data and assigns it to the table view
            s='select * from Customers'
            cur.execute(s)
            self.data_list = cur.fetchall() #query data comes back
            if len(self.data_list)>0: #If the length of the list is bigger than 0
                for i in range(len(self.data_list)-1,-1): #Whilst i is in the range of the length of the data list
remove row
                    print("removing row",i)
                    self.data_list.pop(i)
                    self.model.removeRow(index.row(self.data_list[i]))
            self.model=QtGui.QStandardItemModel(self)
            self.tableView.setModel(self.model)
            for row in self.data_list:    #For each row in the self data list
                    items = [
                        QtGui.QStandardItem(str(field))
                        for field in row #For each field in row set the name for the column
                    ]
                    self.model.appendRow(items)
            self.model.setHeaderData(0, QtCore.Qt.Horizontal, "CustomerID")
            self.model.setHeaderData(1, QtCore.Qt.Horizontal, "Username")
            self.model.setHeaderData(2, QtCore.Qt.Horizontal, "Password")
            self.model.setHeaderData(3, QtCore.Qt.Horizontal, "Name")
            self.model.setHeaderData(4, QtCore.Qt.Horizontal, "Surname")
            self.model.setHeaderData(5, QtCore.Qt.Horizontal, "DOB")
            self.model.setHeaderData(6, QtCore.Qt.Horizontal, "Postcode")
```

```
        self.model.setHeaderData(7, QtCore.Qt.Horizontal, "Email")
        self.model.setHeaderData(8, QtCore.Qt.Horizontal, "Weight")
        self.model.setHeaderData(9, QtCore.Qt.Horizontal, "Height")
        self.model.setHeaderData(10, QtCore.Qt.Horizontal, "Gender")
        self.model.setHeaderData(11, QtCore.Qt.Horizontal, "Goal")
        self.model.setHeaderData(12, QtCore.Qt.Horizontal, "PreviousWeight")
        self.model.setHeaderData(13, QtCore.Qt.Horizontal, "PreviousHeight")

    def show_selected(self): #Shows the currently selected data
        index = self.tableView.currentIndex() #returns currently selected cell
        row_index=index.row() #returns the row of the currently selected cell
        column_index=index.column()#returns the column of the currently selected cell
        cell_contents=index.data()#returns the contents of the currently selected cell
    def update(self): #Updates the database with the changed data in the table view
        index = self.tableView.currentIndex() #returns currently selected cell
        row_index=index.row() #returns the row of the currently selected cell
        colum_index=index.column()#returns the column of the currently selected cell
        cell_contents=index.data()#returns the contents of the currently selected cell
        id=self.model.data(self.model.index(index.row(), 0))
        index_int1=self.model.data(self.model.index(index.row(), 1))
        index_int2=self.model.data(self.model.index(index.row(), 2))
        index_int3=self.model.data(self.model.index(index.row(), 3))
        index_int4=self.model.data(self.model.index(index.row(), 4))
        index_int5=self.model.data(self.model.index(index.row(), 5))
        index_int6=self.model.data(self.model.index(index.row(), 6))
        index_int7=self.model.data(self.model.index(index.row(), 7))
        index_int8=self.model.data(self.model.index(index.row(), 8))
        index_int9=self.model.data(self.model.index(index.row(), 9))
        index_int10=self.model.data(self.model.index(index.row(), 10))
        index_int11=self.model.data(self.model.index(index.row(), 11))
        self.model.data(self.model.index(4,1))
```

```
        #parameter query - get criteria from Python
        cur.execute("""UPDATE Customers SET
Username='%s',Password='%s',Name='%s',Surname='%s',DOB='%s',Postcode='%s',Email='%s',Weight
='%s',Height='%s',Gender='%s',GoalID='%s'
                    WHERE
CustomerID='%s'"""%(index_int1,index_int2,index_int3,index_int4,index_int5,index_int6,index_int7,
index_int8,index_int9,index_int10,index_int11,id)) #not case sensitive inside the string
        msgBox = QtGui.QMessageBox()
        msgBox.setText('Are you sure you want to save changes?')
        msgBox.addButton(QtGui.QPushButton('Yes'), QtGui.QMessageBox.YesRole)
        msgBox.addButton(QtGui.QPushButton('No'), QtGui.QMessageBox.NoRole)
        ret = msgBox.exec_()
        if ret==0: #If answer is 'yes' save the changes
            con.commit()
            QtGui.QMessageBox.information(self, "Success", "Changes saved")
        else: #If anwser is 'no' hide the message box
            msgBox.hide()
    def graph(self): #Plots the graph dependent on number of male and female users in the database
        cur.execute('select distinct Gender, count(Gender) as CountOf from Customers group by
Gender')
        global gender_list
        gender_list=cur.fetchall()
        print(gender_list)
        # Data to plot
        try: #Try the total of the both genders count
            total_int=(gender_list[0][1])+(gender_list[1][1])
            print(total_int)
        except: #If doesn't work total is equal to just one of those counts
            total_int=gender_list[0][1]
        else: #Else print error
            print("error")
        male_per=(gender_list[1][1]/total_int)*360
        female_per=(gender_list[0][1]/total_int)*360
```

```
        labels_str = gender_list[0][0], gender_list[1][0]
        sizes_list = [female_per, male_per]
        colors_list = ['pink', 'blue']
        explode_values = (0.1, 0)  # explode 1st slice

        # Plot
        plt.pie(sizes_list, explode=explode_values, labels=labels_str, colors=colors_list,
              autopct='%1.1f%%', shadow=True, startangle=140)

        plt.axis('equal')
        plt.show()



    def allworkouts(self): #Displays all the workouts using an inner join function
        s='select Workouts.WorkoutID,Workouts.DateBooked, Customers.Name from Workouts inner join Customers on Workouts.CustomerID=Customers.CustomerID'
        cur.execute(s)
        self.distinct_list = cur.fetchall() #query data comes back
        if len(self.distinct_list)>0:#If the length of the list is bigger than 0
            for i in range(len(self.distinct_list)-1,-1):#Whilst i is in the range of the length of the data list remove row
                print("removing row",i)
                self.distinct_list.pop(i)
                self.model.removeRow(index.row(self.distinct_list[i]))
        self.model=QtGui.QStandardItemModel(self)
        self.tableView.setModel(self.model)
        for row in self.distinct_list: #For each row in the distinct list create an items list
                items = [
                    QtGui.QStandardItem(str(field))
                    for field in row #For each field in row set the name for the column
                ]
                self.model.appendRow(items)
```

```
        self.model.setHeaderData(0, QtCore.Qt.Horizontal, "WorkoutID")
        self.model.setHeaderData(1, QtCore.Qt.Horizontal, "DateBooked")
        self.model.setHeaderData(2, QtCore.Qt.Horizontal, "Name")


#Class - BMI window
class BMIWindow(QtGui.QMainWindow, BMI_win):
    def __init__(self, parent=None): #Sets up the UI and connects all the buttons to the functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.data=[]
        self.model = QtGui.QStandardItemModel(self)
        self.tableView.setModel(self.model)
        welcome_window.calorieBtn.clicked.connect(self.load_data)
        self.calculateBtn.clicked.connect(self.calculate)
        self.backBtn.clicked.connect(self.goback)
        self.graphBtn.clicked.connect(self.graph)
    def goback(self): #Resets the labels, hides the current window and goes back to the welcome
window
        self.hide()
        self.height_lbl.setText("")
        self.weight_lbl.setText("")
        self.BMI_lbl.setText("")
        self.result_lbl.setText("")
        welcome_window.show()
    def calculate(self): #Calculates the BMI for the user and displays according to what the outcome is
        cur.execute('SELECT Weight, Height FROM Customers WHERE
Username=?',(login_window.username_str,))
        self.weight_list = cur.fetchall()
        dataWeight_float=self.weight_list[0][0]
        dataHeight_float=self.weight_list[0][1]
        dataHeightMetres_float=dataHeight_float/100
        self.BMI_float=dataWeight_float/(dataHeightMetres_float*dataHeightMetres_float)
```

```
        self.BMI_float=round((self.BMI_float),2)
        self.weight_lbl.setText(str(dataWeight_float)+"kg")
        self.weight_lbl.setStyleSheet('color: white')
        self.height_lbl.setText(str(dataHeight_float)+"cm")
        self.height_lbl.setStyleSheet('color: white')
        self.BMI_lbl.setText(str(self.BMI_float))
        self.BMI_lbl.setStyleSheet('color: white')
        if self.BMI_float <=19.5: #If the BMI is below or equal to 19.5 display underweight
            self.result_lbl.setText("Underweight")
            self.result_lbl.setStyleSheet('color: white')
        elif self.BMI_float >19.5 and self.BMI_float<25:#If the BMI is larger than 19.5 and smaller than
25 display healthy weight
            self.result_lbl.setText("Healthy Weight")
            self.result_lbl.setStyleSheet('color: white')
        elif self.BMI_float >25 and self.BMI_float <30: #If the BMI is larger than 25 and smaller than 30
display overweight
            self.result_lbl.setText("Overweight")
            self.result_lbl.setStyleSheet('color: white')
        elif self.BMI_float >30 and self.BMI_float <35: #If the BMI is larger than 30 and smaller than 35
display obese
            self.result_lbl.setText("Obese")
            self.result_lbl.setStyleSheet('color: white')
        elif self.BMI_float >35 and self.BMI_float <40: #If the BMI is larger than 35 and smaller than 40
display severly obese
            self.result_lbl.setText("Severely Obese")
            self.result_lbl.setStyleSheet('color: white')
        elif self.BMI_float >40: #If the BMI is larger than 40 display morbidly obese
            self.result_lbl.setText("Morbidly Obese")
            self.result_lbl.setStyleSheet('color: white')
    def load_data(self): #Load the data and select previous weight and weight from customers table
        cur.execute('SELECT PreviousWeight, Weight from Customers WHERE
Username=?',(login_window.username_str,))
        self.new_weight_list = cur.fetchall()
```

```
        global previousweight
        global weight
        self.previousweight_int=self.new_weight_list[0][0]
        self.weight_int=self.new_weight_list[0][1]




    def graph(self):
            template='''<!DOCTYPE html>
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
    </style>
  </head>
  <body>
    <button onclick="test();">Launch</button>
    <p id="out"> </p>
    <canvas id="myCanvas" width="150" height="425"></canvas>

    <script>
       var canvas = document.getElementById('myCanvas');
      var context = canvas.getContext('2d');
      var a=new Array(5, 8);
function test(){

      var arrayLength = a.length;
```

```
for (var i = 0; i < arrayLength; i++) {

   //Do something
  draw_bar(50,i);
  //document.getElementById("out").innerHTML+=","+a[i];
}

    }
    function draw_bar(size,ind){

    context.beginPath();

    context.rect(size*ind, canvas.height-size/16*a[ind], size,size/8*a[ind]);
    context.fillStyle = 'yellow';
    context.fill();
    context.lineWidth = 4;
    context.strokeStyle = 'black';
    context.stroke();

       context.font = "20px Arial";
      context.fillStyle = "red";
      context.textAlign = "centre";
context.fillText(a[ind],size*ind+size/2-10,canvas.height-size/16*a[ind]-10);
    }
   </script>
</body>
</html> '''
            b=(str(self.previousweight_int)+", "+str(self.weight_int))
            print(b)
            url=template.replace("5, 8", b)
            self.webView.setHtml(url)
```

