ROHIN HEER

OCR GCE A-LEVEL COMPUTER SCIENCE PROGRAMMING PROJECT

Name: Rohin Heer

Candidate Number: 5616

Centre Number: 12520

Project Name: Daycase Patients Overnight Stay Audit

Table of Contents

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

A-Level Computer Science Programming Project

Daycase patients Overnight stay Audit

Step 1 : Analysis

Introduction

For my project, I will create a day-case database for overnight stays at The London Clinic. My client has asked me to create an audit for the number of day-case patients that have stayed overnight each month. I have also been asked to tally the different reasons for staying overnight, take note of the consultant the patient is under and the ward/room transferred to. This data should all be put into a bar chart at the end of each month to show reasons of transfer against number of transfers. Each bar will represent a consultant and this will highlight any trends with certain consultants to prevent day-case patients blocking impatient beds.

How is it computationally solvable and the current system?

I think my program is computationally solvable as it is the most efficient method to gathering this data as it allows for a much easier way of recording the statistics of overnight day-case patients. The system my client used to use was very longwinded as they have to individually type in all the necessary details (e.g.; the patient's consultant, room number (usually 300-316), date, time, reason for staying, ward/room transferred to) for each patient and then manually count the total number for each reason of overnight stay, that had been recorded throughout the month. Doing this, mistakes can easily be made as there a lot of repetitive data and these mistakes can cause the wrong conclusion from the trend of the results which gives false information to the Senior Management Team. Therefore, in my new solution, creating a database using SQLite in Python will automatically store all the information from these features then calculate the totals of the different reasons of transfer at the end of each month with the consultants relevant to the reason, (this is presented in a graph) saving time and effort for the nurses who have other needs to tend to. The data on the existing system is not very secure so I will add a login system to prevent unauthorised access to sensitive information. Using computational methods will allow me to configure a better solution than the current procedure and enable my client to present more accurate and professional data to their senior management team.

Current system

Transfers to other Floors from DSW
June 2022

Room No	Date of transfer	Transferred To Floor Name	Consultant Name	Reason for Transfer
			Prof Saleh	GA Discharge after 2130
			Mr El-Amir	GA late discharge after2130
			Mr Wareing	Overnight stay late theatre GA
			Prof Jackson	Emergency admission for theatre @ 2000, late dischargeSWS
			Mr Antscherl	Late discharge, GA, theatre @1900
			Mr Crinnion	GA O/N stay post-op drain
			Mr Westcott	Late theatre @ 20:30 late discharge
			Miss Zakir	Late theatre finish @ 21:15
			Mr Mitry	Patient transferred post-op O/N stay
			Mr Pringle	Late discharge Emergency admission
			Mr El-Amir	O/N stay planned

This image of the current system is a table in word document that my client uses to enter the room number, date of transfer, floor transferred too, the consultants name and reason for transfer. I will be using similar fields in the databases of my own.

Computational Methods	How will this be used?
Thinking Abstractly	I will use abstraction to remove unnecessary information from the audit as other nurses and my client do not need to know why a patient is with that specific consultant and the patient's personal details such as gender/age. The users of my software will also not need to know how long the patient is staying for or data of discharge, therefore this information will not be included. For the patient, I will only include their admission number which is their room number and ward number so my client can keep track on where the patient has moved in the clinic. I will also have their reason for transfer displayed so the user can see why they are staying at the clinic.
Thinking Ahead	For this program, considering the pre-conditions, I would need to prevent the database being altered by two people at the same time and prevent deadlocking so I will have to isolate the data (ACID) and I shall do this by making the database read-only if another user tries to make changes to the database and nurses who shouldn't be able to access it won't have an admin login so they can't interfere with existing data. This admin login I will give to the only users who should be able to look at this data. The program will only exist on one PC in the head nurse's office where it can only be accessed by whoever is in charge of the ward that day. Thinking ahead, libraries will need to be used such as AppJar to provide a simple script language for the user interface and SQL lite, which allows the use of SQL commands to build my database. This saves time and makes my code my reliable as the library code is pre-tested. I will need to consider the consultants information to know who's

	responsible for a particular patient and any comments the user wants to make for the reason of overnight stay such as time of discharge or to account for any altercations. I will do this by creating another column in my database for any additional comments and reviewing The London Clinic's website for information about the Consultants as well as discussing with my client about their desired list of consultants. Furthermore, every month I should meet with my client to update her about my program and receive feedback to see what I could improve to meet her requirements accurately.
Thinking Procedurally	To create this audit, I will need to split this project into four main modules. 1. MainProgram 2. Plot Graphs 3. SQL database 4. AppJar – login page Each module should be broken down into subroutines of public classes by using OOP to link these programs together and call the functions from 2,3 and 4 into the Main Program. The program will create a graph at the end on the month to precisely represent the consultant and overnight stays. This will help the user view the data quickly and efficiently, so they are able to get back to patients on the ward as soon as possible. In addition, I will need to think procedurally when I am validating the database as all the fields name to be correct and any data that is entered by the user incorrectly shouldn't be considered. The first page that pops up should be the login page so the valid nurse is able to access the data securely.
Thinking logically	I will be thinking logically by addressing the jobs in order of importunacy (hierarchy charts) such as building the database and creating unique functions for retrieval of data for each month before the design of the graph as well as any buttons for extra features. I will also make a flow chart to view the possible input and outputs so I can plan how I code methodically using my different

	features. These features will also determine on my client's opinion, and I will adjust any of them to suit the teams and any of her needs at any time. I am going to be thinking logically when working on the login system as the user will only get 3 tries to login with the correct details otherwise the program will close.
Thinking concurrently	Every day at a certain time the daily database needs to move all the information to the current database. Thinking concurrently, the database will save for that day. I will check this by finding out if the daily database refreshes after 8pm and if the previous days database is saved in the current database. This also makes every days data easily viewable to the user. On the last day of the month at around 8pm the title of eg, "Jan current database" will change to eg, "January's data". In addition, at the end of each month I need to make sure the daily database refreshes and a new "current month" database to be made ready for the next month, eg "February" and the previous one saves under the correct month, eg "January". To ensure this happens, I will check if the previous months data saves under the correct name and two new database is created for fresh inputs for the following month. In addition, at the end of each month the current database data needs to be retrieved into the graph creating the summary table too. I will do this by using a function eg "retrive01data()" in the graph tab. Once created, I shall also check if the graph is formed in the correct format. Furthermore, a new "view graph summary" button should be made each time for each month and I will check if a new one is displayed.

Stakeholder

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

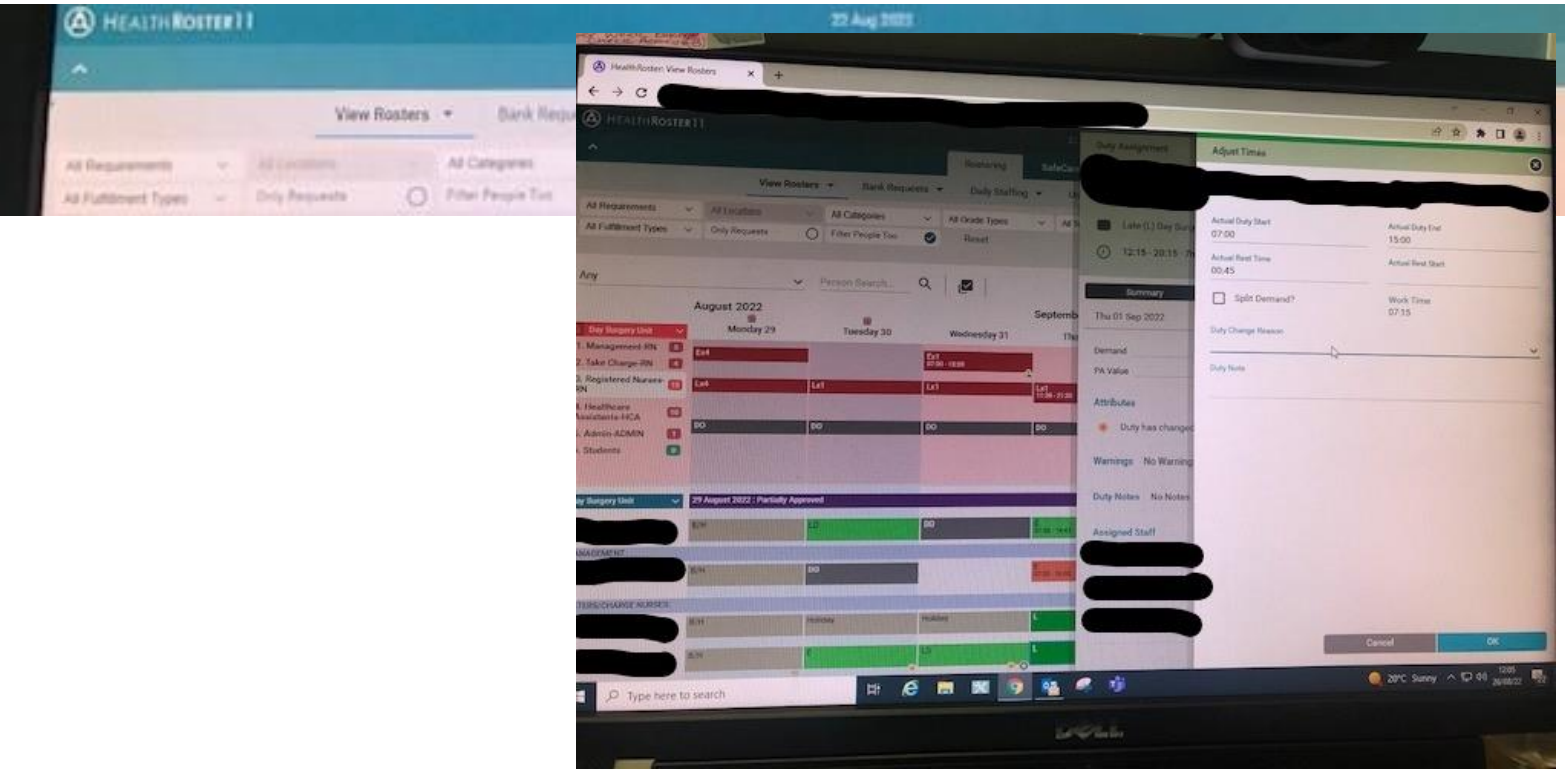
My client is my mother who works at The London Clinic and a ward manager. She wants this product to help identify if the Daycase unit opening hours needs to be extended to a later time as this is what the senior management has recommended to her, and identify which consultants need to be spoken to if they are not following policies or procedures. This is appropriate for my client's needs as it will save admin time for the senior team in unit to correlate data. Moreover, this will allow more time and better quality of care for patients. My client's end user is the senior nursing team as they provide more detailed data for reasons of an overnight stay. These other nurses on the ward using the system will be my stakeholders too and with this in mind, I will be gathering their requests/adjustments and implementing them into my program. This will be done by me asking them questions and writing up their suggestions into a table.

Existing software

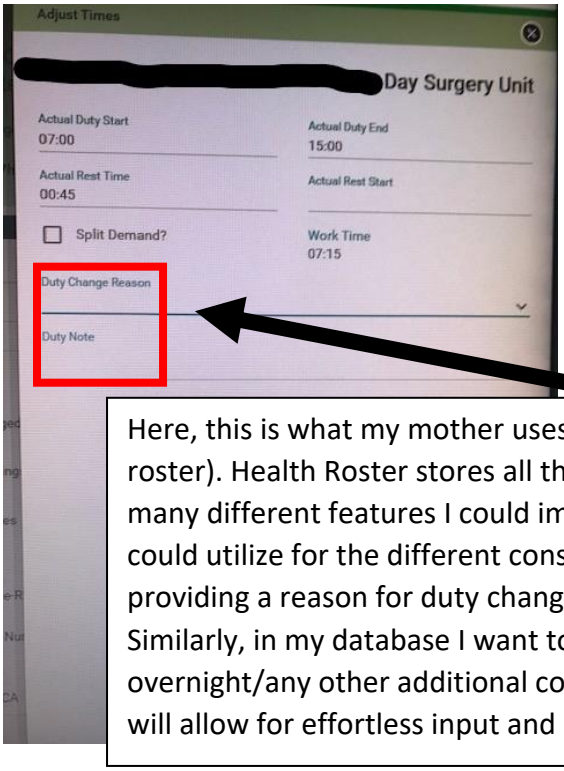
To get some ideas and inspiration I have been looking at other types of software which has a similar purpose to mine. These include features I could add to my software to make it more efficient and user-friendly.

Example 1:

Health Roster – The London Clinic



Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**



Here, this is what my mother uses to manage/change the rotas for her staff (MEDITECH – health roster). Health Roster stores all the hospital's staff shifts and information. This User interface contains many different features I could implement into my audit as it has many different categories which I could utilize for the different consultants. This software allows for changes to the database by providing a reason for duty change as well as accepting any specific notes towards that change. Similarly, in my database I want to make it quick and easy to select why the patient is staying overnight/any other additional comments about the change, therefore bringing forward this feature will allow for effortless input and bring forth a simple look to the software.