```
#Class - User Details Window
class  UserDetailsWindow(QtGui.QMainWindow, userDetails_win):
    def __init__(self, parent=None): #Sets up the UI and links buttons to functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        welcome_window.updateProfileBtn.clicked.connect(self.both)
        self.updateBtn.clicked.connect(self.updateUser)
        self.change_passBtn.clicked.connect(self.change_pass)
    def goback(self): #Hides the current window and displays the welcome window
        self.hide()
        welcome_window.show()



    def loadDetails(self): #Loads the details of the currently logged in user
        cur.execute('SELECT Name, Surname, Postcode, Email, Weight, Height, Gender, PreviousWeight,
PreviousHeight, GoalID FROM Customers WHERE Username=?',(login_window.username_str,))
        self.details_list = cur.fetchall()
        for each in self.details_list: #For each in the list of the users details assign each of the items to a
variable
            global NAME_str
            global SURNAME_str
            global POSTCODE_str
            global EMAIL_str
            global GENDER_bool
            NAME_str=each[0]
            SURNAME_str=each[1]
            POSTCODE_str=each[2]
            EMAIL_str=each[3]
            global WEIGHT_float
            global HEIGHT_float
```

```
            WEIGHT_float=each[4]
            HEIGHT_float=each[5]
            GENDER_bool=each[6]
            global PREVIOUS_WEIGHT_float
            global PREVIOUS_HEIGHT_float
            PREVIOUS_WEIGHT_float=each[7]
            PREVIOUS_HEIGHT_float=each[8]
            global GOALID_int
            GOALID_int=each[9]
        print("current values: " ,NAME_str, SURNAME_str, POSTCODE_str, EMAIL_str , WEIGHT_float, HEIGHT_float, GENDER_bool, PREVIOUS_WEIGHT_float, PREVIOUS_HEIGHT_float, GOALID_int)
    def insertvalues(self): #Insert these values from the textboxes as updated and into the database
        if GOALID_int==1: #If the goal id is 1 set the goal string to 'cut'
            GOAL_str="Cut"
        elif GOALID_int==2: #If the goal id is 2 set the goal string to 'build'
            GOAL_str="Build"
        elif GOALID_int==3: #If the goal id is 3 set the goal string to 'transform'
            GOAL_str="Transform"
        self.forename_edit.setText(NAME_str)
        self.surname_edit.setText(SURNAME_str)
        self.goal_combo.clear()
        GOAL_list=["Build","Transform","Cut"]
        self.goal_combo.addItems(GOAL_list)
        find_index_int=self.goal_combo.findText(GOAL_str)
        self.goal_combo.setCurrentIndex(find_index_int)
        self.postcode_edit.setText(POSTCODE_str)
        self.email_edit.setText(EMAIL_str)
        self.weightEdit_2.setValue(WEIGHT_float)
        self.heightEdit_2.setValue(HEIGHT_float)
    def both(self): #Loads both of the loadDetails() and insertvalues() functions
        self.loadDetails()
```

```
        self.insertvalues()
    def updateUser(self): #Function where the SQL query is run to update the user details
        self.updatedForename_str=self.forename_edit.text()
        self.updatedSurname_str=self.surname_edit.text()
        self.updatedPostcode_str=self.postcode_edit.text()
        self.updatedEmail_str=self.email_edit.text()
        self.updatedGoal_str=self.goal_combo.currentText()
        if self.updatedGoal_str=="Cut": #If the updated goal is cut then set the id to 1
            self.updatedGoal_str=1
        elif self.updatedGoal_str=="Build": #If the updated goal is build then set the id to 2
            self.updatedGoal_str=2
        elif self.updatedGoal_str=="Transform": #If the updated goal is transform then set the id to 3
            self.updatedGoal_str=3
        self.updatedGender_bool=self.gender_combo.currentText()
        self.updatedHeight_float=self.heightEdit_2.text()
        self.updatedWeight_float=self.weightEdit_2.text()
        self.updatedWeight_float=int(float(self.updatedWeight_float))
        if self.updatedWeight_float == WEIGHT_float: #If the updated weight is equal to the current
weight then update all the details and previous weight with previous weight
            cur.execute("""UPDATE Customers SET
Name='%s',Surname='%s',Postcode='%s',Email='%s',Weight='%s',Height='%s',Gender='%s',
GoalID='%s', PreviousWeight='%s' WHERE Username='%s'"""%

(self.updatedForename_str,self.updatedSurname_str,self.updatedPostcode_str,self.updatedEmail_s
tr,self.updatedWeight_float,self.updatedHeight_float,self.updatedGender_bool,self.updatedGoal_st
r, PREVIOUS_WEIGHT_float,login_window.username_str))
            con.commit()
            self.loadDetails()
        else: #Else update the previous weight with the current weight
            cur.execute("""UPDATE Customers SET
Name='%s',Surname='%s',Postcode='%s',Email='%s',Weight='%s',Height='%s',Gender='%s',
GoalID='%s',PreviousWeight='%s' WHERE Username='%s'"""%

(self.updatedForename_str,self.updatedSurname_str,self.updatedPostcode_str,self.updatedEmail_s
```

```
tr,self.updatedWeight_float,self.updatedHeight_float,self.updatedGender_bool,self.updatedGoal_str, WEIGHT_float,login_window.username_str))
            con.commit()
            self.loadDetails()


    def change_pass(self): #Hides the current window and shows the password change window
        self.hide()
        password_change_window.show()


#Class - Password Change Window
class  PasswordChangeWindow(QtGui.QMainWindow, passwordChange_win):
    def __init__(self, parent=None): #Sets up the UI and links the buttons to the functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        self.saveChangesBtn.clicked.connect(self.changepass)
    def goback(self): #Hides current window and shows the user details window
        self.hide()
        user_details_window.show()
    def changepass(self): #Reads the password from the database for the currently logged in user and updates their password with their new input
        cur.execute('SELECT Password FROM Customers WHERE Username=?',(login_window.username_str,))
        self.password_list=cur.fetchall()
        self.current_password_str=self.password_edit.text()
        hashed_password_str=self.password_list[0][0]
        self.current_password2_str=hashlib.md5(self.current_password_str.encode('utf-8')).hexdigest()
        if hashed_password_str==self.current_password2_str: #If the password from the db is equal to the hash of the current password input update the user's password
```

```
            self.new_password_str=self.newpassword_edit.text()
            newest_password_str=hashlib.md5(self.new_password_str.encode('utf-8')).hexdigest()
            QtGui.QMessageBox.information(self, "Success", "Password updated")
            cur.execute("""UPDATE Customers SET Password='%s' WHERE
Username='%s'"""%(str(newest_password_str), login_window.username_str))
            con.commit()

        else: #Else display an error saying the current password is wrong
            QtGui.QMessageBox.critical(self, "Error", "Current password incorrect")



#Class - cut routines window
class CutWindow(QtGui.QMainWindow, cut_win):
    def __init__(self, parent=None): #Sets up the UI and links the buttons to the functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        self.plan1Btn.clicked.connect(self.plan1)
        self.plan2Btn.clicked.connect(self.plan2)
        self.plan3Btn.clicked.connect(self.plan3)
    def plan1(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1001
        global duration_str
        duration_str=str(1.5)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="1" OR ExerciseID="3" OR
ExerciseID="18" OR ExerciseID="9" OR ExerciseID="15"')
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()
```

```
    def plan2(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1002
        global duration_str
        duration_str=str(1)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="12" OR ExerciseID="8" OR
ExerciseID="10" OR ExerciseID="19" OR ExerciseID="2"')
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()
    def plan3(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1003
        global duration_str
        duration_str=str(2)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="11" OR ExerciseID="5" OR
ExerciseID="7" OR ExerciseID="14" OR ExerciseID="6"')
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()
    def goback(self): #Hides current window and shows the workouts window
        self.hide()
        workouts_window.show()



#Class - build routines window
class BuildWindow(QtGui.QMainWindow, build_win):
    def __init__(self, parent=None): #Sets up the UI and links the buttons to the functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        self.plan1Btn.clicked.connect(self.plan1)
```

```
        self.plan2Btn.clicked.connect(self.plan2)
        self.plan3Btn.clicked.connect(self.plan3)
    def plan1(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1004
        global duration_str
        duration_str=str(2)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="1" OR ExerciseID="7" OR
ExerciseID="15" OR ExerciseID="8" OR ExerciseID="10"')
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()

    def plan2(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1005
        global duration_str
        duration_str=str(2)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="2" OR ExerciseID="3" OR
ExerciseID="12" OR ExerciseID="16" OR ExerciseID="20"')
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()
    def plan3(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1006
        global duration_str
        duration_str=str(2)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="1" OR ExerciseID="13" OR
ExerciseID="10" OR ExerciseID="18" OR ExerciseID="15"')
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
```

```
        workouts_window.show()
    def goback(self):
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()



#Class - transform routines window
class TransformWindow(QtGui.QMainWindow, transform_win):
    def __init__(self, parent=None): #Sets up the UI and links the buttons to the functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        self.plan1Btn.clicked.connect(self.plan1)
        self.plan2Btn.clicked.connect(self.plan2)
        self.plan3Btn.clicked.connect(self.plan3)
    def plan1(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1007
        global duration_str
        duration_str=str(1.5)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="1" OR ExerciseID="8" OR
ExerciseID="10" OR ExerciseID="15" OR ExerciseID="5"')
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()


    def plan2(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1008
        global duration_str
        duration_str=str(2.5)
```

```
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="2" OR ExerciseID="8" OR
ExerciseID="10" OR ExerciseID="12" OR ExerciseID="19"')
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()
    def plan3(self): #Selects all the exercises for the plan
        global planID_int
        planID_int=1009
        global duration_str
        duration_str=str(2.5)
        cur.execute('SELECT * FROM Exercises WHERE ExerciseID="5" OR ExerciseID="6" OR
ExerciseID="7" OR ExerciseID="11" OR ExerciseID="14"')
        self.fetched_list=cur.fetchall()
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()
    def goback(self):
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()
#Class - booking window
class BookingWindow(QtGui.QMainWindow, finalscreen_win):
    def __init__(self, parent=None): #Sets up the UI and links the buttons to the functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        workouts_window.proceedBtn.clicked.connect(self.total)
        self.bookBtn.clicked.connect(self.gobook)

    def total(self): #Calculates the total for the workout
        cur.execute('SELECT PaymentRates FROM Trainers WHERE TrainerName=?',(trainer_name,))
        payment_str=cur.fetchone()
        for each in payment_str: #For each in the payment list make it equal to the number
            payment_str=each
```

```
        cur.execute('SELECT TrainerID FROM Trainers WHERE TrainerName=?',(trainer_name,))
        trainer_id_int=cur.fetchone()
        global trainer_actual_int
        trainer_actual_int=trainer_id_int[0]
        print(trainer_actual_int)
        global price_float
        price_float=float(payment_str)*float(duration_str)
        price_float=str(price_float)
        price_float='£'+ price_float +'0'
        self.total_lbl.setText(price_float)
        self.total_lbl.setStyleSheet('color: white')
        self.total_lbl.resize(76, 78)
    def goback(self):
        self.hide() #Hides current window and shows the workouts window
        workouts_window.show()
    def gobook(self): #Confirms the workout by writing it to the database
        cur.execute("SELECT WorkoutID FROM Workouts ORDER BY WorkoutID DESC LIMIT 1")
        workout_id_int=cur.fetchone()
        new_workout_id_int=workout_id_int[0]+1
        cur.execute("SELECT CustomerID FROM Customers WHERE
Username=?",(login_window.username_str,))
        user_id_int=cur.fetchone()
        actual_user_id_int=user_id_int[0]
        cur.execute("INSERT INTO Workouts (WorkoutID, ExercisePlanID, CustomerID, TrainerID,
DateBooked, Price) VALUES ( ?, ?, ?, ?, ?, ? )",
                    (new_workout_id_int, planID_int, actual_user_id_int, trainer_actual_int, new_date,
price_float))
        con.commit()
        QtGui.QMessageBox.information(self, "Success", "Workout successfully booked!")
        self.hide()
        welcome_window.show()
```

```
#Class - password reset window
class PasswordResetWindow(QtGui.QMainWindow, passwordreset1_win):
    def __init__(self, parent=None): #Sets up the UI and links the buttons to the functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.nextBtn.clicked.connect(self.gonext)
        self.backBtn.clicked.connect(self.goBack)
    def goBack(self): #Hides the current window and shows the login screen
        self.hide()
        login_window.show()
    def gonext(self): #Generates a random string to be used as the 'reset password'
        self.username1_str=self.username.text()
        self.email1_str=self.email.text()
        cur.execute('SELECT Username, Email FROM Customers WHERE
Username=?',(self.username1_str,))
        details_list=cur.fetchall()
        self.email2_str=details_list[0][1]
        if self.email1_str==self.email2_str: #If the email from the db is equal to the email input by user
generate random string
            string=""
            x=("ABCDEFGHIJKLMNOPQRSTUVWXYZ123456789")
            counter=0
            while counter<10: #Whilst the counter is less than 10 keep adding the random choice to the
string
                k=random.choice(x)
                string=k+string
                x = x.replace(k, "")
                counter=counter+1
            hashed_string_str=hashlib.md5(string.encode('utf-8')).hexdigest()
            server = smtplib.SMTP('smtp.gmail.com', 587)
            server.starttls()
            server.login("olekdonkey@gmail.com", "olek2511")
```

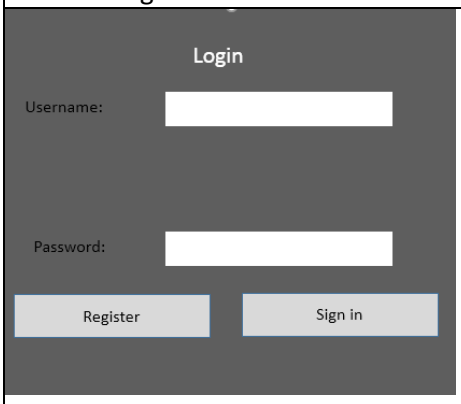
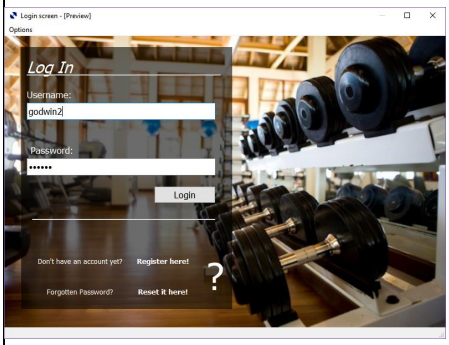
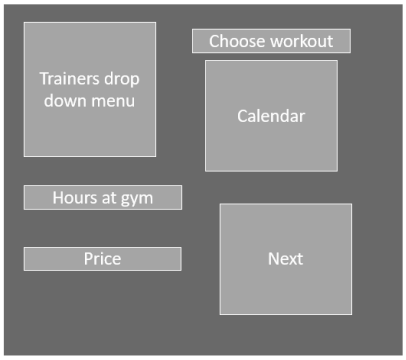
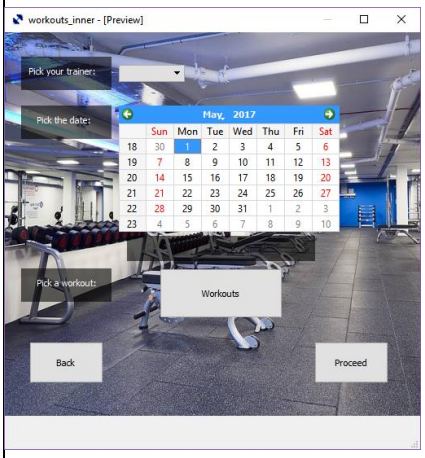
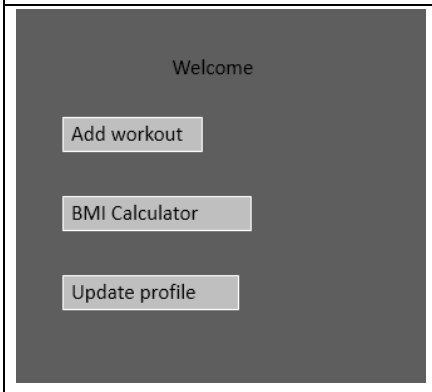
```
            msg = 'Subject: {}\n\n{}'.format("Password Reset", "Login with this new
password"+"\n"+"New Password: "+string)
            try: #Try sending an email with the reset password
                server.sendmail("olekdonkey@gmail.com", self.email2_str, msg)
                print("email sent, with code: ", string)
                server.quit()
                cur.execute("UPDATE Customers SET Password='%s' WHERE
Username='%s'"%(hashed_string_str, self.username1_str))
                con.commit()
                self.hide()
                login_window.show()
            except: #Error if email cannnot be found
                print("email not found")
        else: #Error if email cannnot be found
            print("email not found")
#Class - my workouts window
class MyWorkoutsWindow(QtGui.QMainWindow, myworkouts_win):
    def __init__(self, parent=None): #Sets up the UI and links the buttons to the functions
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.backBtn.clicked.connect(self.goback)
        self.data=[]
        self.model = QtGui.QStandardItemModel(self)
        welcome_window.myworkoutsBtn.clicked.connect(self.load_data)
        self.tableView.setModel(self.model)
    def load_data(self): #Loads the customer ID data
        cur.execute("""Select CustomerID from Customers WHERE
Username=?""",(login_window.username_str,))
        cust_id_list=cur.fetchone()
        cust_id1_int=cust_id_list[0]
        cur.execute('select * from Workouts WHERE CustomerID=?',(cust_id1_int,))
        self.data_list = cur.fetchall() #query data comes back
```

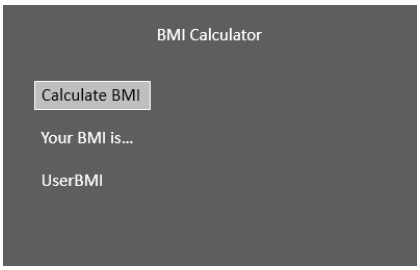
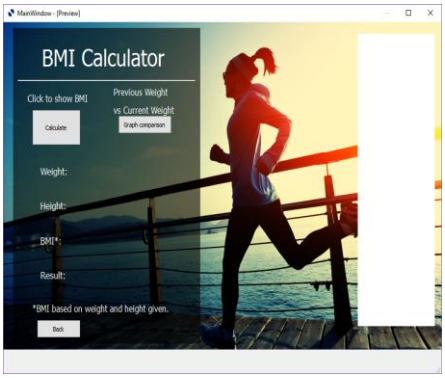
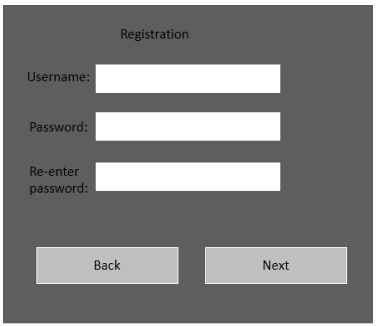
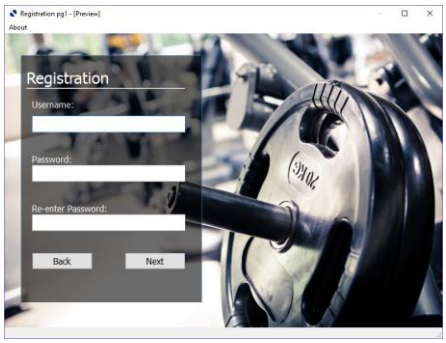
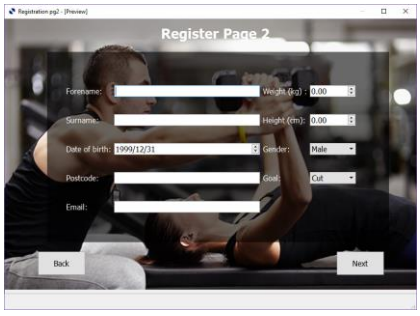
```
        #as a python 2D tuple  (read-only list)
        #clear all previus data, so that our model doesn't get longer and longer duplicating the records
        if len(self.data_list)>0: #If the length of the list is bigger than 1
            for i in range(len(self.data_list)-1,-1): #While the range of the length of the list keep going through the list
                self.data_list.pop(i)
                self.model.removeRow(index.row(self.data_list[i]))
        self.model=QtGui.QStandardItemModel(self)
        self.tableView.setModel(self.model)
        for row in self.data_list:    #For each row in the data create a items list
                items = [
                    QtGui.QStandardItem(str(field))
                    for field in row #For each field in row set a name for the columm
                ]
                self.model.appendRow(items)
        self.model.setHeaderData(0, QtCore.Qt.Horizontal, "WorkoutID")
        self.model.setHeaderData(1, QtCore.Qt.Horizontal, "ExercisePlanID")
        self.model.setHeaderData(2, QtCore.Qt.Horizontal, "CustomerID")
        self.model.setHeaderData(3, QtCore.Qt.Horizontal, "TrainerID")
        self.model.setHeaderData(4, QtCore.Qt.Horizontal, "DateBooked")
        self.model.setHeaderData(5, QtCore.Qt.Horizontal, "Price")

    def show_selected(self):
        index = self.tableView.currentIndex() #returns currently selected cell
        r=index.row() #returns the row of the currently selected cell
        c=index.column()#returns the column of the currently selected cell
        cell_contents=index.data()#returns the contents of the currently selected cell

    def goback(self): #Hides the current window and shows the welcome window
        self.hide()
        welcome_window.show()
```