Example 2:

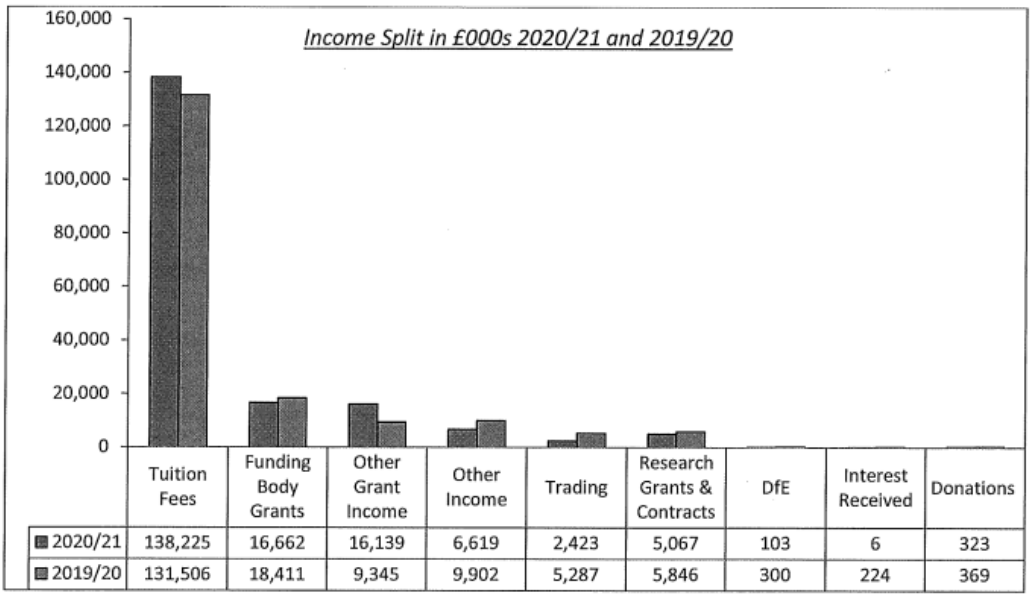
University of Wolverhampton:

Statutory Accounts

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

market and Post graduate research student numbers also fell by 4.8%. However, this reduction was compensated by a significant 25.1% increase in Postgraduate taught student recruitment.

The Covid 19 pandemic continues to impact on international numbers with international numbers down on 2019/20 by 47.8%.



	Tuition Fees	Funding Body Grants	Other Grant Income	Other Income	Trading	Research Grants & Contracts	DfE	Interest Received	Donations
2020/21	138,225	16,662	16,139	6,619	2,423	5,067	103	6	323
2019/20	131,506	18,411	9,345	9,902	5,287	5,846	300	224	369

Overall funding body grants were down by 9%.

Research Grants and Contracts fell by 13%. Other Income decreased by 40% compared to the previous year and reflects the loss of trading income due to COVID-19.

This is the statutory accounts report for the University of Wolverhampton of 2021. Presented here is an audit for the income split and it is represented in a graph. In my project, at the end of each month, I would like my data to be represented clearly and concisely. Taking on board this idea of a bar graph will allow this to happen and it will make it much easier for the busy staff to read. Something that this report doesn't have Is colour, but I would like to incorporate basic colours into my graph, so it is obvious what each bar represents and so the user doesn't get confused with all the information Infront of them.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

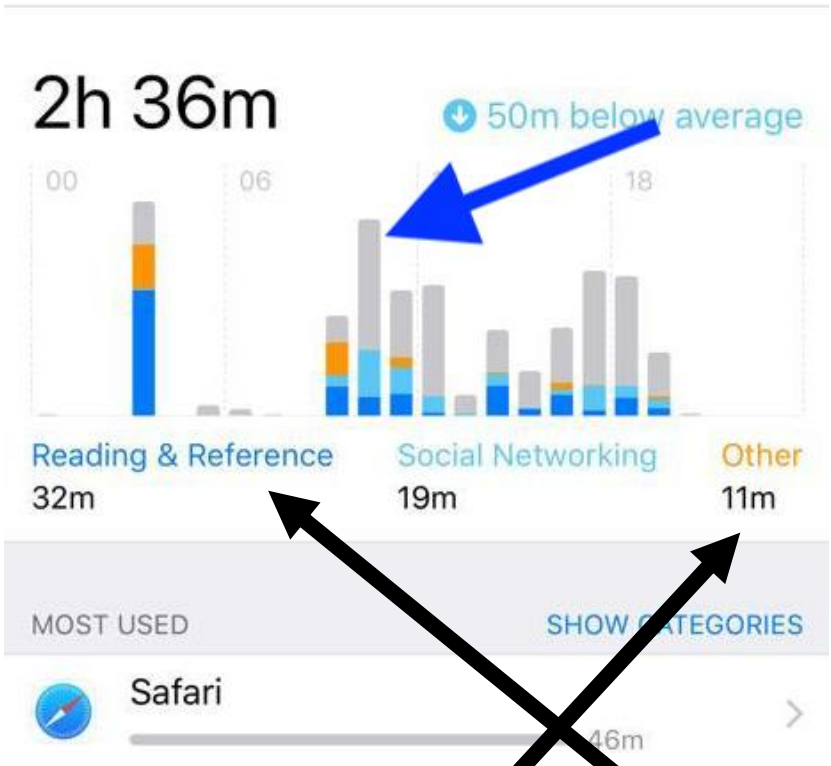
	2020-21			2019-20		
	Operating Activities £000s	LGPS Adjustments £000s	Total £000s	Operating Activities £000s	LGPS Adjustments £000s	Total £000s
Total Income	185,567	0	185,567	181,190	0	181,190
Total Expenditure	(181,018)	(8,609)	(189,627)	(177,937)	(7,907)	(185,844)
Taxation	0	0	0	0	0	0
Surplus/(Loss) for the Year	**4,549**	**(8,609)**	**(4,060)**	**3,253**	**(7,907)**	**(4,654)**

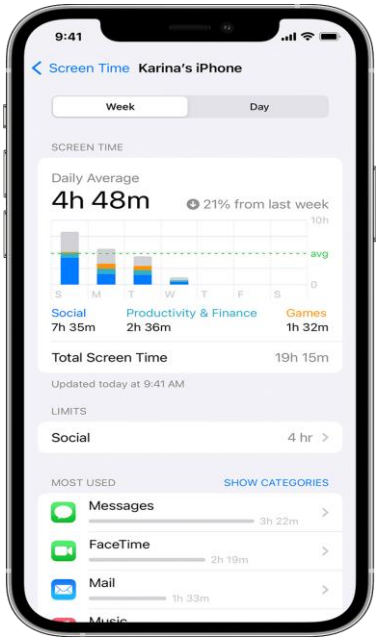
This is the graph summary of the statutory account report of the university for 2021. It presents the key variables in the graph and their assigned values in a table format. Similarly, in my project the key data/trends from my graph will also be presented in a table form so nurses can quickly view key information in word format. This acts as another representation of data so the user can have the option of viewing either one, increasing the versatility of my program for all nurses.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Example 3:

iPhone screen time activity tracker:



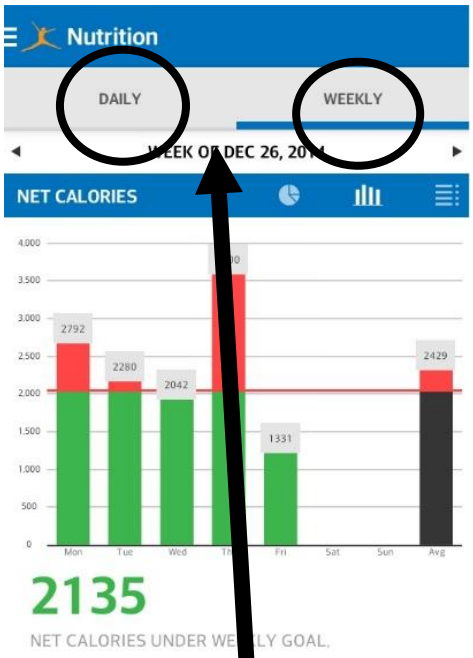


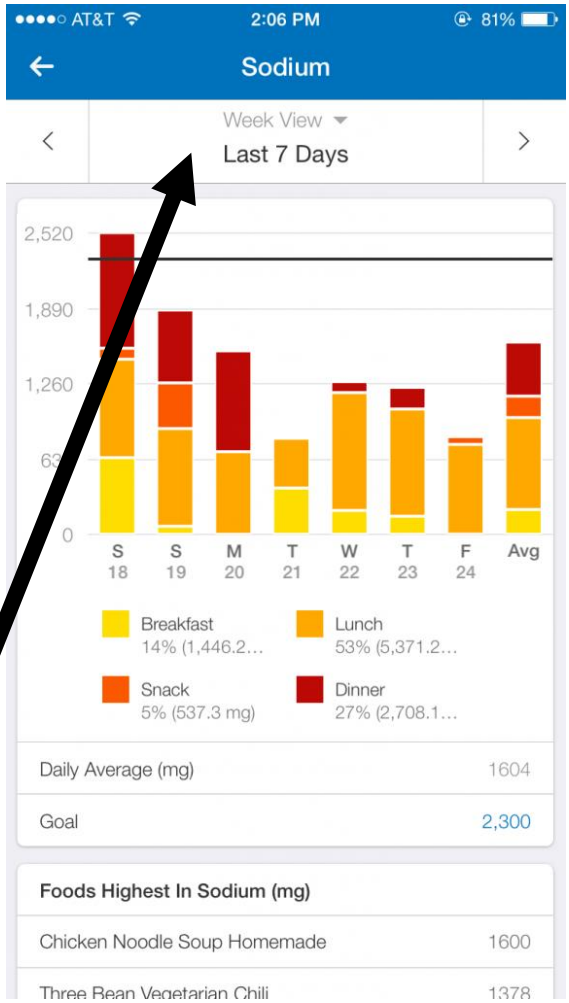
In this program tracker the categories show the amount of time they are used for but in my program this information is unnecessary as time is not needed when the nurse is trying to find out why the patient was moved to an overnight stay.

The iPhone screen time tracker on the iPhone settings app, uses coloured bars to identify each category of types of apps. From this I will implement the idea of coloured bars representing different consultants into my project. This will allow consultants to be easily identifiable and so the viewer doesn't get confused with all the information in front of them increasing its user-friendliness

Example 4:

MyFitnessPal

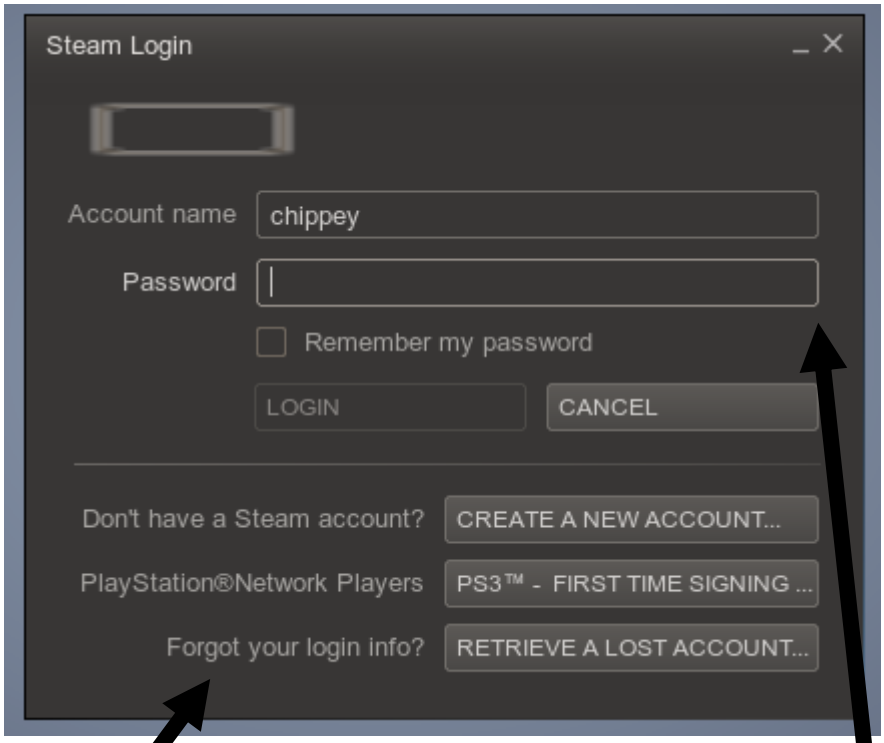




MyFitnessPal sorts their graph summaries of their different categories by daily and monthly. With this in mind, I will add month and year titles to each one, so it is easily distinguishable. If someone asks for the audit data for a certain month the user will be able to comfortably access and obtain what they need quickly making my software more user friendly.

Example 5:

Steam Login Page



This steam login has a forgot login info option that takes the user to their website to change their information. I will use a similar process by adding a "forgot login" button that will tell the user to contact me and display my email. This will allow my login page to be more user friendly by informing them to contact the developer. By doing this, it wont waste the nurses time as they would

These are the entry fields for the username and password of the admin, and I will use this same layout in my login page because it is simple and practical for inputting data to gain access to the main menu.

Features:

Login Page

In the program, on the first page, there is going to be a login page. The user would see an "enter username" and "enter password" textbox that will not accept invalid inputs like leaving the textbox blank, but it will accept the correct username and password which matches the admin login. If the username is incorrect the output would be "Incorrect username" and if the password is incorrect the output would be "incorrect password". Both would end the program. In addition, if the user cannot remember their username or password, they can click a "forgot login" button which brings a pop up containing the developer's(me) contact information. My login system will be like steam because it is a simple design which will allow nurses who are less able with computers to navigate around the login page with ease.

Main Menu

The menu will implement two main buttons, one on the left side of the page and one on the right that do not exist on the existing system which lead off to other areas named "Audit graph" and "Audit databases". The "Audit databases" button which brings up the database information of each month and the current database being used/edited during the current month. The user would also be able to view the "Audit graph" which displays the bar graph summary for each month during the year. On the main menu there would be a "logout" button for which the user goes back to the login page.

Database Page

The database will have two sections the "(Month name)/current database" and the "daily database". Each will have a simple layout and format so it is easy to enter data about a patient who is being moved to an overnight stay. In the "current database" for the present month, after you make a change and press the button "save changes", a textbox would pop up to ask, "are you sure you want to make these changes?" and if the user clicks "Yes" the changes will be saved to the database and the database will close. If the user clicks "No" any altercations will be deleted, and the database will close. There would be a "back" button which goes back to the year selection page. For the" (Month name)/current database" and "daily database" it will have a "back" button as well which goes back to the database selection page and "save changes" button at the bottom of the actual database which goes through the same process as the current database. The actual databases will be similar to the example of the health roster system as it displays the information about staff rotas

concisely, coordinating key data around the page which I require in my database pages. (E.g., coloured key = consultant)

Year selection page

Display all the years in a list on the page where each is a button to view data for different years. Here there will be a "Back to main menu" button which will take the user directly back to the main menu.

Month selection page

Display all the months in a list on the page where each is a button to view data for different months. Here there will be a "Back" button which will take the user directly back to the year selection.

Audit graphs Page

There will be a year selection stage which then shows the different months. This will bring up the audit graph page where the (Month name) graphs will be and it will also contain a "back button" which goes back to the month selection page. It will also contain a "view graph overview/summary" button which will allow the user to view the summary of results of specific month, from highlighting any key information, revealing trends as well as assigning values to each reason of overnight stay. The number will also be colour coded reflecting the consultant. The actual audit graph will be similar to the Wolverhampton statutory accounts as previously mentioned because a bar chart is the easiest way to present large chunks of data at once while maintaining the aesthetic look (iPhone tracker example). In addition, the graph summary will be like the table in the statutory accounts as important information is clearly visible to the user.

Limiting Factors:

There will be some limitations in my program that will affect what I am able to do in my project because of the lack of time, knowledge or skill. These features could be added on later to the program if it becomes essential to have.

One of my features the "forgot login" button would be limited because the user would not be able to actually change their username or password and the contact developer pop up will only display my email and not link anything to communicate with me directly. In addition, this is not really necessary as there is only going to be around five potential users which isn't worth the time of coding for the amount of people who are actually going to have a login. Moreover, logins will be given out by me the developer and I am relying on

trust that the nurses keep those login details safe and secure. There could be a problem if someone find out the login details as they would have access to some information of the patients on the ward at the time. The Information in the database is also stored in plain text which is not encrypted so unauthorised access to the database makes the content stored more vulnerable to data manipulation.

Another limitation is that my database can only be accessed by one user at a time so two users cannot log into it at once, which could potentially be a drawback as for example, the user waiting to access the database may need to make a more important change or needs to add something quickly as they are busier than the user who is already using the database. This can be problematic as in the future if more nurses have access to the database, there will be more occasions when multiple people want to use the database so the software would have to be updated to a multi-user system and because this is not in the specification, I will not be implementing this feature.

Additionally, in my database the user in read only mode wouldn't know if the person who is making to changes to the database has come out the program or not so they will have to keep checking every so often until they can login and I will not be adding a feature to fix this as this is a skill I lack.

My user interface will be very clear and distinct prioritising the functionality over the design making my software very user-friendly. Due to lack of time, I think my user interface will not be as detailed.

Computer Requirements

My audit software needs to be able to be used on the hospital staff's desktops and laptops.

Requirements

- 1GHz Central Processing Unit
- 1 GB HDD space
- 2GB RAM
- Windows 7-11
- Python installed
- SQLite installed

My client currently has an i7 desktop, windows 10 with 1TB HDD which is more than enough to install python and SQLite with any imported “appjar” features for the user interface, therefore it will have a very small storage capacity and be straightforward to run.

Testing

When testing the software, I will need carry out white box testing by internally trying every single possible pathway throughout my program ensuring it allows for all valid and invalid inputs. I will have to observe the outputs I expected and ones which were unexpected. Any errors will have to be logged (error table) and each success has to be documented. These errors will have to be debugged to give the correct output until there are none left in the entire program. Similarly, I need to be careful with my indenting of my code as it can cause a lot of unnecessary errors which will take time to fix delaying more important error fixes such as syntax errors and logical errors.

I will be giving my program to my client to test after I have finished most of the essential success criteria so she can give me a rundown of what she thinks about the software and what level she feels the user-friendliness is at, within a range from 1-10. This will be repeated with the staff on her ward to see if their judgement is similar but this will only happen if my client is first happy with the software as she is the manager. I will also be meeting with my client once a month to update her on what I have done and I will take on any concerns she has and try implement them changes going forward.

Success Criteria:

Number	What must the system achieve?	Explanation	How this will be done?	Quantitative or Qualitative	Importance
1.	My program should allow a user to login with the correct credentials.	My program must be able to allow the users to login to the software	This can be done by testing all the types of inputs the program can take including	Quantitative as this relies on the program only accepting valid inputs to by successfully recognising invalid data.	Essential

		providing the correct admin login	data which is invalid denying access to the software and this is important so it should be tested first.	This should be tested in prototype 1.	
2.	The daycase to overnight patient data present in the database should be represented on a bar chart.	Data should Be plotted on a bar graph so it easy for my client and her staff to view the summary of results for each month.	The information from specific field names in the database such as, Consultant, reason for overnight stay will be copied to the graph by using public classes and functions. This is a key part of my audit software.	Quantitative as necessary data will be copied to the graph or not and if it will be represented within the correct format on the graph. All this is going to be tested by me during prototype 3.	Essential
3.	Information must be logically displayed on each page	My program must have a clear display of the audit databases and graphs for each month going in order making my software more user-friendly for the nurses	The graph and Month database for each month will be separated into two and each will be accessed through the year selection page. The admin welcome text would be at the top	Qualitative as this will be recorded by meeting with my client every month to find out any feedback on the user interface of my software. This will be tested during prototype 1	Essential

			of the screen on the main menu.		
4.	My program must be able to display pages in a sensible time.	The software should be responsive to the user allowing for information to be viewed quickly and this important in a busy hospital where time is limited.	This could be done by loading only parts of the database which would reduce loading times and it is can be beneficial for information only needed by a specific nurse.	This is qualitative as this depends on my client and her staff and what necessary information, they use the most.	Not Essential
5.	Data entered into the data base should be valid and stored in the correct format.	My program should be able to prevent invalid inputs such as “blank” so information can be seen swiftly in the correct layout.	Only allowing inputs specified, eg: for the consultants pick a number between 1-18 each representing someone different. This is should reject any symbols or letters entered by the user.	Quantitative as program can only accept valid inputs which of some are specified and this is going to be tested by me throughout the whole of development. (prototype 1,2,3)	Essential
6.	The colour scheme should be representable of The London Clinic: light/dark blue and white.	This will be done to match the colours of The London Clinic website and this is	I will be taking inspiration for the website for my usage of colours and discussing	This is qualitative as my colour scheme is based on my client’s opinion and it will be further	Mildly essential

		also what my client has requested. This will provide more of a familiar feel to the staff which should make it simpler to use.	with my client and her staff about where they want to see the colours on the user interface.	reviewed during our meetings throughout my design and implementation phase. I will also confirm with her staff by word of mouth if they are happy with the presentation of the program.	
7.	My program should be able to run on a standard desktop/laptop in the clinic.	My software needs to work on this machine to be functional.	I must make sure my code is efficient, abstracting any unnecessary code such as useless statements and this will decrease the file size of my program, speeding up its loading time.	This is qualitative as testing of my code will determine if it is able to run on the desktops. I will need to test every module and every possible path that can be taken which gives an output. Testing done in prototype 1 and prototype 3	Essential
8.	Daily database for each day should refresh everyday to allow for new data.	Once the days data has been collected and transferred to the current month database the daily database should	A function will be called to change the name of that database to that specific month and at the same time a new daily table will be created	This is an example of quantitative data as I am running the program and checking if a new table is formed every day. I will test this in prototype 2, making sure the	Essential

		refresh making it easier for staff to focus on adding new information without the distraction of other days data, decreasing the chance of a mistake.	incrementing the date by one. (going into the next month)	table refreshes at the same time everyday revealing consistency within my program.	
9.	The program colour scheme should be same as the OS theme used.	If the user changes the OS theme the colours of the program should change too.	This will work by identifying the users chosen theme and applying the colours from it to my program. This will make my software look better improving the design of my user interface.	Qualitative as this will depend on my client's perspective of whether it is **preferred to the clinic's colours or not.** This will be considered later on in development as it is not very important.	Not Essential
10.	Data can be seen/altered at any time	Data from the monthly database can be accessed by the user at any time to make any changes or it can be viewed to obtain any	I will make sure the month database can be edited by trying to change the data the following day I inputted it and then the day after that at a	This is Quantitative as I am testing the accessibility of my database and if data was kept correctly, which is an important part in the early developments of the software. (prototype 1)	Essential

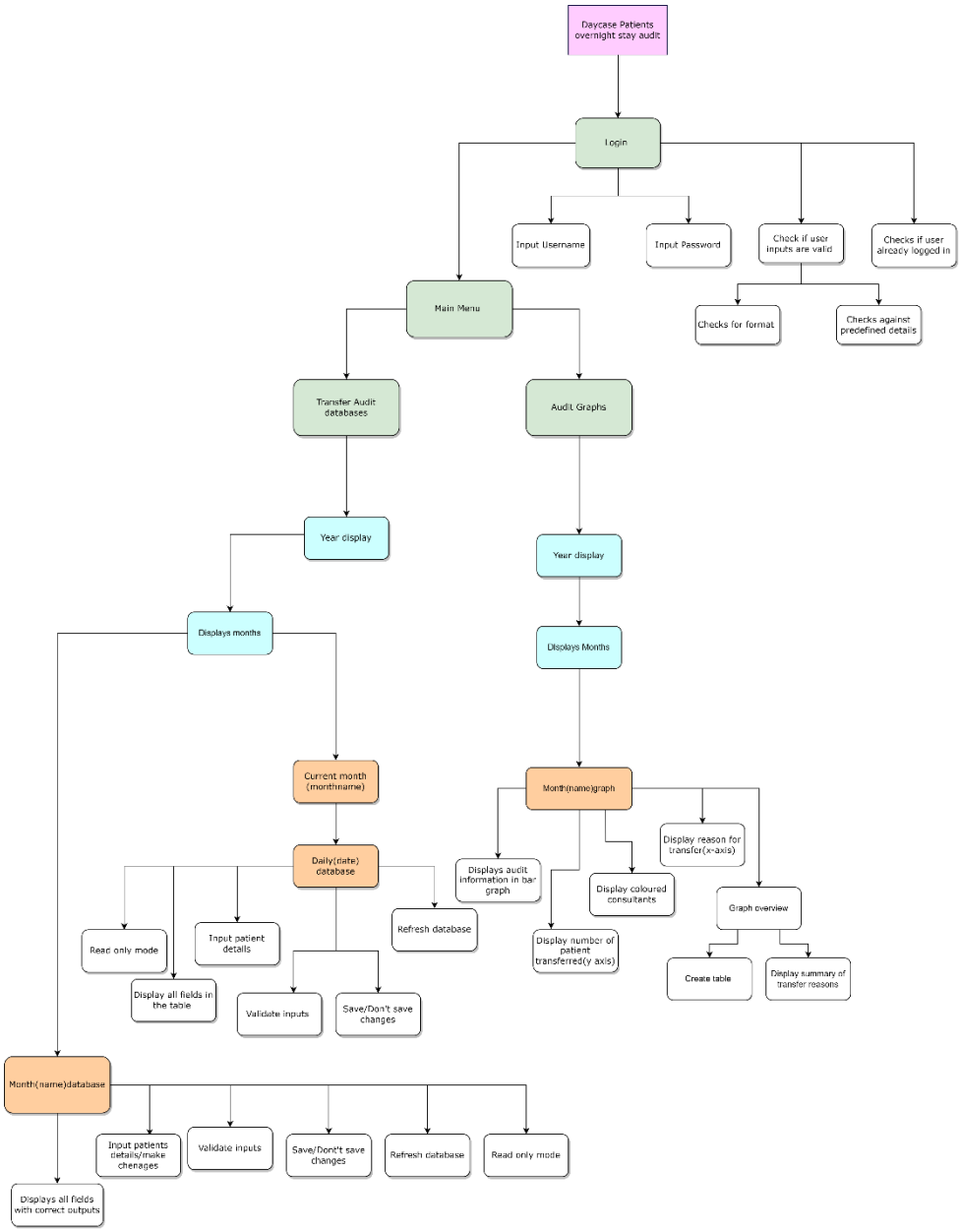
		necessary information	different time. At the same time the saved data should remain under the correct fields.		
11.	Make sure I have an updated database	My database should be able to update after new information has been added as data needs to be accurate and up to date.	I need to make sure two people cannot access the database at the same time as one of the records might not be saved and to prevent this, the person that logins to the software when someone is already in the software making changes to the databases, will be permitted to login but placed in read only mode so they can only view not make any altercations to the databases.	This is an example of quantitative as the program will only be able to be changed by one user at a time and this will be tested as the database is made which is during prototype 1 and 2.	Essential

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Design

Introduction

My program will be made up of 4 main modules such as the login, main menu, Audit databases, and Audit graphs and each of these will divide into many subroutines. The login page will allow the user access to the software. The main menu will display the name of the audit and the options for database information as well as the audit graphs. The database information will contain the monthly database and the daily database allowing for daily inputs of patient information. In the audit graphs, there will be a graph for each month and a summary for each. These modules will be linked by functions in public classes being called into the main program. OOP will be used for the inheritance of classes such as the "graph overview" will inherit some objects from the "(Month name) graph" if they are representing data of the same month. This will save time and effort in the implementation part of my project.