```
#Linking each window
app = QtGui.QApplication(sys.argv)
#Linking the windows the to classes
login_window = LoginWindow(None)
register_window = RegisterWindow1(None)
welcome_window = WelcomeWindow(None)
register_window_page2 = RegisterWindow2(None)
workouts_window = WorkoutsWindow(None)
admin_window = AdminWindow(None)
bmi_window = BMIWindow(None)
user_details_window = UserDetailsWindow(None)
password_change_window = PasswordChangeWindow(None)
cut_window = CutWindow(None)
build_window = BuildWindow(None)
transform_window = TransformWindow(None)
booking_window = BookingWindow(None)
password_reset_window = PasswordResetWindow(None)
myworkouts_window = MyWorkoutsWindow(None)
##starting the application
login_window.show()
app.exec_()
```

Evaluation

Usability testing

Initial design	Final outcome	Comment
		The final outcome changed compared to the inital design. If I had more time I would have perhaps tried adding more custom made buttons to the design. Also, I would have made the window bigger
		Initally I thought I would fit all the information onto 1 single window, however I had to make 2 windows in order to make the coding work. Also, I did not want so much information on one screen as it didn't look visually appealing
		Initially I thought I would only need 3 buttons, however I had to add the my workouts button for the user to be able to view their workouts. Also, the window is much bigger. If I had more time I would have created smaller

		buttons on a smaller window.
		Initially, I didn't think I would need a button to display the graph. Also, the weight, height and result labels were not in the design. If I had more time I would have made a seperate window for the graph instead of having a white rectangle for the web view on the right side of the window
		This window was really close to my inital design. I only had to add the picture in the background. Also, I wanted to follow the design of my previous windows to make the windows match. If I had more time, I would have made the window smaller and added a darker picture in the background
		This window also stayed close to my inital design. I had to change the font to white since the picture was to bright and on the black box I could not have black font.

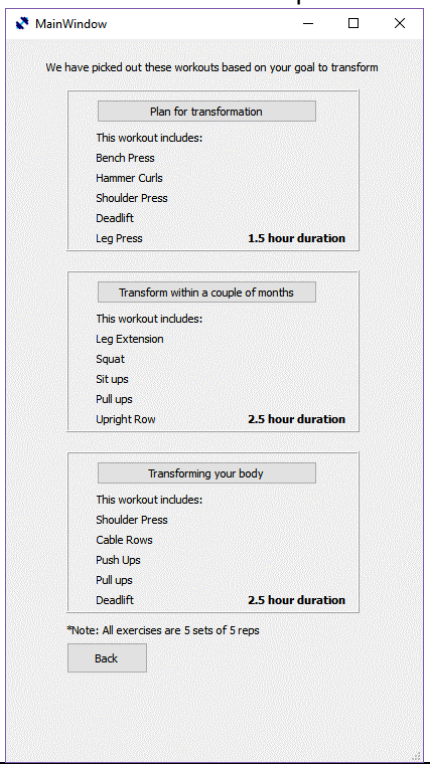
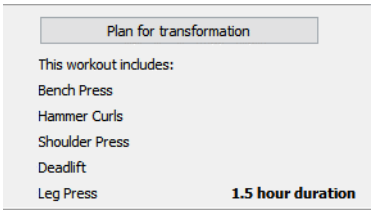
User feedback

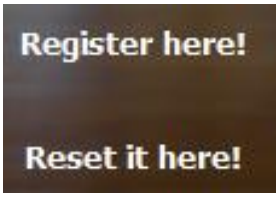
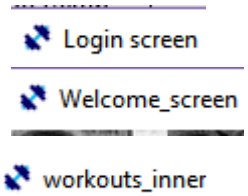
I got a family member to test my program. They went through the program as a completely new user and all I did was time them without helping them. First, they registered for a new account using their details. Then, they picked their workout, trainer, and date. After booking this, they calculated their BMI. They then edited their details and inputted a different weight to their previous one. Then they changed their password. Then they went into the BMI calculator, calculated their BMI and displayed the graph. Then, they viewed their workouts and logged out. Then, they reset their password which was sent to their email and then they logged in with this password and changed their password again. Then they finally logged out. The total time for this test was 6 minutes and 43 seconds. I then asked for feedback about the usage of the program. They said that there are of buttons throughout the whole program, especially when trying to display the graph in the BMI window. Also, they said they liked the design with the pictures. Some of the labels fonts were too small. Also, they didn't like the register label – preferred a button instead and it was unclear where to register at the start. They said they liked how your workouts could be displayed after you booked them, but didn't like how this was just ID's from the database and not data they could understand.

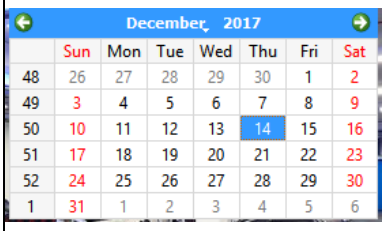
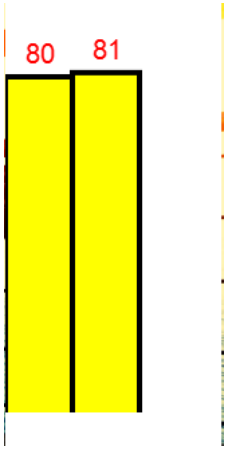
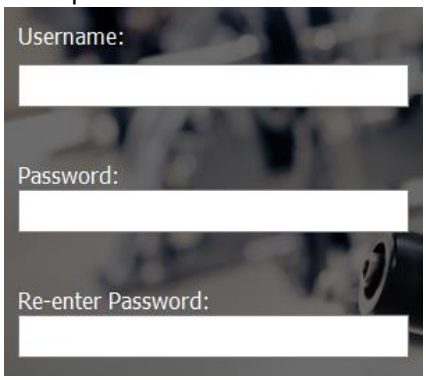
From this test, I realised a few things. Firstly, I had too many windows in my program. The constant button clicking was a clear issue which the test user didn't like. Also, the fonts on the label were too small which was another issue. Furthermore, the label for the register window was also an issue as it was unclear what it was meant for. Even though I understood it, a first-time user found it difficult to. The time of the test wasn't a concern for me – I knew that because of the validation I had sometimes it would take some time before the right username or password is typed in.

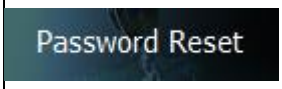
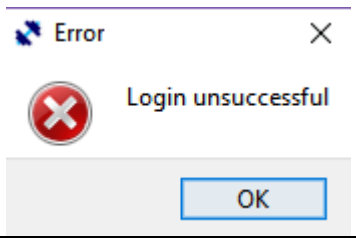
Evaluation of success criteria

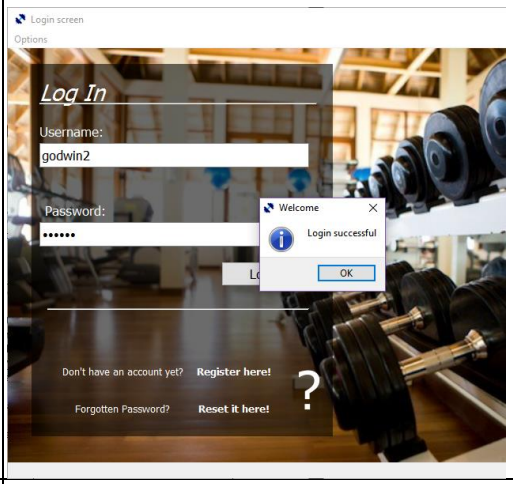
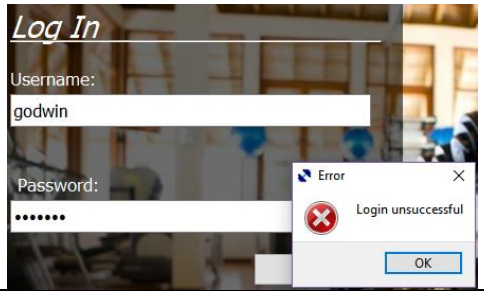
Test no.	Example of input	Relation to success criteria	Expected output	Actual output
1	Back button pressed (normal data)	User should be able to go to back to previous screen from any screen (apart from home screen)	Program will not crash – user will be taken to previous screen	User is taken to the previous screen
2	Back button pressed 10 times (extreme data)	The user should be able to keep going back quickly through the windows	Program should not crash – user should be able to keep going through screens	Program does not crash when going through the many windows

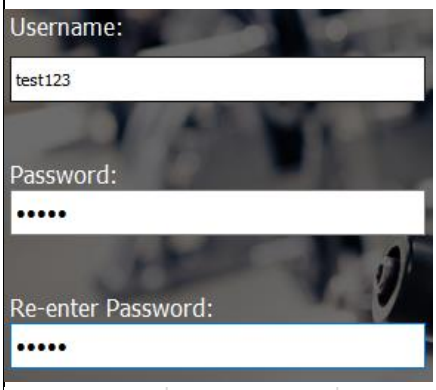
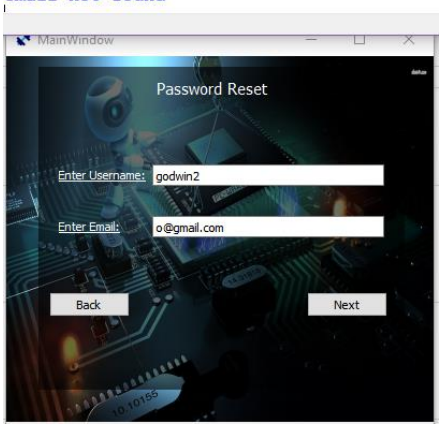
3	Check all windows have pictures relating to the gym (normal data)	An easy and visible interface for the user	All windows have a picture	Not all windows have a picture 
4	Check if there are any scroll bars within any windows (normal data)	The user should never have to scroll down within any window	To make sure the user doesn't scroll at all through any windows	No scrolling through any windows
5	Check the list of workouts and see if they can be selected with a button (normal data)	User must be able to select their workout to book it	User must be able to select their workout and these workouts must be based on their goal	Each workout has a button to select the given exercises 

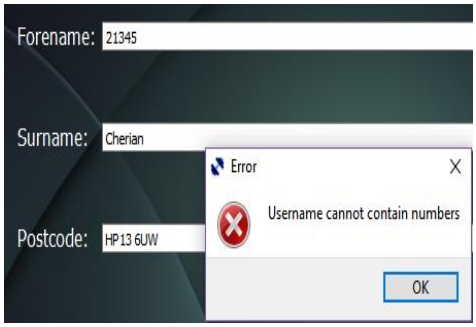
6	Check the text labels are in bold for registering a new account and resetting password (normal data)	The registration and reset password labels must be in bold and when clicked must take user to the window	To make sure the labels are visible and can navigate the user to the reset password/ register windows	Both of the labels are in bold – user said they are too hard to understand – not visible straightaway that clicking the label will register you. 
7	Check each window has a custom gym weights icon (normal data)	Each window must have an icon with gym weights	To make sure the interface is user friendly and makes the program unique	All windows have the gym icon 
8	Press 'tab' button in each window and see if taken in order of appearance of text boxes/ inputs (normal data)	User must be able to tab through input boxes in order	To make the interface easy to use for the user	User can tab through each line edit and buttons in order of appearance
9	Enter button pressed to confirm certain inputs on screen (normal data)	User must be able to press enter button to confirm certain inputs	To make the interface easy to use for the user	Enter button does not work on all screens

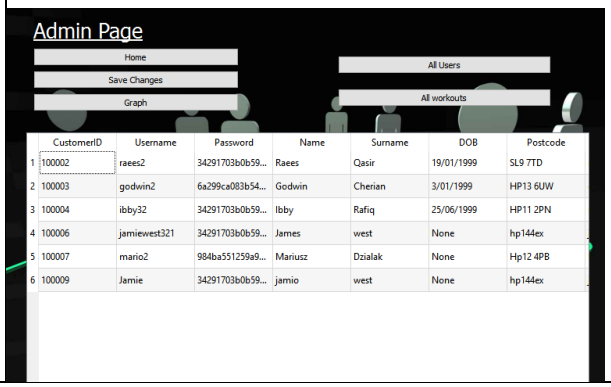
10	Click on a calendar date to select the workout (normal data)	Check this date is then saved to the database	To make sure the correct date gets saved for a workout	Date can be selected 
11	Check the weight and height number box increments when pressed (normal data)	An incremental box must be used for the weight and height input	To make it easy for the user to pick a decimal number	User can pick a decimal number easily 
12	Check the graph displays the last weight and the current weight of the user (normal data)	To make sure the user can see the difference easily and visually aid them using a graph	Make sure the graph displays the correct values and is the correct size on the window	
13	Check there is a label next to every single input box (normal data)	Labels must be shown next to each text box displaying what must be shown	Making sure the user knows what data to input into the input box	All input boxes have a label 
14	Go into each screen and check for label (normal	Labels at the top of	Each screen has a	 Each screen has a label

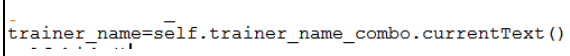
	data)	windows to say what the name of the current window is	label	
15	Go into each screen and check label is big enough font (ask Andrew) (normal data)	Labels must be large enough without the need of the user zooming in	Each screen's label font is large enough	„Not all labels have a large enough font" 
16	Go into each window and check each button is labelled (normal data)	Buttons must be labelled to say what they do when clicked	Each button is labelled correctly	Each button is labelled correctly 
17	Log in as admin and log in as a normal user (normal data)	There must only be 2 account types – admin (only one person has access to) and normal user	Log in as user and taken to welcome screen, log in as admin and taken to admin window	2 types of account – user and admin 
18	Log in using admin username and user's password (erroneous data)	Can only login as the type of user if the account is	Program rejects the input and display	The username and hash of the password is printed when the incorrect details are typed in 

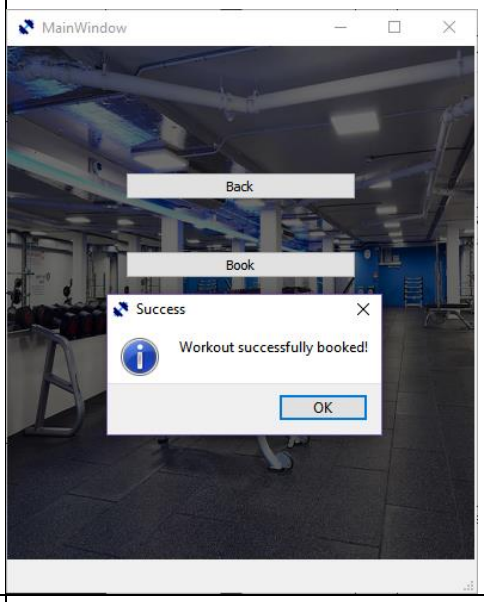
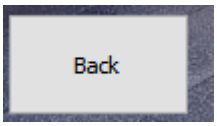
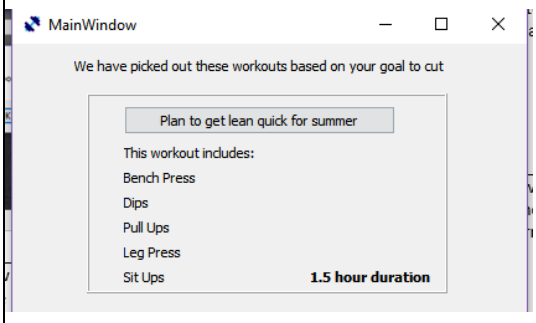
		assigned to that level i.e. admin details for admin login	s error messag e	
19	“Godwin” (normal data)	User must be able to register for a new account if they do not have one	Details accept ed – user can be register ed	Username accepted when registering for a new account
20	“godwin2” “league” (normal data)	User must be able to log into their account	User can login with the userna me and passwo rd	User can login using these credentials 
21	“godwin” “wind123” (erroneous data)	User can only log into their account using only their credenti als	Display error messag e “incorr ect details”	Cannot login – error message displayed 

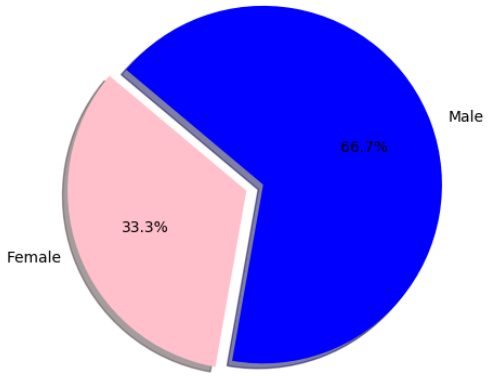
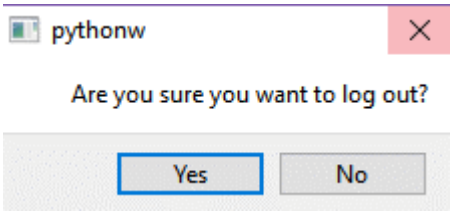
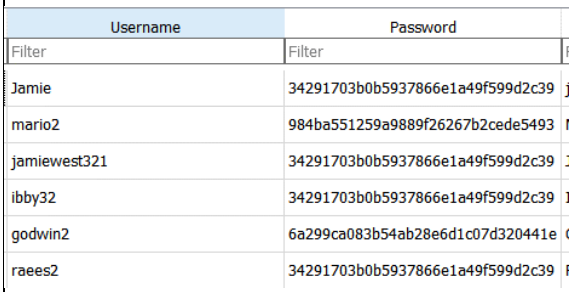
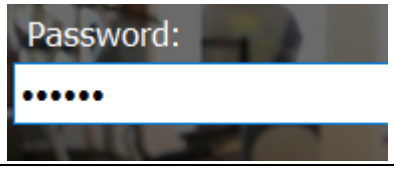
22	Check if data has been written to the database after user has registered (normal data)	Program must store all the details of the user given during registration	Data changed in database	Data written to database once user has registered their account  100011 test123 30c21386190...
23	Reset password using email and password of user (normal data) “godwin2” “godwincherian@gmail.com”	The user must be able to reset their password using username and email	Reset password reset sent to email	Reset password sent to email (printed to console for testing purposes) /// email sent
24	Reset password using email (which is not in the database) and username of different user (erroneous data) “godwin2” “o@gmail.com”	User cannot be sent an email unless they have registered it during the registration stage and cannot use another user’s username to do so	Email not sent, password not reset	Email not found and message not sent email not found 
25	Update personal user details (normal data) “Godwin” → „Alex”	User must be able to update their	User details updated to databa	Details updated in the database godwin2 517b08dafd8a... Alex Jones

	„Cherian” → „Jones”	personal details which will then be saved to the database	se	
26	Update personal user details (erroneous data) – updating name with just numbers “Godwin” →”12345”	The same validation from the registration screen should be taken and invalid data should not be updated	Error message displayed	Error message is displayed 
27	Change current password (normal data) “league” →”hi321”	To make sure this password is checked for strength and saved to the database	Updated password in database	Updates the password in the database 
28	Change current password (erroneous data) “league” → „^$£”	To make sure passwords with invalid data are not stored	Error message - password not accepted	 Error message is displayed

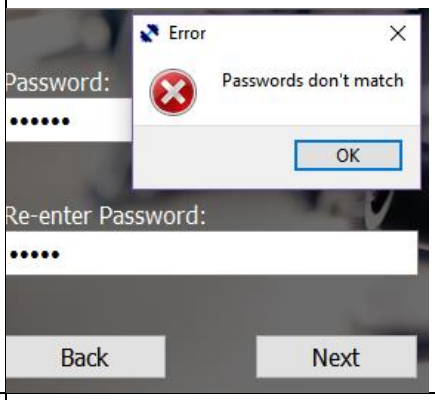
29	BMI button pressed (normal data)	The user must be able to calculate their Body Mass Index by clicking one button	BMI displayed on screen along with weight and height	All labels displayed 
30	BMI button pressed 10 times (extreme data)	The user must be able to press it many times without the program crashing	Label only displayed once – no crash	No crash – same result as above
31	Go into BMI calculator and check all labels are displayed	To make sure the user knows which values were used to do the calculations	All labels displayed in same font and size	All labels same size and colour 
32	Check the admin can view all the details of all the users in the database (normal data)	To make sure that any changes to these users can be made	Admin can view all user's details	Admin can view all the details 

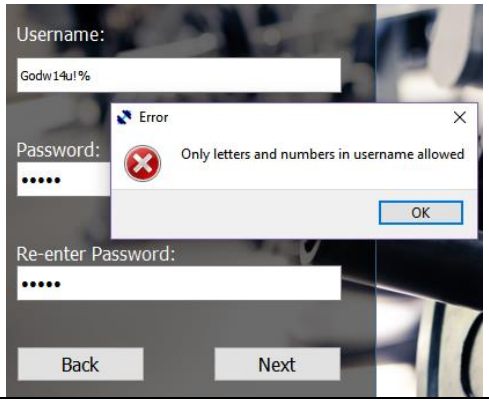
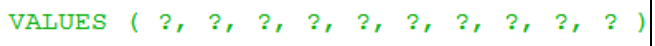
33	Check if table uses inner join function (normal data)	To make sure the inner join function works and the table is displayed for the admin	Inner join shown in table view	Shown in table 
34	Trainer picked from combo box (normal data)	To make sure a trainer can be picked for a workout	Stored as a variable	Stored as a variable when picked the name from the combo box 
35	No trainer picked from combo box (erroneous data)	To make sure the program rejects no input in the combo box	Program doesn't let user proceed	Program doesnt let user no pick a trainer. First trainer from the list is set as the default option. 
36	Check the payment is displayed as a label (normal data)	To make sure the user knows the total price and this also gets stored into the database	Payment is displayed as label	Payment is displayed as a label 

37	Click the book workout button (normal data)	To make sure all workouts which are booked are stored in the database	The workout is written to the database	Books the workout 
38	Click the book workout button 10 times (erroneous data)	To make sure bookings arent repeated many times	Program only books one session	Hides the current window when OK is pressed into the message box so cannot press the button 10 times
39	Check each window has a button to go back and next (if applicable)	To make sure the user can navigate through the program	Each window has back and next button	Each window has a back button 
40	Check workouts are for cut workout (normal data)	To make sure the workouts are picked for the goal e.g. Cut workouts for cut goal not build workouts for cut goal	Workout displayed for cut routine if goal is cut	The goal was the cut and the cut workouts were displayed 

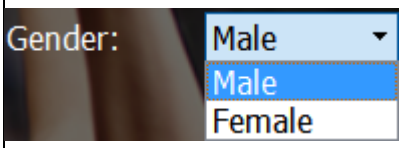
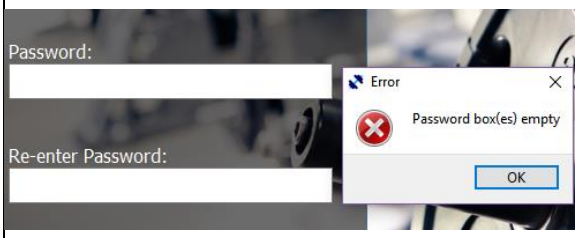
41	Click the pie chart button (normal data)	To make sure when a female or male is added the percentages change and therefore the pie chart	The male to female percentage is shown	Pie chart is displayed in a seperate window 
42	Log out button pressed (normal data)	User must be able to log out of the program	Once the button is pressed, the user is taken back to the login screen	User is taken back to login window once the yes button is clicked 
43	Check passwords in database and see if hashed (normal data)	Passwords must be hashed in the database to prevent a person logging into an account even with access to the database	Passwords hashed	Passwords in database are hashed 
44	Input password (normal data)	To make sure any input boxes which require	Password blacked out of view	Password blacked out of view 

Username	Password
Filter	Filter
Jamie	34291703b0b5937866e1a49f599d2c39
mario2	984ba551259a9889f26267b2cede5493
jamiewest321	34291703b0b5937866e1a49f599d2c39
ibby32	34291703b0b5937866e1a49f599d2c39
godwin2	6a299ca083b54ab28e6d1c07d320441e
raees2	34291703b0b5937866e1a49f599d2c39

		password input are always 'echoed out'		
45	Input password on register screen (normal data)	To make sure the passwords match when re-entering the password	Password should be stored at registration	Password is stored and written to database after a successful registration
46	„league" password „hi321" re-enter password (erroneous data)	To make sure the passwords have to match	Error message displayed and input rejected	Error message is displayed 
47	"godwin" entered into username box (normal data)	Usernames must be at least 5 characters long and can only contain letters and numbers	Accepts the username	Accepts the username godwin
48	"Godwin65" entered into username box (extreme data)	Make sure the program accepts capital letters, numbers	Username is accepted	Accepts the username Godwin65 as a valid username

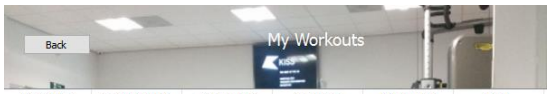
49	“Godw14u!%” entered into password box (erroneous data)	Make sure the program rejects any invalid characters for the username	Username rejected	Username is rejected 
50	Check all SQL queries contain question marks (normal data)	This is to prevent SQL injection	All SQL queries contain question marks	All SQL queries contain question marks to prevent injection 
51	“olek” entered into name box (normal data)	To make sure names only contain letters	Accepted as valid name	Accepts the name as valid
52	“Olek” entered into name box (extreme data)	To make sure capital letters are also accepted	Accepted as valid name	Accepts the name as valid
53	“124412” entered into name box (erroneous data)	To make sure the program rejects anything that is not letters	Rejected input – error message	Rejects the number input 
54	Input date of birth to make the age 17 (normal data)	To make sure the program accepts any age between 16 and 100	Accepts age as valid	Accepts the user and registers the account to the database
55	Input date of birth to make the age 101 (erroneous data)	To make sure the program rejects	Rejects this age - error	Still accepts the user and registers to database -fail

		any ages below 16 and above 100	message	
56	Type in valid email “odzialak@gmail.com” into email box (normal data)	The program should validate real email addresses	Accepts email as valid	Accepts the email as valid
57	Type in invalid email “olek35156@@g..co” (erroneous data)	The program should reject and invalid emails	Rejects email – error message	Accepts the email as valid - fail
58	Type in weight of user as 35kg (normal data)	To make sure the weight is between 30kg and 120kg	Accepts the weight as valid	Accepts the weight as 35kg and writes to database
59	Type in weight of user as 10kg (erroneous data)	To make sure the program rejects any weight below 30kg and any above 120kg	Rejects this weight – error message	This weight cannot be selected within the GUI. The minimum and maximum size restraints dont allow the user to input these erroneous values