Why is each module needed?

1) The login page – this will be the first window the user welcomed with when the software is opened. This will contain two entry boxes for logging in and two buttons which one will be used to login with the entered inputs and one called “forgotten password” that directs user to an information box containing my contact information. I will also incorporate a quit button to close the program because the user will return to the login page if they log off from the main menu. I need this window for the nurses to access the all-audit information and for them to be able to make any changes.

2) Main menu – the main menu is small section but is needed to access different areas of information in my program such as the audit graphs and database information. It creates a user-friendly interface so nurses can get what information they need with ease. It also acts a middle ground between logging in and using the functions available to the user.

3) Transfer Audit databases sub-menu – this module allow for one of the main functions of the software which is the storage of patient information and their reason for overnight stay. This can be broken down into subroutines of month database or to view the current months or daily audit which all can be edited and saved. This will track the overnight stays throughout the month and it will minimise any missing data during a busy day in the hospital as all data is in one place. Without this, data for each month will be on multiple word documents increasing the risk of misinput.

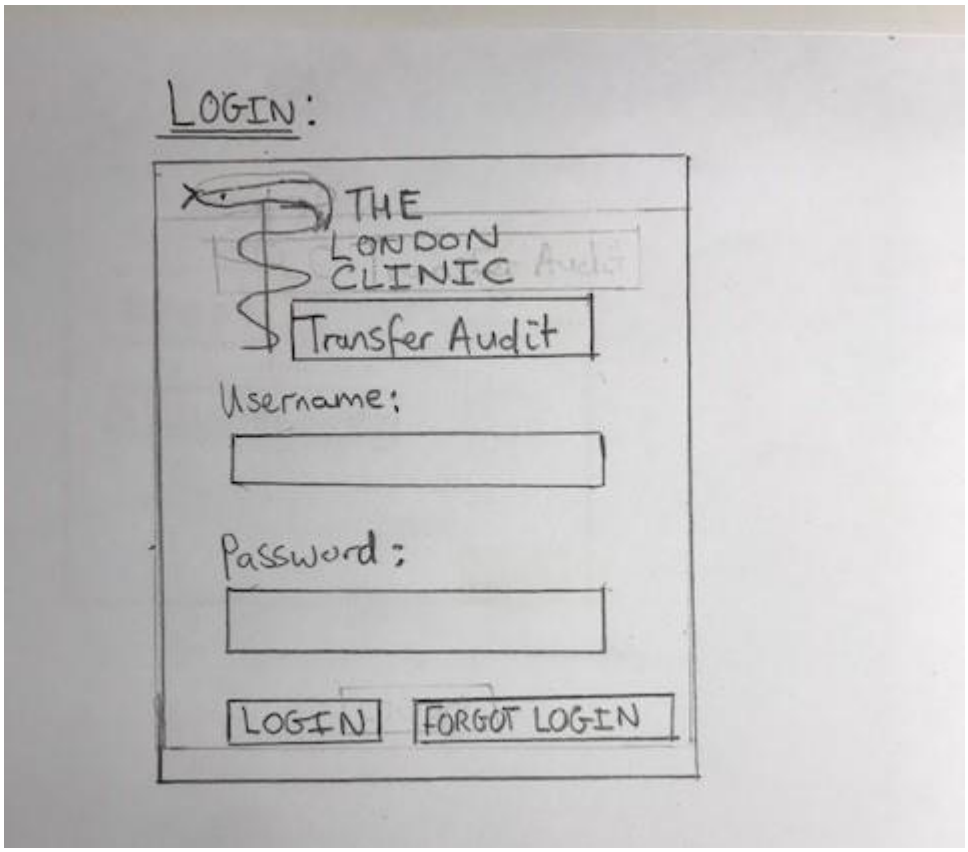
4) Audit graphs sub-menu – This module is for displaying the database graphically providing a better presentation of lots of data. Without this, the data will be harder to see which can lead to false information being obtained. It is also needed to view trends in the graph and display it in a table again making it simpler to find essential information. This will be colour coded in regards to each consultant the patient is under, improving its user-friendliness.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Initial designs

I will be using the Appjar library for the user interface.

Login:



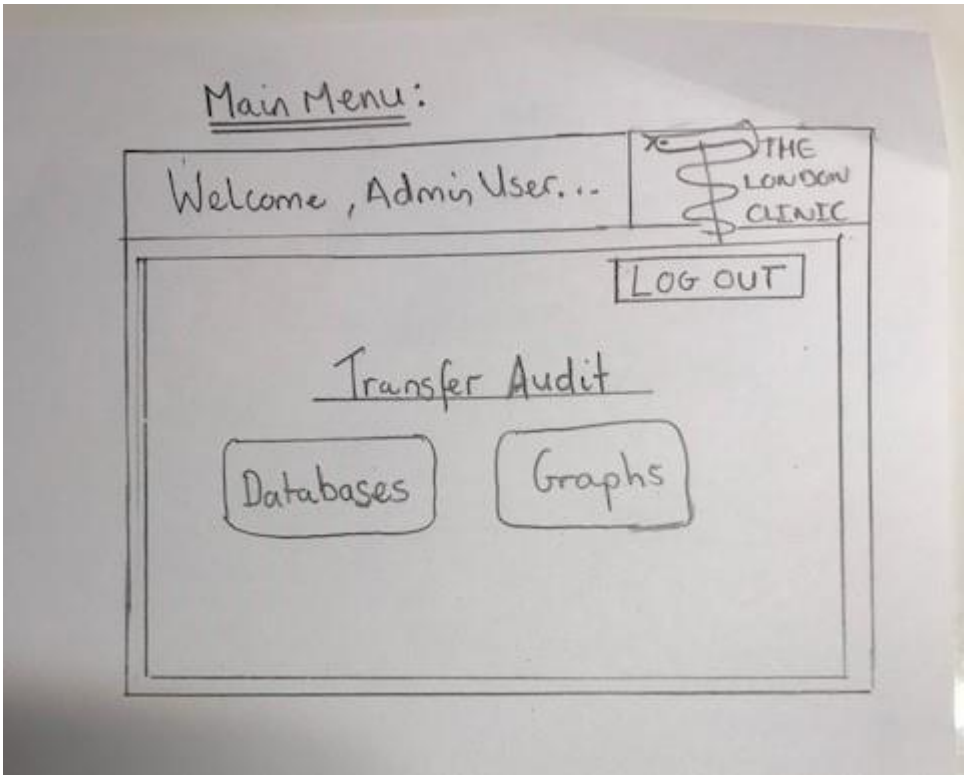
This screen will be the first thing the user is met with. It displays the title of the entire system and two textboxes for the username and password.

There are two buttons. The “login” button that will take the user to the main menu and the “forgot login” button which will bring up a pop-up box with my contact details.

It will also have the company logo in the middle top of the screen. This will give the login screen a more official and professional look. In addition, with the use of colour, it will be more appealing the user.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Main Menu:

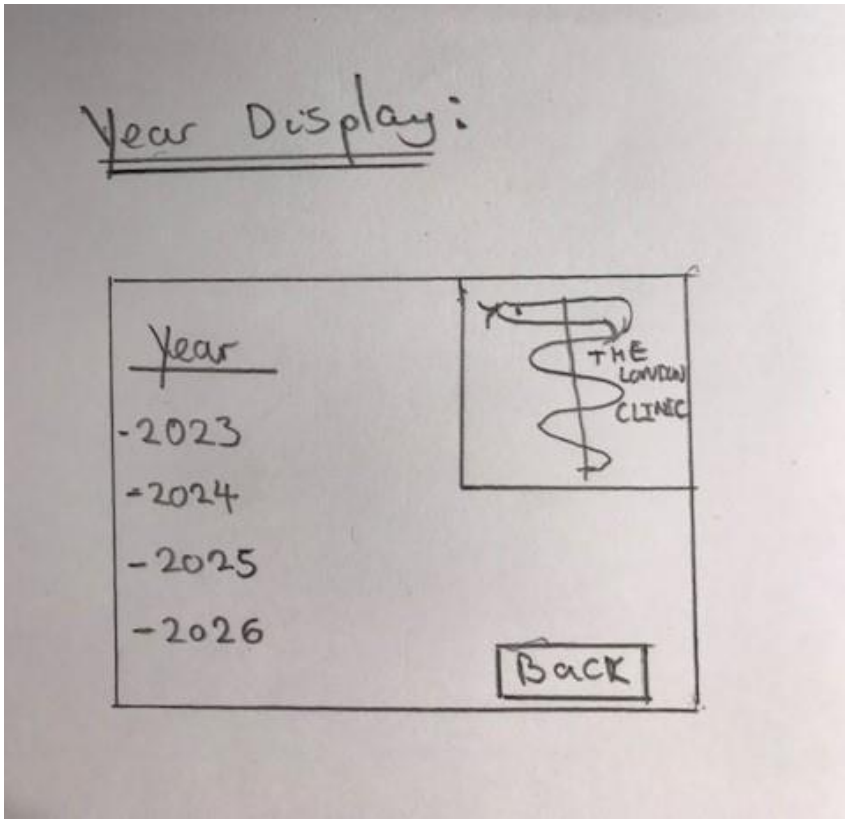


In this screen, the main menu will be displayed where at the top of the screen the admin user will be welcomed.

There will be two buttons under the title of my program, “databases” and “graphs” describing the two main functions of the software. There will also be a logout button to take the user back to the login page. This “logout” button will be a different colour to the other buttons to clearly represent to the user what function they need.

Ensuring professionalism, I will have the hospital logo on the top right of the page.

Year Display(additional):



This year display screen is an additional feature to my program and because of this it will have a simple look. It will be shown to the user after either two options on the main menu.

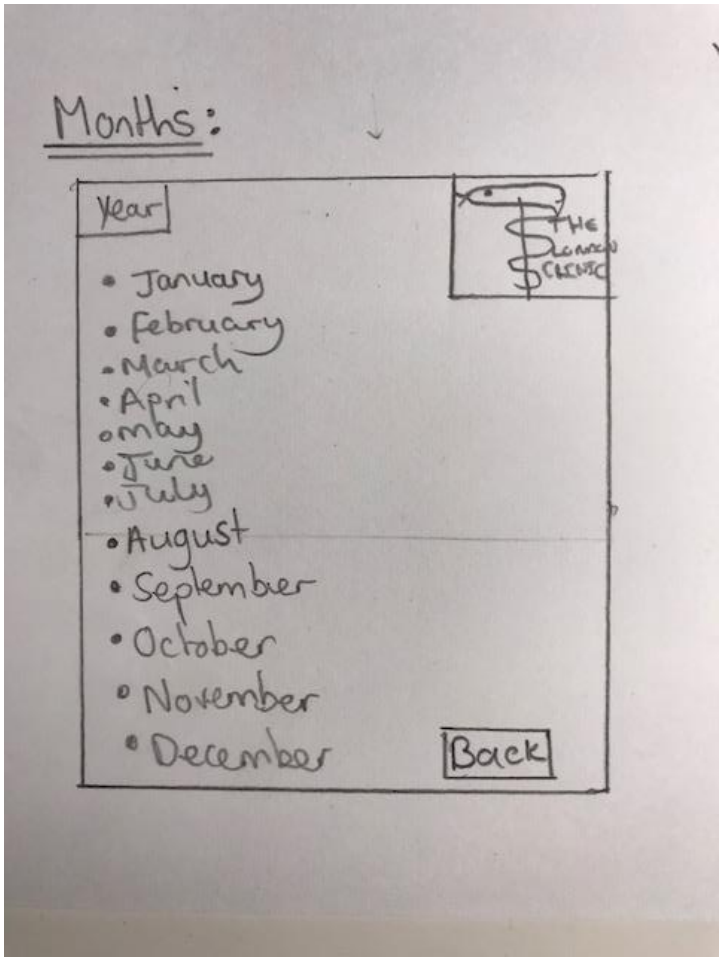
This will display five buttons under the title of year, from the current year of 2023 to 2026 which follows onto the correct data for that year.

In the bottom right of the page, there will be a back button that returns the user to the main menu.

Here, again there will be the hospital logo in the top right to show the user they are using the correct program.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Month Display(additional):



This is another additional screen my program will have. It will be shown to the user after either two options on the main menu.