60	Type in height of user as 120cm (normal data)	To make sure the height is between 100cm and 210cm	Accepts this height as valid	Accepts the height as valid
61	Type in height of user as 10cm (erroneous data)	To make sure the program rejects any	Rejects this height – error messag	This height cannot be selected within the GUI. The minimum and maximum size restraints dont allow the user to input these erroneous values

		height below 100cm and any above 210cm	e	
62	Check the only available gender options are male and female (normal data)	No other input into combo boxes is allowed	Only inputs into combo box are female and male	Only options ava ilable are male and female 
63	Input boxes which are required to have input checked if empty (erroneous data)	These input boxes cannot be left empty	Rejected – error message saying "cannot be empty"	Error message is displayed 
64	Choose date for workout – today's date (normal data)	A date must be picked for a workout	Program accepts date as valid	Program accepts today's date
65	Choose date for workout – yesterday's date (erroneous data)	A date cannot be picked before today	Rejects date – cannot be picked	Cannot click any dates before today's date 
66	Choose workout (normal data)	A workout must be picked to book a session	Program accepts workout	Program accepts this as a valid workout
67	Proceed without choosing a workout (erroneous data)	Cannot proceed without selecting a workout	An error should be displayed telling the user to pick a	Cannot proceed without workout – user wont be able to book workout

			workout	

68	Check an update query is used every time the user updates their details (normal data)	A copy cannot be made of the current record	Update query shown each time details updated	Update query is used when user updates details
				`cur.execute("""UPDATE Customers SET Username='%s',P`
				`WHERE CustomerID='%s'"""%(index_int1,`

69	Check the user can only view their workouts (normal data)	To make sure the user cannot see other user's workouts	Only user's workouts shown	User can only see their workout  WorkoutID ExercisePlanID CustomerID TrainerID DateBooked Price 1 1005 100003 1001 27/04/2017 £14.40 2 1008 100003 1001 29/04/2017 £18.00 3 1009 100003 1001 17/05/2017 £18.00

70	Check there are comments written next to all functions and loops (normal data)	To make the code maintainable for other users	Each loop has a comment next to it	Comments next to each loop, procudure, function, selection
				`ister_pg2.ui")[0]  # Load the register page 2 UI`
				`[0]  # Load the login UI`
				`ui")[0]  # Load the homepage UI`
				`ister_pg1.ui")[0]  # Load the register page 1 UI`
				`orkout_inner.ui")[0] #Load the workouts inner page UI`
				`i")[0] #Load the admin page UI`
				`Load the BMI UI`
				`Update.ui")[0] #Load the user update UI`
				`asswordChange.ui")[0] #Load the password changeUI`
				`Load the BMI UI`
				`[0] #Load the build routine UI`
				`orm.ui")[0] #Load the transform routine UI`
				`lscreen.ui")[0] #Load the final screen UI`
				`asswordReset1.ui")[0] #Load the password reset UI`
				`kouts.ui")[0] #Load the my workouts UI`

71	Check all variable names (normal data)	To make the code maintainable for other users	Each variable has a meaningful name	All variables have meaningful names
				`NAME_str=each[0]`
				`SURNAME_str=each[1]`
				`POSTCODE_str=each[2]`
				`EMAIL_str=each[3]`
				`global WEIGHT_float`
				`global HEIGHT_float`
				`WEIGHT_float=each[4]`
				`HEIGHT_float=each[5]`
				`GENDER_bool=each[6]`

Usability

- Every window will have a back button to go back to the previous window (does not apply to windows where you cannot go back e.g. start screen)
    - Every window has a back button apart from the login screen – fully successful
- Buttons must big enough to click easily for navigation and for confirmation where required
    - Not all buttons were large enough with big enough font for the user – partial failure – test no. 24 where in the reset password window the buttons are too small – I would make the window bigger as well as the buttons
- All windows must have pictures relating to the gym to make for an easy to see interface
    - Not all windows had pictures relating to gym, but all had pictures which linked in with the overall theme e.g. password reset had a picture of the internet – partial failure – test no. 3 where the window has no picture at all – with more time I would add a picture to each window
- The user must never have to scroll down within windows
    - The user never had to scroll within any of the windows – fully successful
- See the list of workouts for their goal easily and can select them with a button
    - The user could see the goal and the exercises were easily readable for them which they could select with a button – fully successful
- Text labels in bold where the user can register a new account or reset their password
    - The text labels were in bold for register and reset – fully successful
- A custom gym weights icon for each window
    - Each window had the icon of a weight – fully successful
- Being able to tab through text boxes instead of clicking each one at a time
    - User could tab through each text box instead of clicking – fully successful
- Enter button (only where there are 'confirm buttons') to take user to the next screen
    - Enter button only on some screens to confirm the input – partial failure – test no.9 – if I had more time I would link the confirmation buttons to the enter key
- Combo boxes where there is a finite amount of choices (e.g. for goal a combo box for 3 options to "cut", "build, or "transform")
    - Combo boxes for all finite amount of choices, user said that this is an optimal amount of combo boxes – fully successful
- A calendar for the user to select the date of their workout
    - A fully functional calendar which lets the user select the date of their workout – fully successful
- An incremental number box for weight and height with a certain number of decimal places (e.g. 82.5kg for weight)
    - The number box is a good addition according the user and you can increment through easily – fully successful
- A graph to show differences between weight so the user can see easier
    - The graph displays the difference between previous and current height, however it won't display anything until there is a previous weight – partial failure – with more time I would have the graph display the current weight only if no previous weight is found – I would also not use a list but keep the integers as 2 separate integer variables to make this possible since the array inside the HTML graph requires 2 integers, but a list can also be added.
- Labels next to each text box displaying what needs to be input
    - Each text box has a corresponding label next to it – fully successful

- Labels at the top of windows to say what the name of the current window is
    - Each window is labelled with the correct name – fully successful
- Large enough size font to make sure the user can read the labels without zooming in
    - User comment – "I couldn't read all the labels properly as they were too small" – partial failure – test no.28 where the labels are very small for the user – with more time I would increase their size and increase the size of the line edits to match them
- Buttons labelled to say what they do when clicked e.g. proceed will take user to next screen whereas update labelled button will update information
    - Each button labelled correctly according to what it does – fully successful

<u>Performance</u>

- There must 2 account types – an admin type (only one user can have this type of account), and a user account (all other users)
    - Two different functioning account types – fully successful
- The user must able to register for a new account if they do not have one
    - User can register for a new account which gets written to the database – fully successful
- The user must able to log in with this new account after registering using the username and password
    - User can log in using the created username and password after registration – fully successful
- The program must store all the details of the user given during registration
    - The program writes all the details to the database – fully successful
- The user must able to reset their password using their username and email
    - User can reset their password – fully successful
- An email must be sent to the user with the reset password with which they can log in
    - Email is sent to user with the updated password – fully successful
- The user must able to update their personal details which will then be saved to the database
    - User can update their details which then get overwritten in the database – fully successful
- The user must able to change their current password and save this to the database
    - User can update their current password and this gets written to the database – fully successful
- The user must able to calculate their Body Mass Index by clicking one button
    - The BMI is calculated by pressing one button – fully successful
- The BMI calculator must display the weight and height used to calculate the BMI, the BMI value, and what status of health (e.g. overweight) depending on the value.
    - A label is displayed showing which status of health a user is – fully successful