This page will display all the months of the year so 12 buttons in total, all leading to those months data.

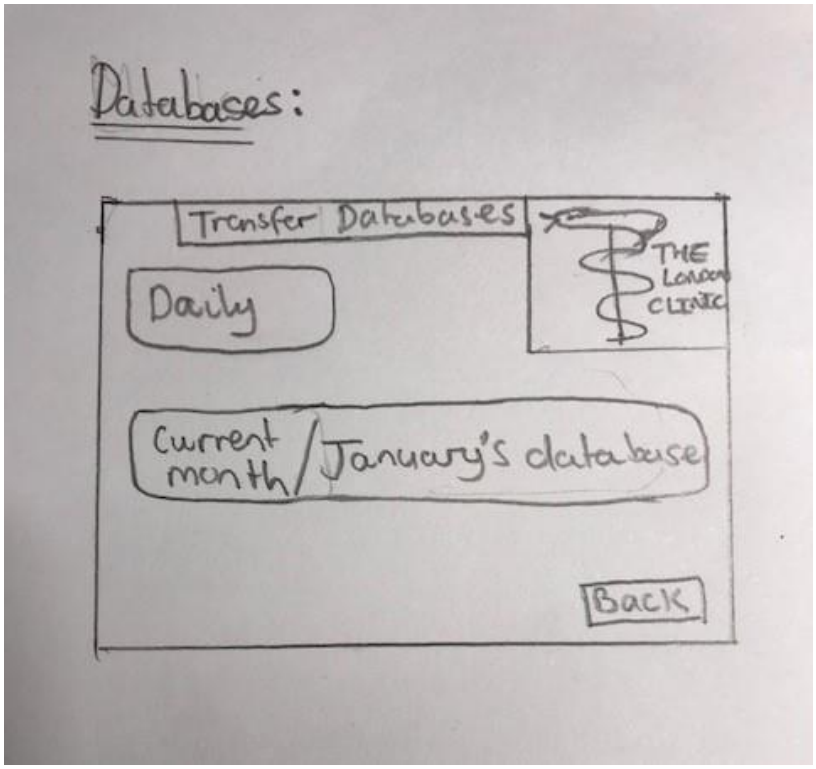
I could use abbreviations for each month to increase visual simplicity of the software but this would have to rely on all the users to already know what each one means.

It will also have a back button in the bottom right to take the user back to the year selection page.

The logo will be present again in the top right of the screen to show that the program is official.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Transfer Database Sub-menu



This screen is one of the main features of my program with the title transfer databases.

It will contain three buttons, “daily”, “current month” and “back”. The daily button will show the user the data which has been collected for the current day. The current month button will show all the data that has been inputted and collected throughout the month.

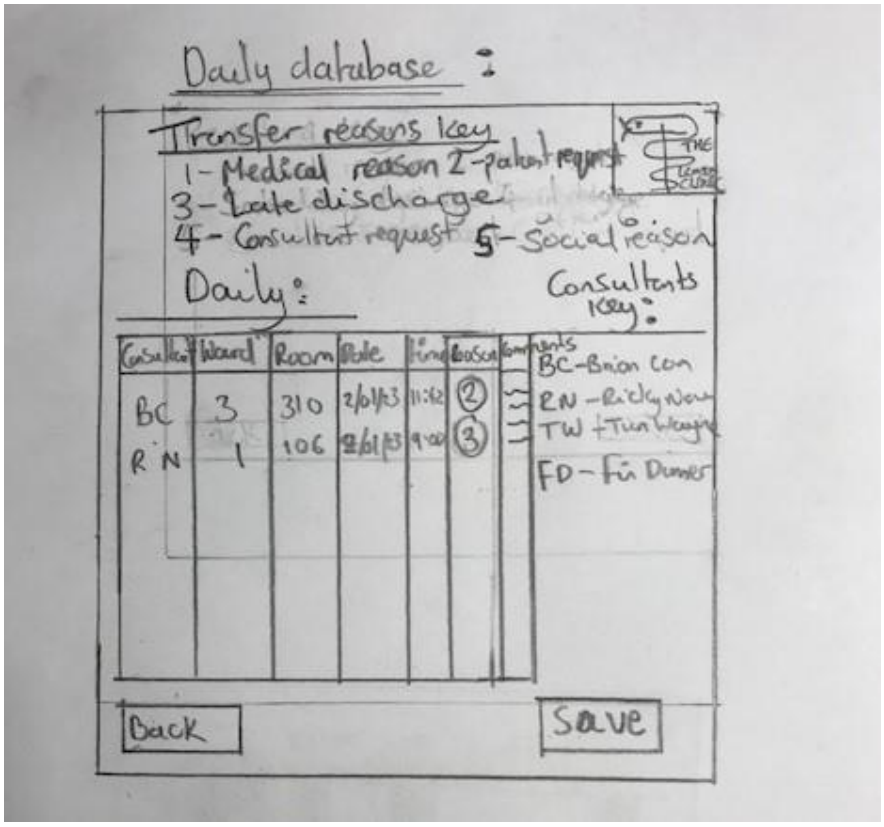
For “current (month)” and “daily” button I need to ask my client and her colleagues about feedback of how they would like each month button to be displayed on this screen and this will be corrected in my final designs.

Finally, there will be a back button which takes the user back to the months page.

The hospital logo will be displayed again in the top right corner.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Daily database (key submodule of Transfer Databases)



This page is what comes up in the Daily button. This displays the transfers that have taken place for the current day and information about each. The table consists of the consultant, ward number, room number, date, time, reason for transfer and a comment box for any additional details about the patients move.

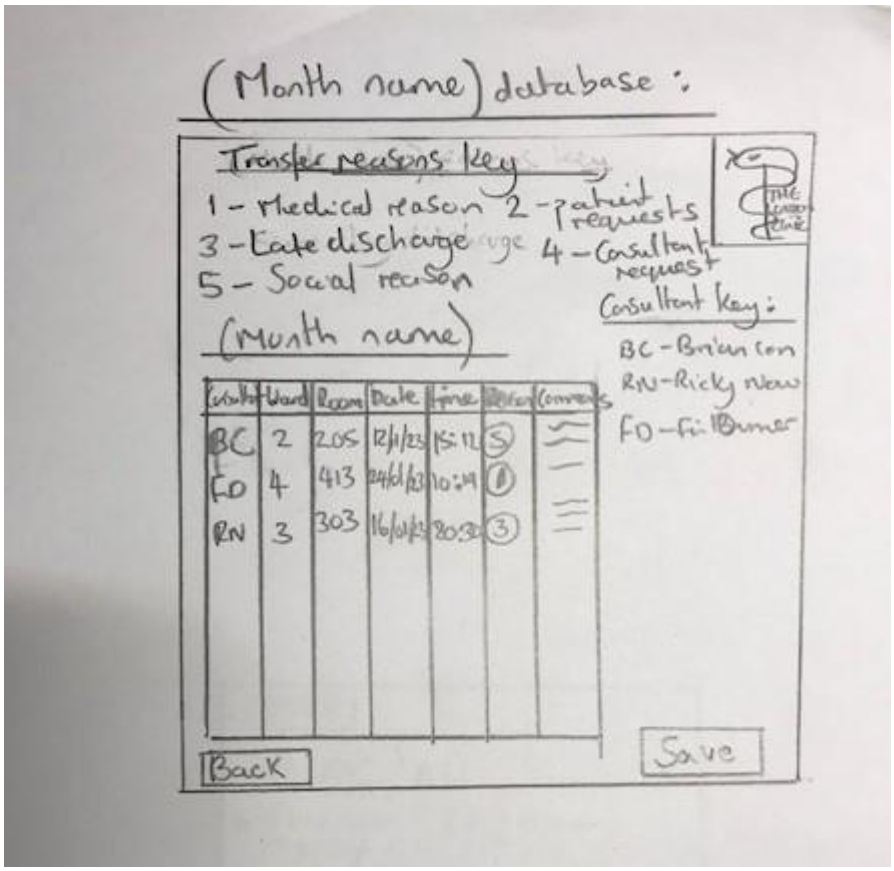
On the top and side of the screen there are keys for transfer reasons and consultants. This will allow inputting data to be less of a challenge by saving time for new users using my software. It also gives a simple look to a large table containing multiple fields. The consultant names will be listed with their initials, to minimise distractions on the screen and for guidance.

There is “back” button which takes the user back to the transfer database sub-menu. Moreover, in the bottom right of the screen there is a “save” button which saves changes made to the database so essential data is not lost.

Hospital logo is present in the top right for aesthetic purposes.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

(Month name) database (key submodule of Transfer Databases)



Here the monthly database is displayed, this is after the “current month” button is clicked. The same table which is displayed in the daily database is on this page. This will show the whole months data for the ward.

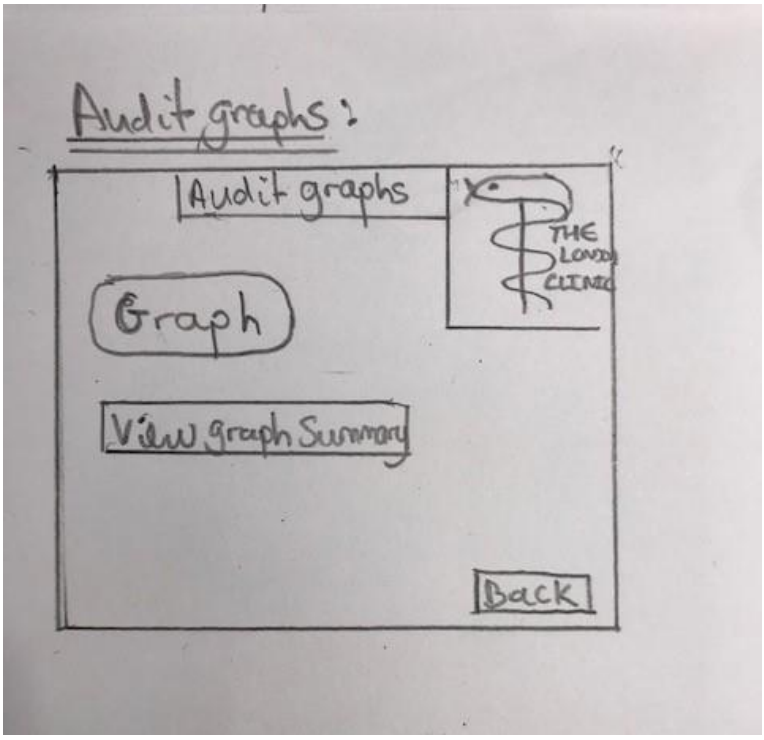
There is a consultant key to the right of the screen which shows the initials of each person and a key on the top of the screen representing each reason for transfer. This will make data inputs less of a hassle for staff.

The “back” button will take the user back to the transfer database sub-menu. There will also be a “save” button to keep important changes being made to the database.

Company logo is in the top right for branding.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Audit Graphs sub-menu:



This screen holds the other main feature of my program with the title of Audit graphs.

It contains three buttons, “month graph” and “view graph summary”. The graph button will show the database’s data for that month in a bar chart and the summary button will highlight any key information, revealing trends between consultants.

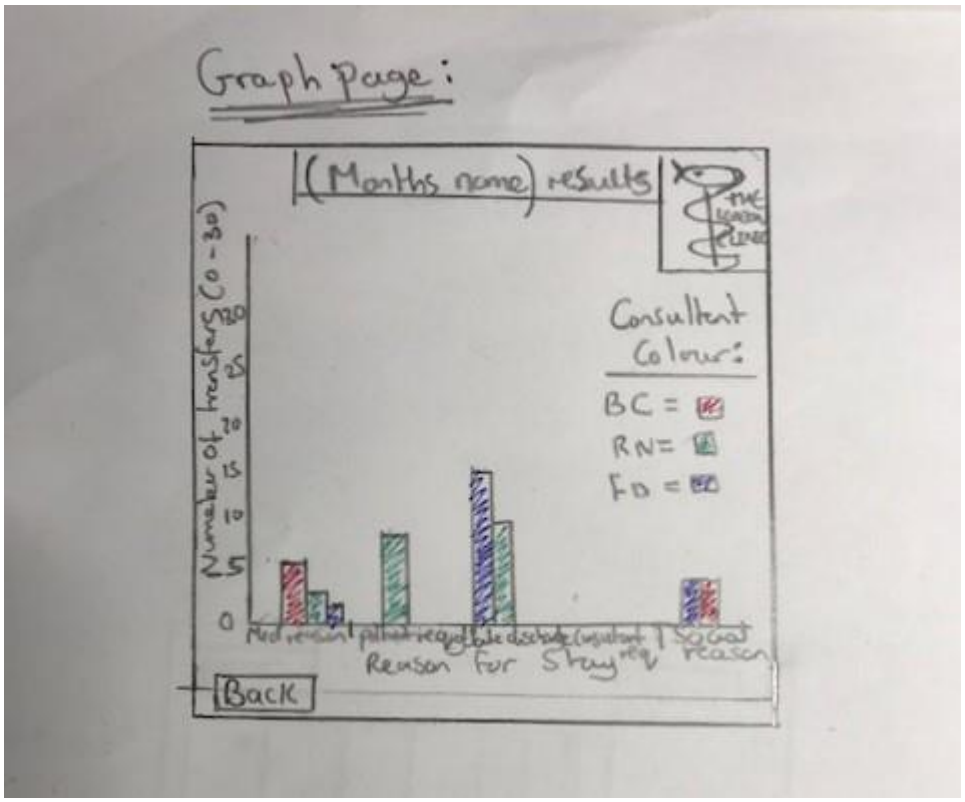
The graph and summary button will correlate with the correct month, so the user is sure they are viewing the desired data. Both buttons will be different from the “back” button so the user can differentiate between the two.

The “back” button will take the user back to the month’s selection page.

The hospitals logo is in the top right corner for officiality.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Graph page (key submodule of Audit Graph sub-menu)



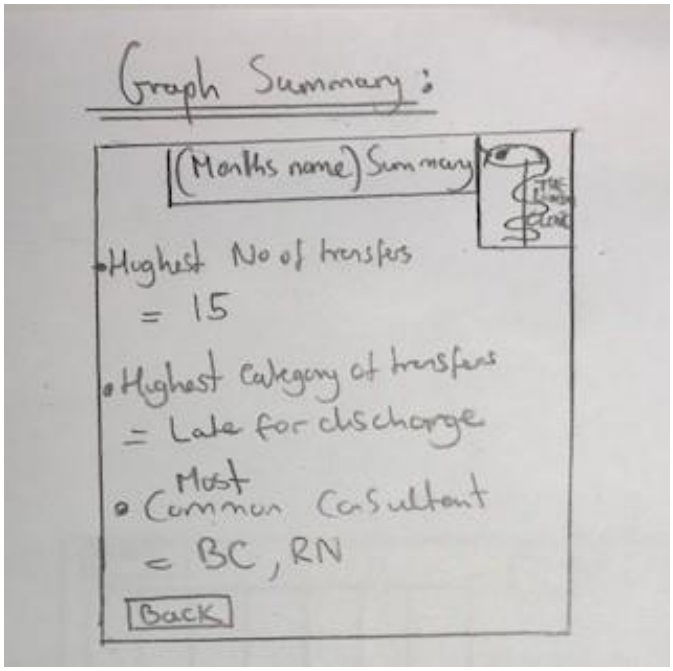
This page comes after the "graph" button is pressed and this presents the results in bar chart of reasons of transfer against number of transfers throughout the month. The bar graph is coloured depending on the consultant.

On the right of the page there is a colour key for each consultant, so it is easy to identify each bar and who it relates to. This results in saving time when looking for data from a specific consultant.

There will be a "back" button in the bottom left that takes the user back to the Audit graph sub-menu.

The London Clinic logo in the top right corner.

Graph summary (key submodule of Audit Graph sub-menu)



This is the summary of the bar chart that finds the highest/common data and displays it in word format for staff trying to obtain any useful information such as, the most common consultant, which shows the person may not be following hospital protocol. As a result, the staff will be spoken to and questioned about their data.

The different features of the graph will be places down the page so it is easier to read.

A "back" button is in the bottom left to take the user back to the Audit graph sub-menu.

The company logo is present for presentation purposes only.

Feedback:

I asked the nursing staff who were going to use the software what they thought about my initial designs, and this is what they liked and what could have been improved. I'm going to listen to Mrs Heer's opinions the most as she is my main client.

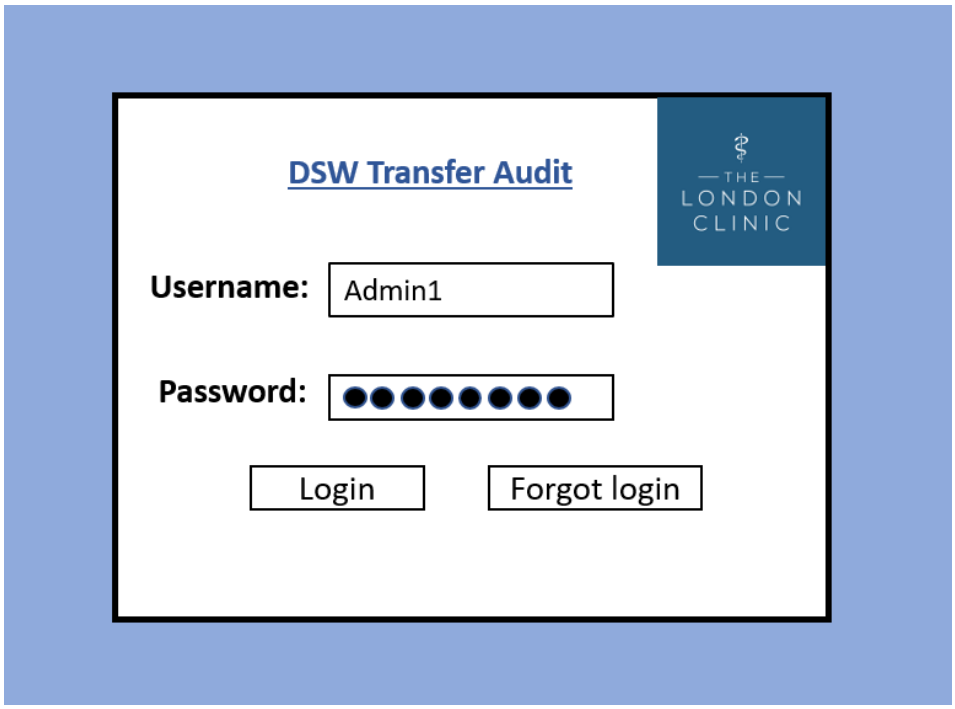
Admin staff	**Feedback**	
	Strengths	**Weaknesses/Improvements**
Mrs Heer (Client)	"The layout looks well-planned and thought through as it shows instant results of the transfers and it is not too complex. Easy to understand for anyone else to interpret the data."	She said the "back" buttons should all be on the bottom left of each page to maintain consistency throughout the program. The years/months buttons should be listed down the page because it allows for an ordered sequence. Remove the "welcome admin" on the main menu as we don't know who will be accessing the database at that moment.eg (a nurse from the same ward on someone else's account). The transfer reasons key should have an "other" option if the reason is not available in the selection of choices. Have the clinics logo on each page to confirm it is a London Clinic document and proves there is no plagiarism.
Daphnie Coloney	This admin said, "The login system is a great addition to the security of the software especially because it can	They said the different months and years should be listed in order down the page. "I would prefer the months to be in ""MM/YY" and she said to only have the logo on the login page and main menu as it can be distracting when viewing the database/graph. The

	contain sensitive information."	button of "databases" and "graphs" should be renamed as "patient information" and "Audit graphs" so it is clear and more concise to all the staff using it. Moreover, she said the "logout" button should be at the bottom of the main menu because its "easier to see",
Micheal Rag	The admin said, "I like the layout of the main menu and it makes it simple to get what I want even when I'm busy with patients. This allows me to get what information I require, when I want quickly and give me more time to tend to the needs of patients"	The admin said the buttons should be different colours, so each module is easily identifiable and for it to be aesthetically pleasing. He said the graphs/databases should be placed horizontally on the audit pages. He said the title of the transfer audit should be "DSW – day surgery ward" as it is only for their floor not the whole hospital. In addition, he said he would prefer the "logout" button at the bottom of the main menu.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Final Designs

Login screen

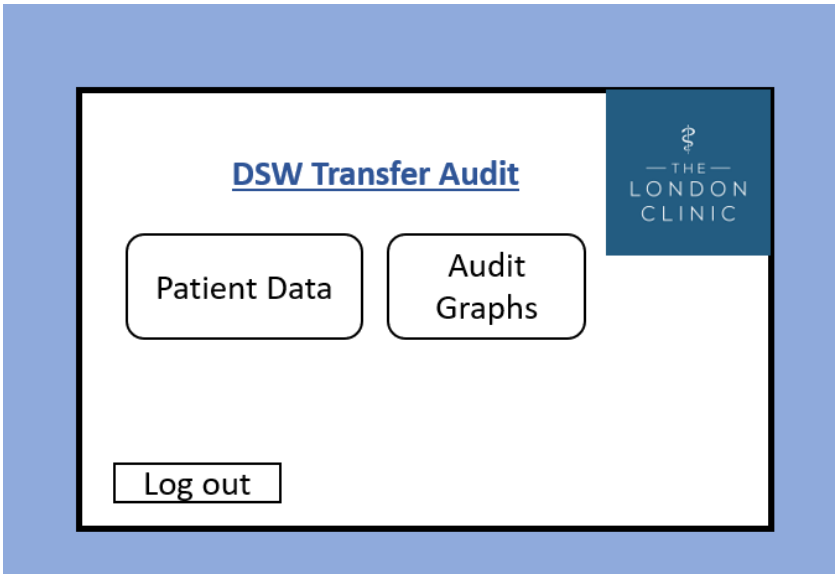


.

With this login screen, I decided to follow the company colours and use blue, white and light blue throughout my program. This gives the pages a more professional and cleaner look. I have changed the name from "transfer audit" to "DSW transfer audit" as one of the staff on the ward requested and this also makes it more specific. The "username" and "password" fields are labelled to not confuse the user and I have moved the company logo to the top right of the screen to reduce distractions as well as increasing simplicity of the login screen. For validity, the username should have a capital letter and I will sure the password will be 6 characters long also beginning with a capital letter.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Main menu



In the main menu the two main options are displayed “patient data” and “audit graphs”. I have moved the “log out” button to the bottom left as most of the admin staff preferred the button to be at the bottom instead of the top.

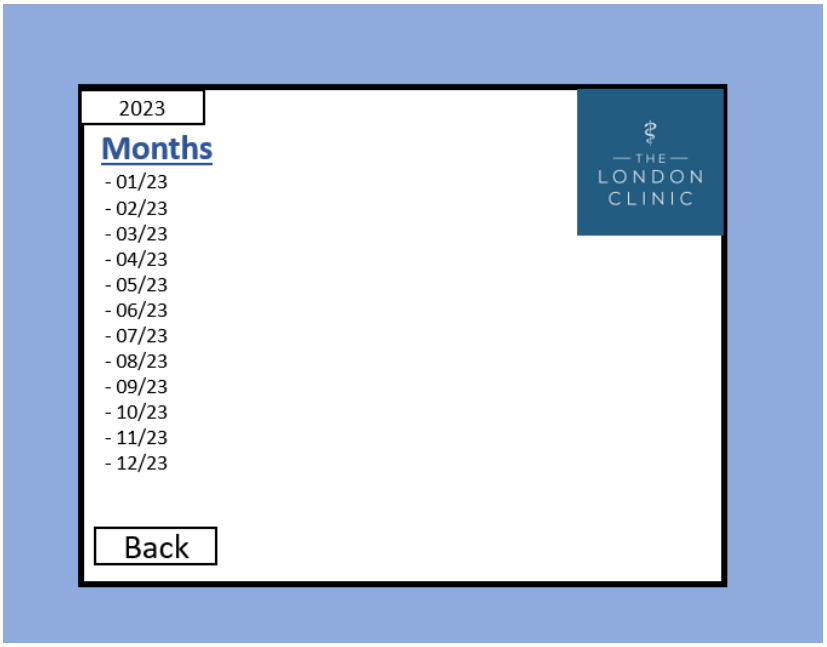
The “welcome admin user...” has been removed as my client said it is unnecessary information that may count as a distraction also another admin may be using someone else’s account so the wrong “welcome” message will be displayed. I have changed the title of the page to “DSW transfer audit” to again specify what floor the audit is for. The title is the same colour as the rest of the software to keep within the company colour scheme.

Year Display

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

The is the year selection where nothing much has changed as it already simplified. The only change that was made was the "back" button was moved to the bottom left of the page to match the rest of the pages in my user interface.

Month Display

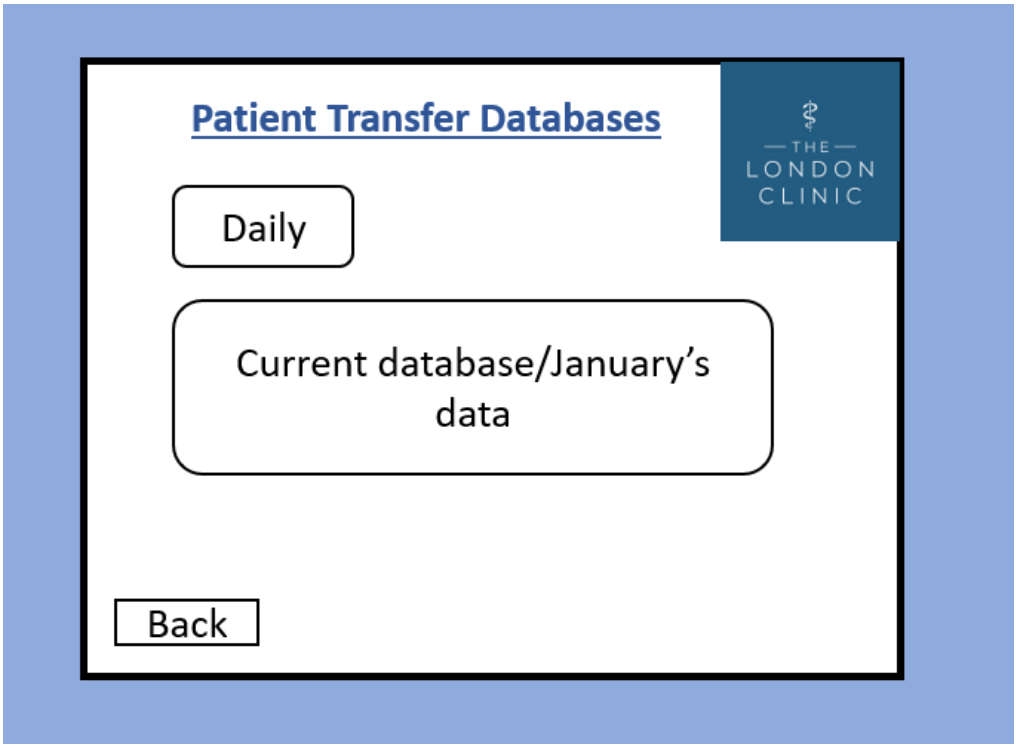


In this additional page the different months have been converted into their MM/YY variants. This has been done as one of the admin staff requested this change because it is easier to see numbers than long words on a screen, reducing the amount of data, increasing simplicity on the page.