- A bar graph must be displayed showing the previous weight of the user found in the database (none if previous weight has never been stored) and the current weight
    - Bar graph only displayed if previous weight is in the database otherwise nothing is displayed – partial failure - with more time I would have the graph display the current weight only if no previous weight is found – I would also not use a list but keep the integers as 2 separate integer variables to make this possible since the array inside the HTML graph requires 2 integers, but a list can also be added.

- The admin must able to view details of all the users in the database
    - The admin can view all the details of all the users – fully successful
- The admin must able to update, delete, add and create new users in the database
    - The admin is only able to update the users – not met – if I had more time I would make sure that I don't go through each index of the database one by one and make sure that only particular records can be filtered for only so that only that record can be updated individually. I would also add a line edit which allows the admin to filter through the data which would become more important with increasing data sets. I would also add a button to add users directly into the database from the admin window
- The admin must be able to show all workouts with a name joined from the users table
    - Admin can view the workouts with a name assigned to each workout – fully successful
- The user must be able to select a trainer from the database in a combo box
    - The user can pick the name of the trainer for their workout – fully successful
- The user must be able to select a date from a calendar for their given workout and this input be stored in the database when the workout is booked
    - The user can select the date for the workout from the calendar which is then stored in the database – fully successful
- The total must be calculated for the workout using the payment per hour of the trainer multiplied by the duration of the workout and displayed to the user
    - The total is calculated which is then displayed to the user – fully successful
- The user must be able to book their workout and this be stored in the database
    - The user can book their workout and this is stored in the database – fully successful
- The user must be able to switch between windows using buttons as a way of navigation
    - The user can switch between windows using the buttons – fully successful
- Workouts must be displayed depending on the goal of the user
    - Workouts are displayed depending on the goal of the user – fully successful
- A pie chart for the admin must be displayed when a button is clicked to show the percentage of male to female users.
    - Pie chart displayed when button is clicked and shows the amount of female to male users in the database – fully successful
- The program must store the previous weight of the user if a new weight has been updated to
    - The program stores the previous weight of the user when the weight is updated – fully successful
- The user must be able to log out of the program once finished
    - The user can log out and all the text boxes are cleared upon logging out – fully successful

<u>Reliability</u>

- The program must hash passwords into the database so that even with access to the database, that person will not be able to access a given account
    - The program hashes passwords – fully successful
- If a text box requires password input, this must always be dotted out so that it cannot be seen

    - All password inputs are ‘echoed out’ – fully successful
- Users may only be able to log in with their password and not any other user’s password from the database
    - Users can login using only their details – fully successful
- When registering, passwords must match for the user to continue
    - When the user registers, the password and re-enter password matches and the user can continue onto the next window – fully successful
- Usernames must be at least 5 characters long and can only contain letters and numbers
    - Usernames must be at least 5 characters long and the user may only continue if they are letters or numbers in the input – fully successful
- When executing SQL queries, question marks must be used to prevent injections by any users
    - Question marks are used in SQL queries – fully successful
- A user’s name may only contain letters
    - Names can only contain letters – fully successful
- A user’s surname may only contain letters
    - Surname can only contain letters – fully successful
- The age of a user cannot be below 16 or above 100
    - The program won’t check the age of the user – not met – if I had more time I would make sure the age is calculated at registration and instead of using the date input in PyQt I would have a calendar widget for the user to pick their date of birth which could then be formatted more easily into a variable which then gets used to calculate the age of the user.
- The email of a user must be valid by sending an email and seeing if there is a response
    - Email not validated by sending the email a message – not met – with more time I would have sent a test email and return True or False if the email exists.
- Weight of the user can only be between 30 and 120kg
    - Weight of the user can only be between 30kg and 120kg, any other number won’t be able to be picked– fully successful
- Height of the user can only be between 100cm and 210cm
    - Program checks the height is between 100cm and 210cm and won’t allow any other input – fully successful
- Only available genders are male and female and nothing can be written to combo boxes
    - Nothing can be written to combo boxes and only available genders are male and female – fully successful
- Text boxes which are required cannot be left empty
    - If text boxes which need input are left empty, an error message appears – fully successful
- When booking a workout, a user cannot select a date prior to the current day i.e. cannot book a session for yesterday or any other days before today
    - User can only selected today’s date or anything after – nothing before – fully successful
- A workout must be chosen for the program to proceed, otherwise the user will not be able to book a session
    - User cannot book a session without choosing a workout – fully successful
- When updating user records, a copy cannot be made of the current record, but instead an update query must be used

    - Whenever there is user updating details, an update query is used – fully successful
- When a user views their workouts, they cannot see the workouts of other users
    - User can only see their booked workouts – fully successful

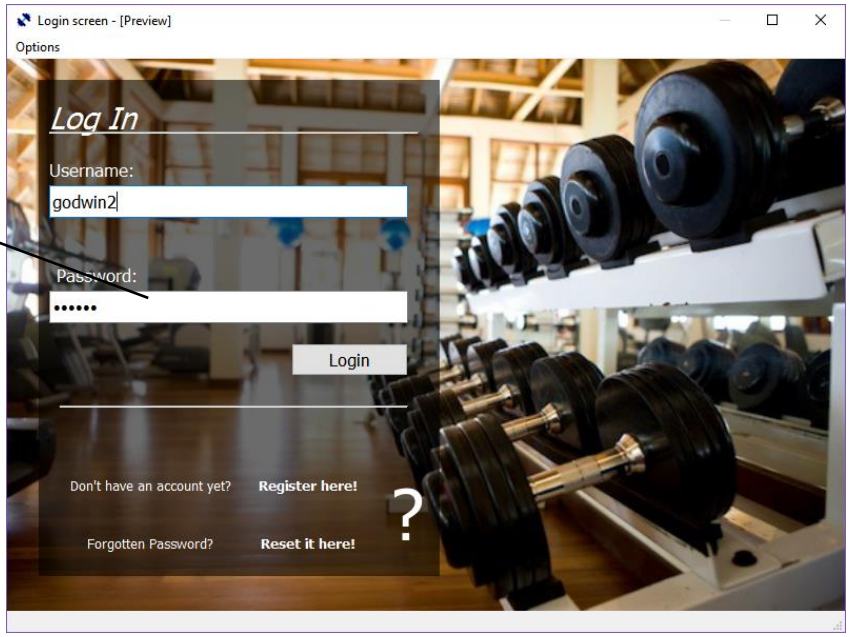
Maintainability

- Comments should be written next to lines of code to make sure it can be understood by other programmers.
    - There are comments next to each function, procedure, loop, class – fully successful
- Meaningful variable names should be given
    - All variables have meaningful variable names – fully successful
- Should be able to add new users easily as admin or be registering for a new account
    - Cannot add new users as admin but can register for new account – partial failure - with more time I would add a button which opens a window and lets the admin register a new user essentially.
- Same formatting to code throughout the program
    - The formatting to code is the same throughout – Hungarian notation – fully successful
- Testing to make sure the program does not crash with extensive use
    - Tested throughout development and ran over 100 times – does not crash unless error in coding – fully successful

Usability feature's effectiveness

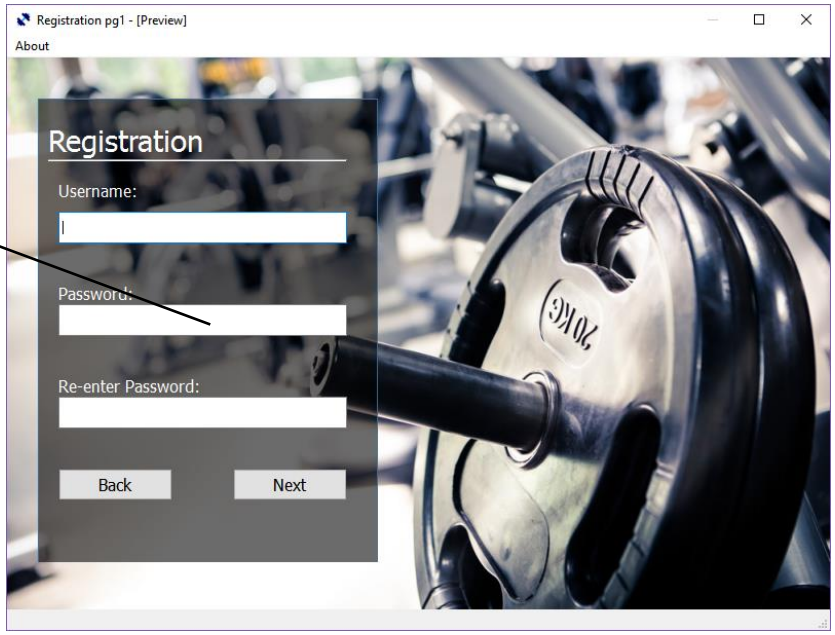
Login window

There are certain aspects about the interfaces which I would change given more time. In the login screen, I could change the font size to make it larger and the font style. The password box could also be a red colour if the user entered an incorrect password. I could also remake the login button so that if the username and password are empty, the box is 'grey', but if there is input in both boxes – even if wrong, the login button would turn an orange colour, to signify that the details are ready for login. I could also add an animated picture to the background for the login screen only (gif could also work)



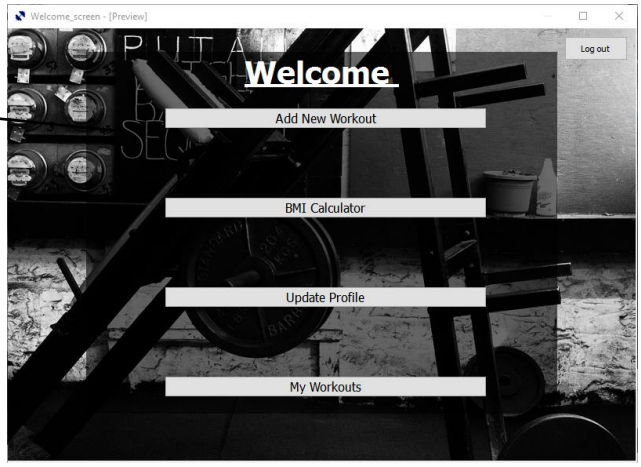
Register window

For the register window, I could add the validation colours e.g. red if password doesn't meet criteria. Also, I could have a tick next to each box and if the passwords, max the tick would change colour from grey to green. Also, I would have liked to combine the two register windows into one single window as having two windows for the registration was something that Andrew said he didn't like and would like changed in the future.



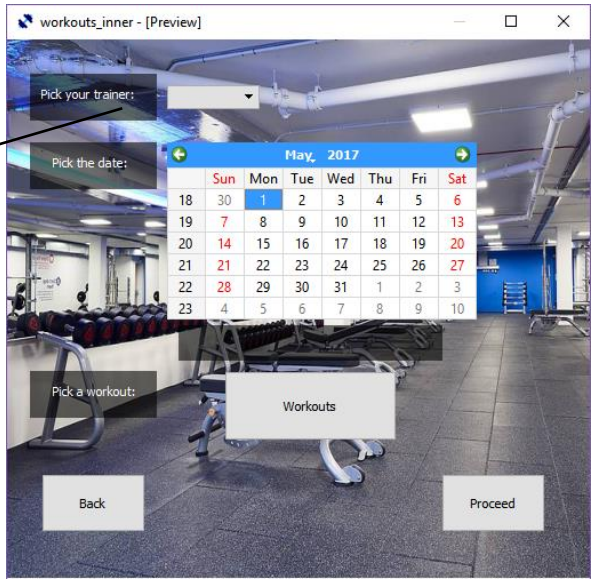
Welcome window

For the welcome window, I could have created a different style of buttons – larger more square buttons to make them easier to read. I also would change the picture as it doesn't match the background transparency. Also, the log out button could be assigned to a picture instead of a button.



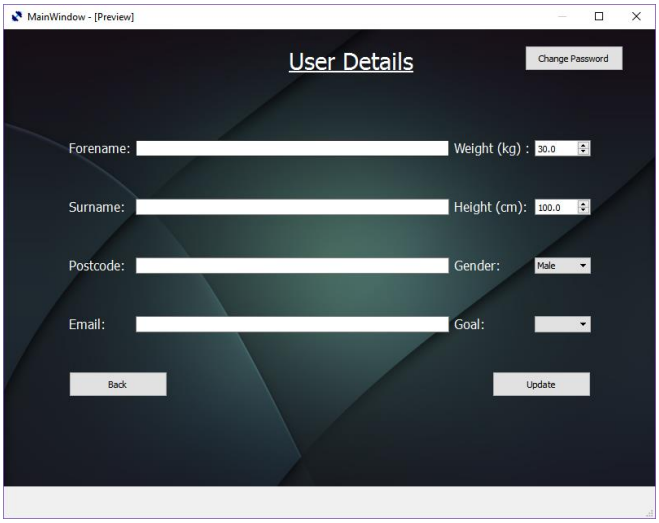
Workouts window