The year chosen is now presented on the top left of the screen so the user does not forget which years data they are viewing. In addition, the "back" button has been moved to the bottom left of the page maintaining consistency throughout the software.

Patient transfer sub-menu

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

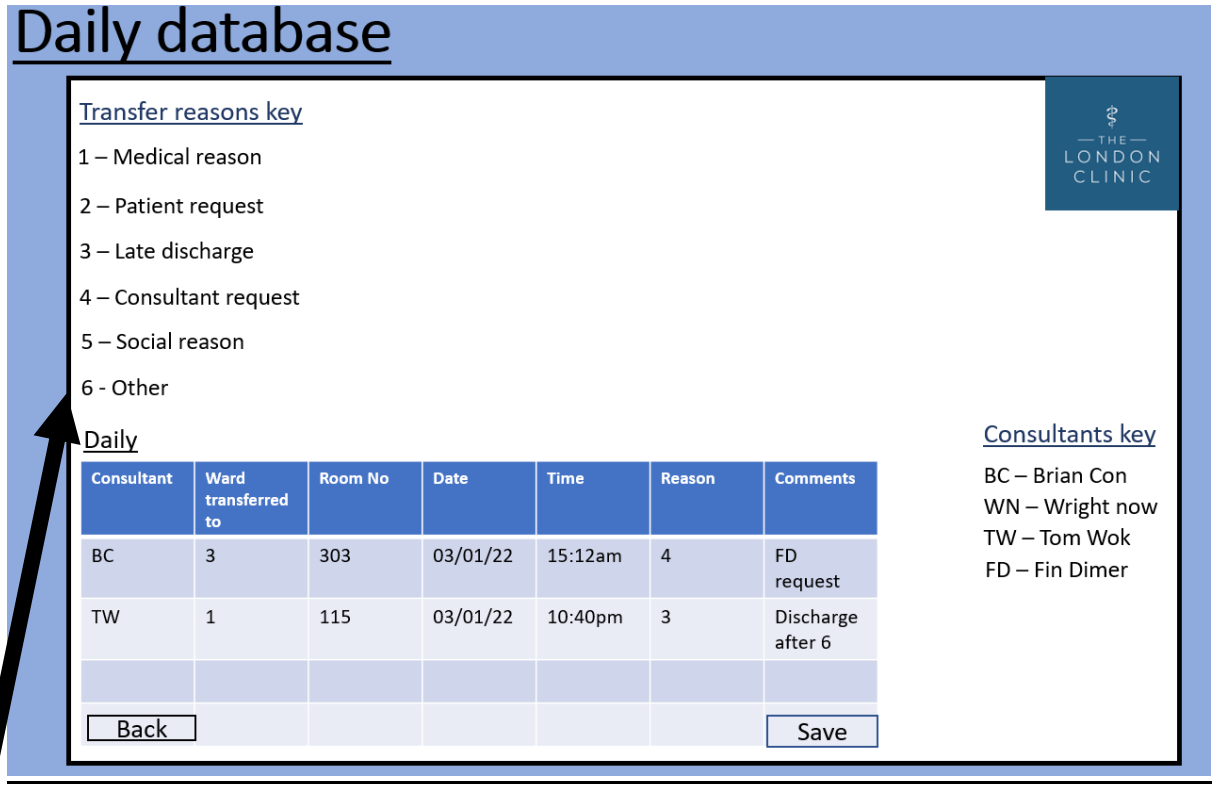


The title of this page was changed from "Transfer databases" to "Patient transfer databases" as my client wanted to be more specific so her staff knew what exactly the databases involved. The title is made to match the company logo for aesthetic purposes.

Most buttons were kept the same as they are already very clear. The "back" button was the only one which moved to the bottom of the screen to again, show consistence in each page.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

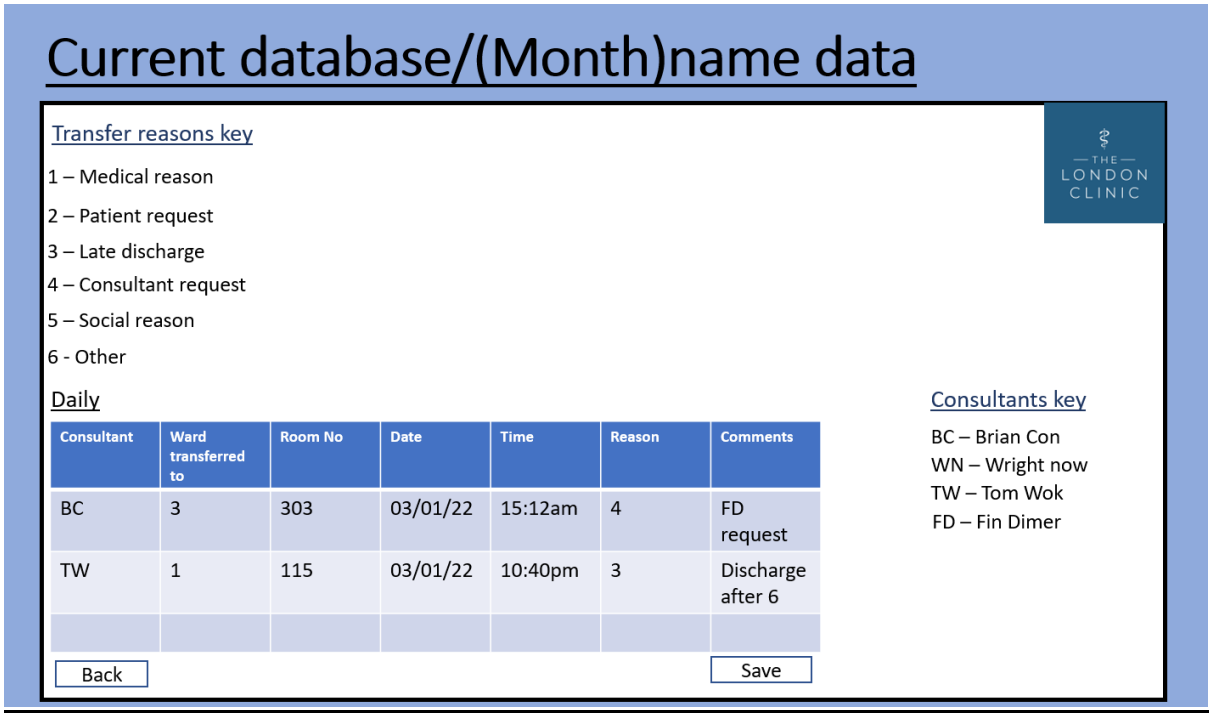
Daily Database



The daily database displays a table for the information about patient transfers with the transfer reason and each consultant having a key for easy input of data. The thing that has changed is there is now a 6th option in the transfer reasons key known as “other”. This was requested from my client as another choice if the reason is not on the list, it can be put as “other” and its details written in the comments. Another change is that the ward field will be a string as my client explained that there will wards that are not assigned to numbers by which means they have a specific name. Each record will be checked for validity. The consultant entity will only allow for a maximum of 3 strings, the room number will only allow for 3 integers, the date and time will be a specific format and the reasons for transfer will only allow for integers to match the reasons key. Both key titles are highlighted blue to represent The London Clinic colours.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

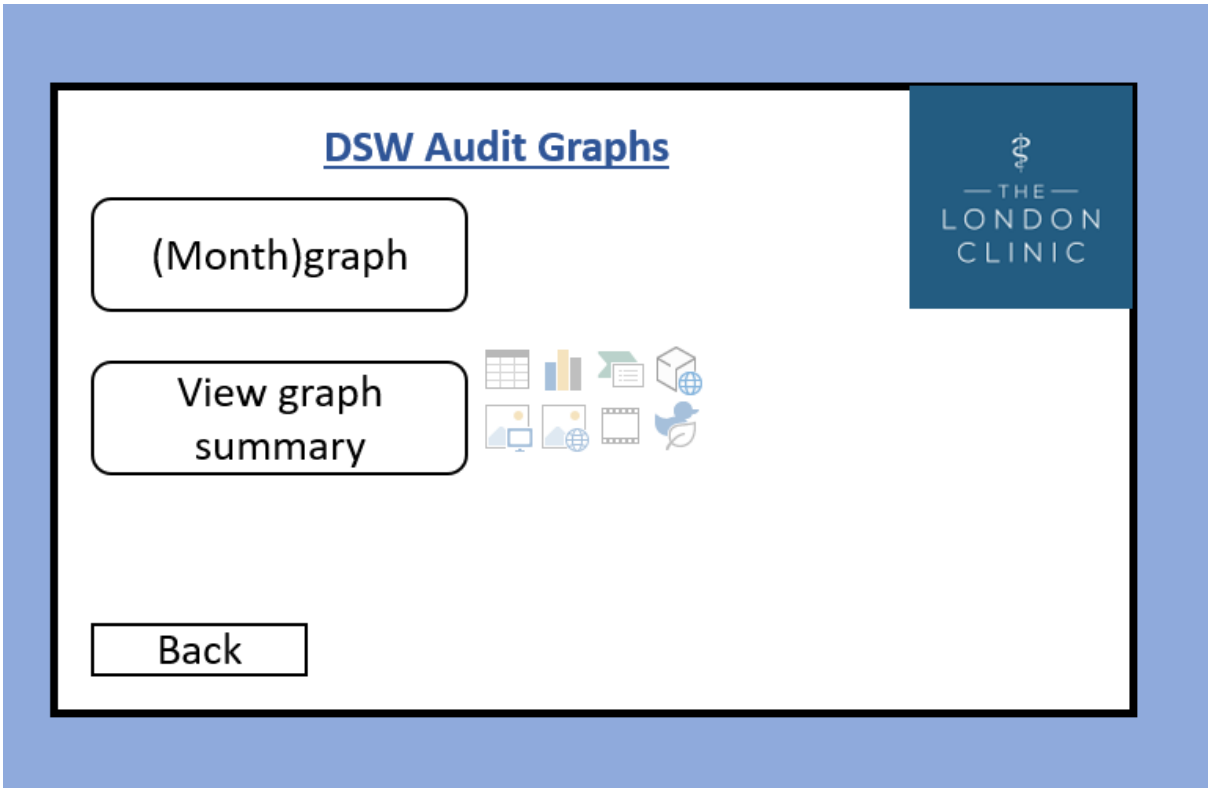
Current database/(month) data



This current database page is the same as the daily database just with a different title which is also blue and the table matches the colour of the logo to make it look more authentic for nurses. The keys again make it a less of a task for busy nurses to input data. Here again, the ward field will be a string as my client explained that there will wards that are not assigned to numbers by which means they have a specific name. Each record will be checked for validity. The consultant entity will only allow for a maximum of 3 strings, the room number will only allow for 3 integers, the date and time will be a specific format and the reasons for transfer will only allow for integers to match the reasons key.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