For the workouts window, I wanted a bright background picture; however I then had to do a black box for all the labels which was not effective. If I had more time I would have changed the picture or added the black box around the widgets instead. Also, I wanted the workouts as list of exercises and not a separate window but I also didn't want the workouts window to be big. I had to create a separate bookings window, but having one window for the workouts and to book it would have been more effective as the bookings window is not necessarily needed. I had to create it to make the program work, but someone who knows how to make the booking work in this window could develop to just have one window



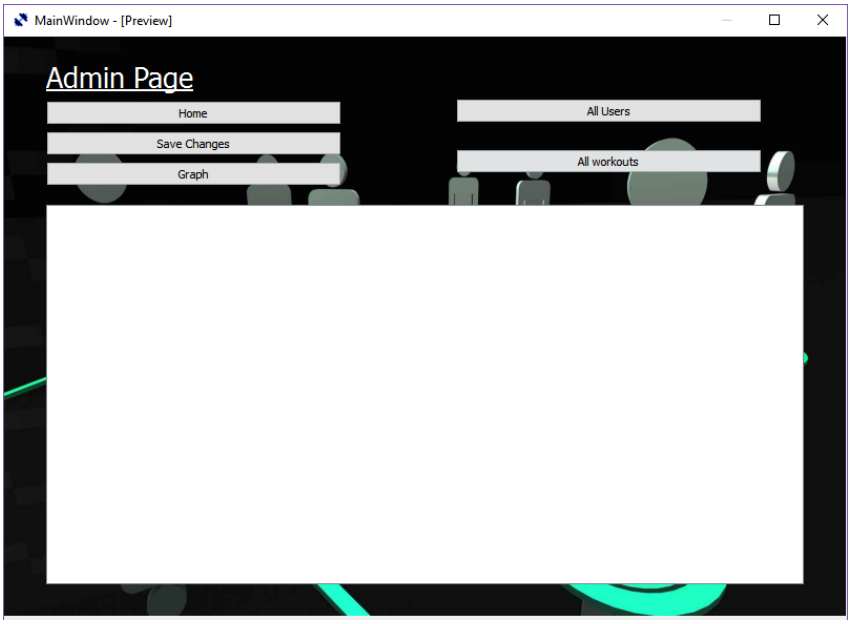
User update window

For the users update window, I wanted to be able to change the password in the same window, but again I was restricted into making another separate window for the password update. If I had more time I would try to combine the two windows where the user can update all the details at once.



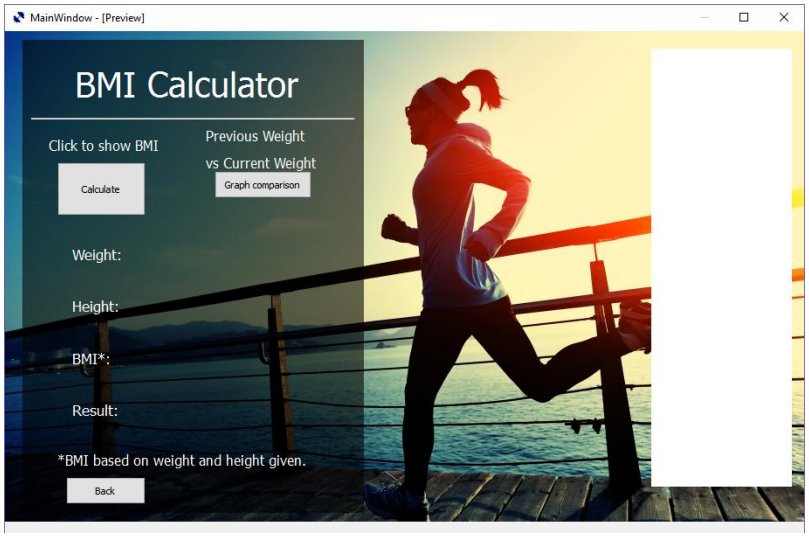
Admin window

For the admin window, I would have changed certain elements. For example, the graph that is displayed is shown in a separate window. This is due to the module imported to create this pie chart. If I could create the pie chart in HTML I could have a web view and display the pie chart in this web view instead of having another window. This would also remove the button which was an issue in my success criteria. Also, I would have a filter for the users so that when data sets become too big the admin can filter for a specific user



BMI window

For the BMI window, I would change the web view of the graph. I would make the window larger and have it separate to the picture where it doesn't block the view of this picture. Since there are 2 buttons that must be pressed to display the graph, I would remove at least 1 of them in the future so with 1 button, the user can compare their previous weight and their current weight



Usability features conclusion

For the usability features, I would want an overall re-design of the buttons. Since Andrew requested less buttons overall, it would be a great idea to have less windows to reduce the amount of navigation buttons i.e. back button, next button. I would also change some of the pictures, and create an overall theme running throughout the program through this. I would also try remove long horizontal line edits as the input usually doesn't reach the end anyway i.e. names average 6 letters in length whereas the line edit accepts around 40.

Maintainability of the solution

My program could be changed in the future. A trainer login could be created and these trainers would be able to add their own workouts. These workouts could then be picked by the users. Also, the trainers could be notified if a user picks them as a trainer via email or phone. They could receive the time, routine and person who will be coming to train with them. For the admin, the adding users could be added as a feature to register new users through the admin. Also, the admin could have a pie chart display the most popular trainers. Furthermore, the number of users could increase. This would mean the admin must look through more users and therefore certain criteria could fail e.g. never having to scroll. A feature that could present a problem is the password reset. If the user has no internet connection, they may not be able to reset their password if the reset password is not printed onto the console. Also, if new features were added, the program could be slower loading up initially. The more modules are imported to compensate for the new classes, the longer it would take for the windows to load up as well as the start of the program. The current solution takes around 4 seconds to load the first window. It first must import the modules and calculate certain variables such as the current time. With increasing data sets, it would take longer for the user to show their current workout, slowing the overall program. Another problem is the limitations of the language. Python doesn't have unlimited functionality and other programming languages may be better suited to further develop the solution. Eventually, the database may need to be backed up – whether this is to another file or as an online backup so that the database can be accessed anywhere. Also, the custom workouts would have to written into another table and the database would have to be changed for this.

Big O complexity

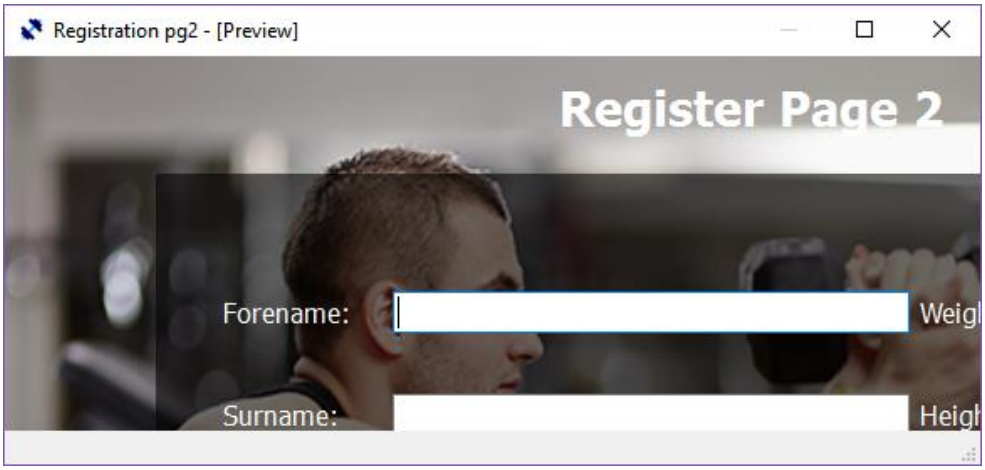
My program has a linear complexity O (n) for the number of users and how long it takes to find an element using linear search. Therefore, when the number of users increases, the time it takes to search for a user will increase. On the other hand, for a binary search of the users, the complexity would be O (log n) whereas the size of the data set doubles, the number of times to be checked only increases by one (due to the nature of binary search and halving the data set). If there were to be custom workouts in the program the program would have a linear complexity O(n) as the number of workouts increases and the time taken to search through them.

Maintenance issues

Even with comments next to each line of code it is hard to have maintainable code since there are thousands of lines a programmer may eventually go through. Therefore, my program would benefit from having more, smaller sized modules being imported so that each module could just have one comment explaining what it does. Also, there is repetition in my code which is not needed which can again be solved using modules. With Hungarian notation, it is easier for the programmer to recognise what the data type is. Therefore, this makes the code more maintainable for someone who may not know the data type straightaway.

Limitations of the solution

The limitations to my solution are that a trainer must be picked for a workout. This is an assumption I've made from the start however, in the future there could just be the person training by themselves without the trainer, but they still need to keep a track of their workouts. Also, the windows in my program are not scalable. They are fixed resolution but they could be developed so that as the window decreases so do the widgets inside it and when the window increases so do the widgets:



Therefore, any computer or system running lower resolution than 900x800 cannot benefit from my solution. Also, the scalability from different operating systems is visible. When developed in Windows 10 initially, and then launched on Windows 7 the formatting of all the widgets would be lost. This must be considered as not everyone will be running Windows 10. Another limitation is that all the exercises must be written by the admin or the trainer. Eventually, there could be a solution where all the exercises are read from a website and then imported into the program

Portability

To make this program portable to different platforms, the resolution of the windows would have to be changed, along with the scalability of the windows (as mentioned above). All the widgets would need to be resized and the buttons would have to be suited to the given OS of the mobile platform. The icon would be left the same and could be used as the main icon for the application. To make this into a mobile app, it would have to be programmed in a different language e.g. for iPhone Objective-C and using XCode as an IDE to make sure the app can be used on an iPhone. Also, the database would have to exist on a server for a mobile solution as it cannot be stored on the user's phone. For a voice interface, such as Siri, it would be quite difficult to make the solution work. This is because of the amount of options and features included in the program. It would be difficult to keep saying voice commands to navigate through windows. Therefore, it would be easier to just have one window where everything happens. This could scale back on the features, but instead could be used with other widgets such as combo boxes instead.

Further development

If I had more time, I would make sure the age of the user is calculated. This is not as easy as it seems at first. Even though to calculate the age of the user is simple using the formula, having to constantly calculate it to make sure the user doesn't exceed the limit may be harder. Also, there could be discounts for workouts if a certain trainer keeps being picked. For example, if a trainer has been picked 5 times in a row, then the 6^{th} workout is at half price. Another feature which could be added with more time is seeing the gyms in the local area and signing up for membership. This could be an additional side program which lets the user register for a new membership at a certain gym. Then they could see the gyms within their IP address radius e.g. 2 miles and see if there are any gyms. Another feature which could be added is more calculators such as a fat % calculator. This would calculate the percentage body fat of a person and display it to them. With further development, the admin window could be developed further into making sure it can add new users. Also, the admin could be able to set specific restrictions on accounts e.g. if an account is spamming bookings for a certain date, then they may get a temporary suspension which doesn't let them log into the program