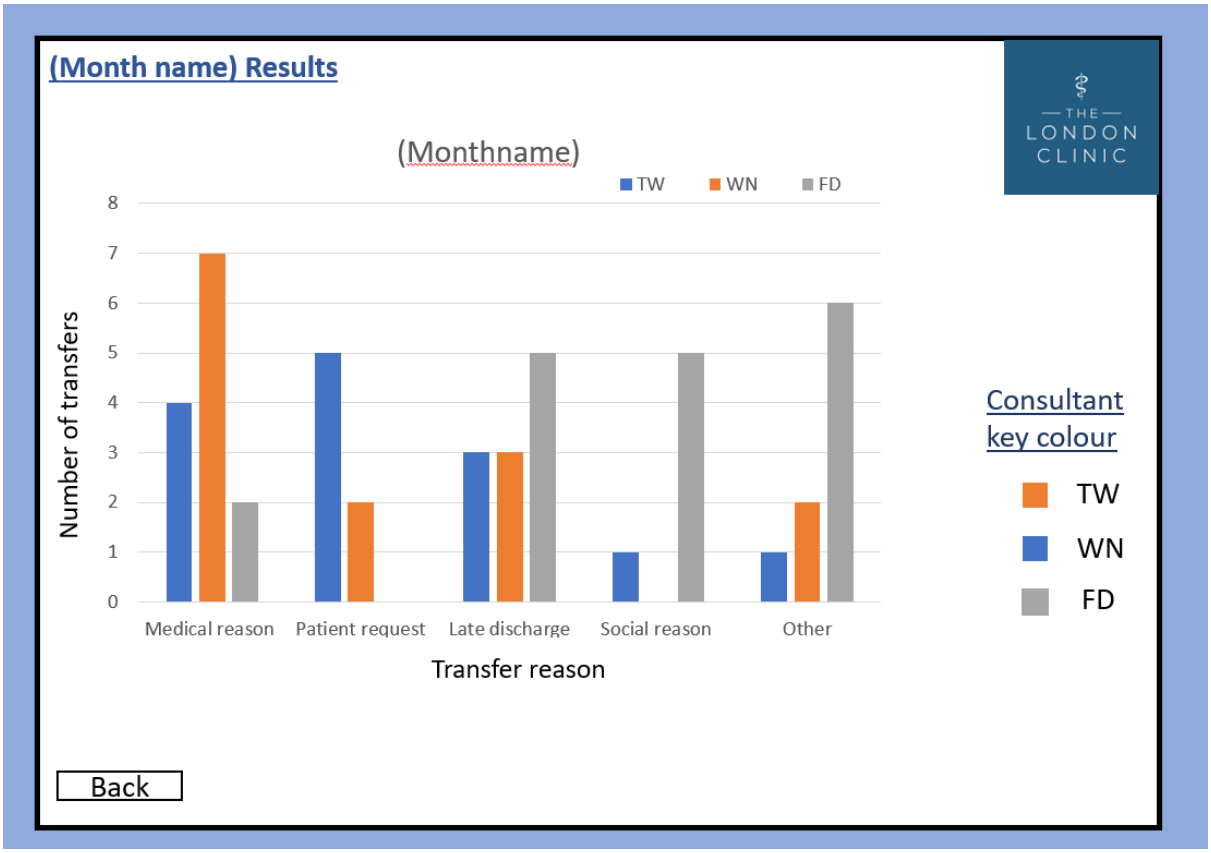
DSW Audit Graphs



The audit graph page is already very simple so only minor changes were made. The title of “audit graphs” was changes to “DSW audit graphs” as my client said this is more specific to the program. The “back” button was again moved to the bottom left because it was preferred by all the staff.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

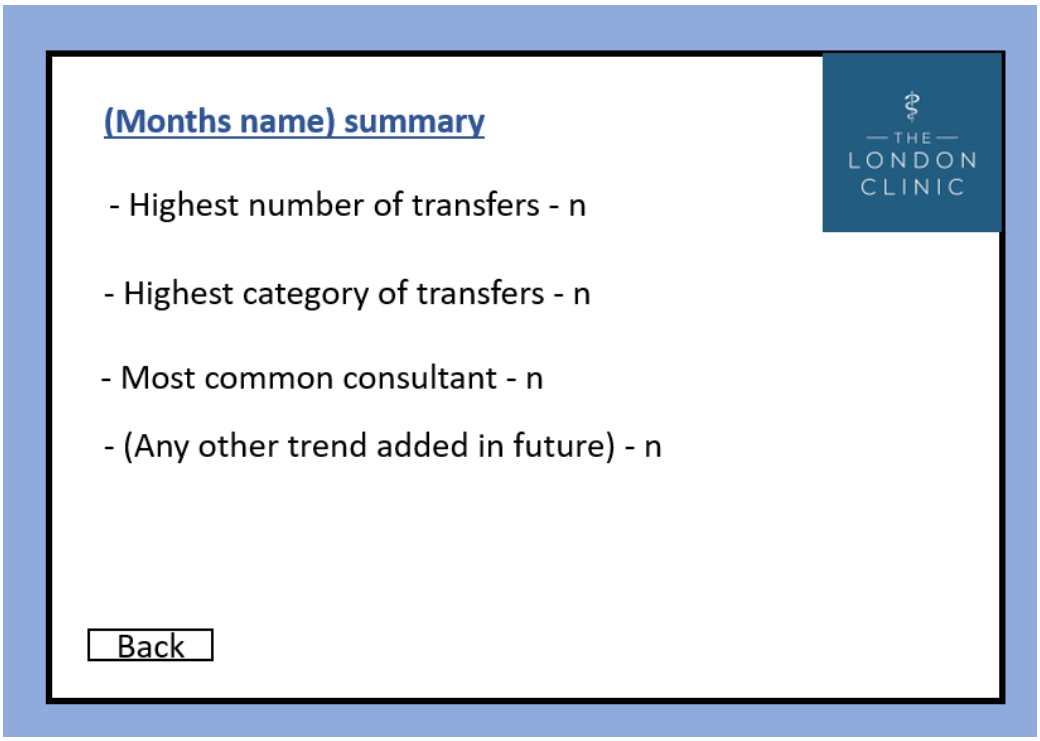
DSW Monthly Graph



Results from a whole months' worth of data will be displayed in graph format here. Each bar will represent a consultant depending on their assigned key. My client wanted to keep The London Clinic logo on this page to show it is official even though there is already a lot of information present on the screen. The graph title again matches the colour of the logo to make the screen more visually appealing.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Graph Summary



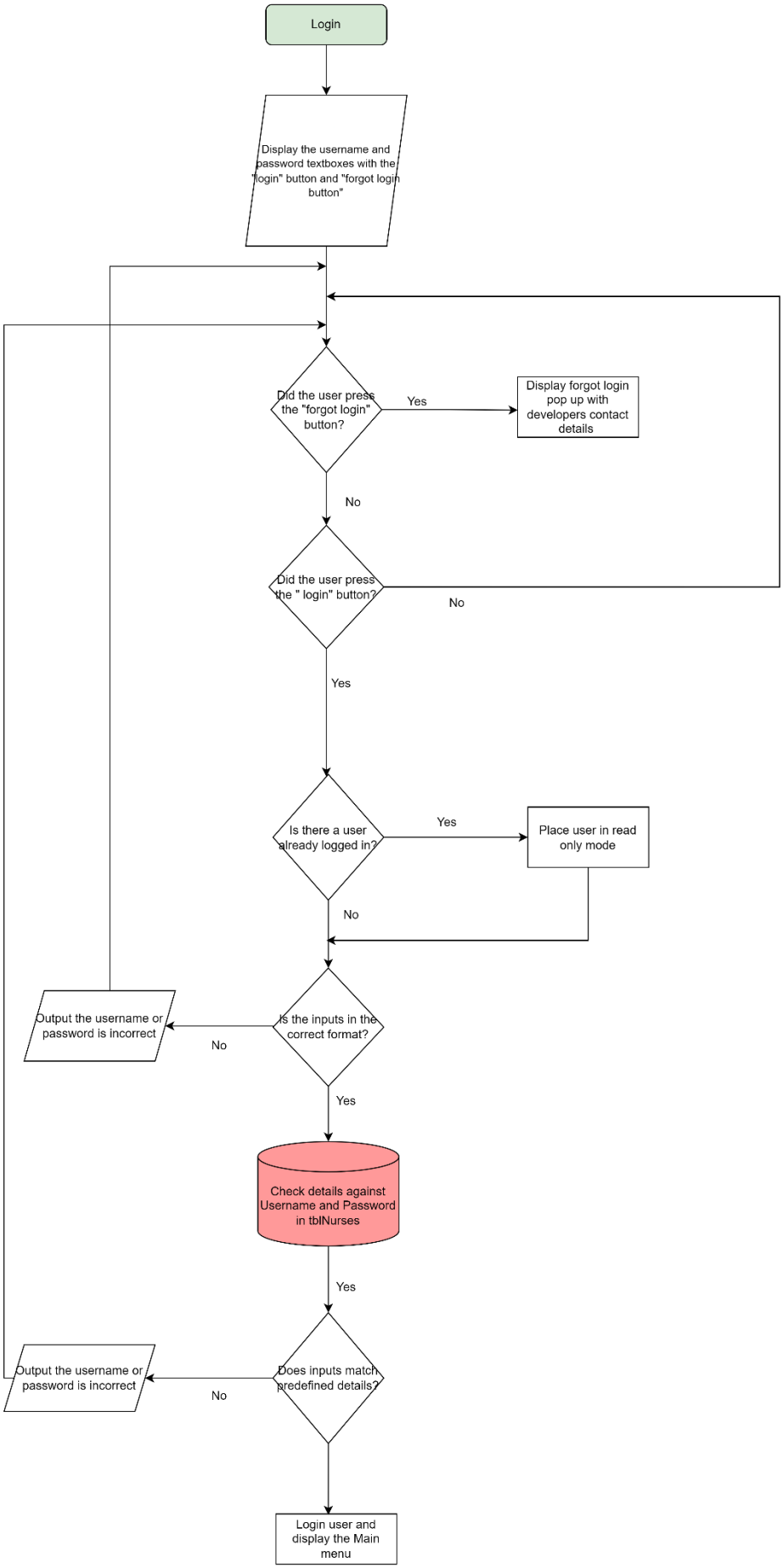
Within this page, is the graph summary in word format. This again highlights main features/trends of the data. These will be in a list going down the page as it easier to view for my client. There will be a blank bullet point that could be used to measure another variable in the future when the program starts being used on the ward (Beta testing). These bullet points will be a “neutral” black colour in contrast to the colour scheme of the software, allowing the user to differentiate between information on the page.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Flow diagrams

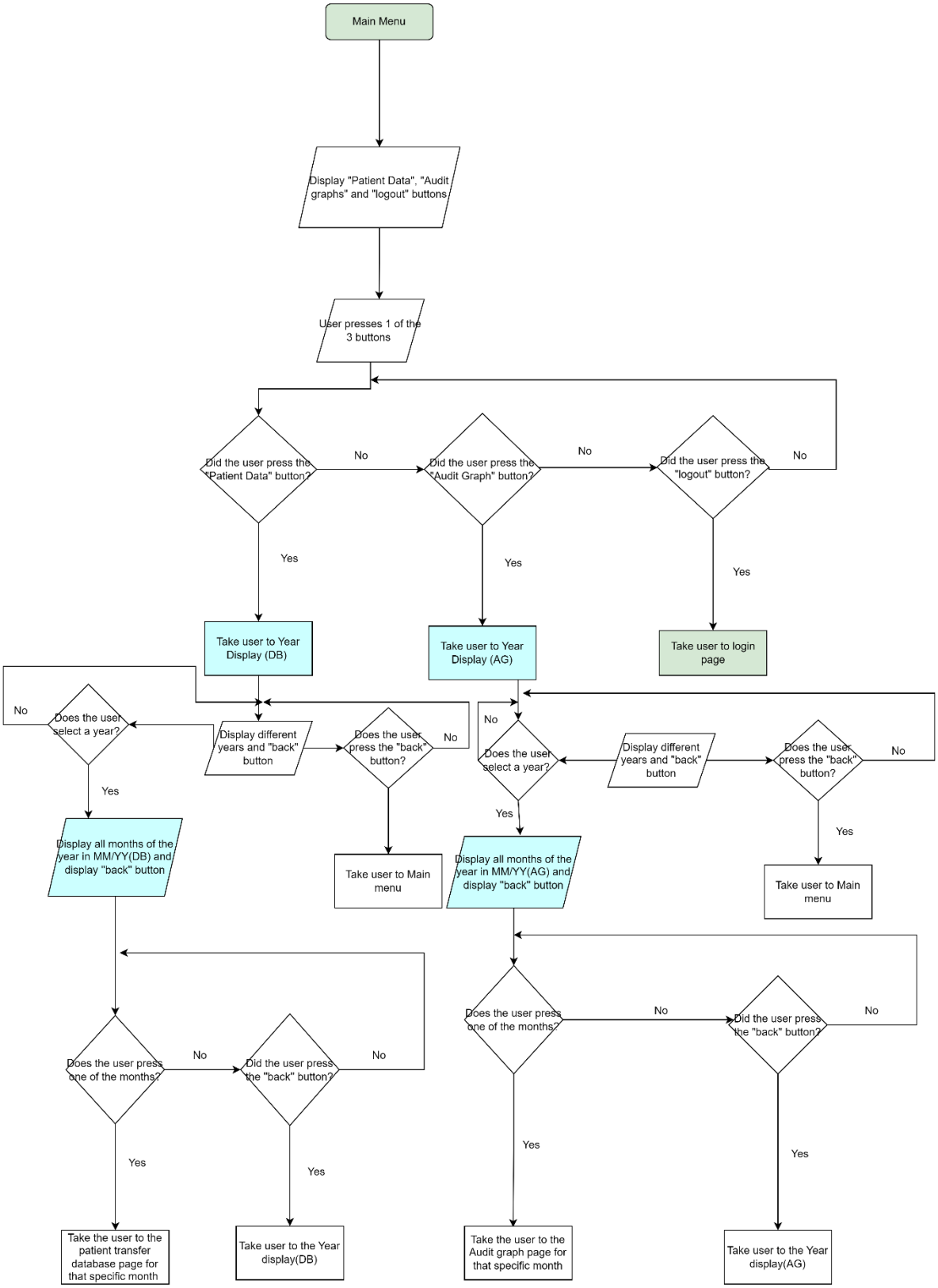
Login

This is the login screen where the user will be prompted to enter their username and password which is then checked against the credentials in tblNurse. If the credentials match the user will gain access to the program and if they don't match, they will be not permitted to enter the Transfer Audit. User will be place in read only mode if another user is already in the program. I designed this login screen in this way as it is a quick and simple solution to gaining access to the program. I could have added a make an account feature where the user can create their own username and password. However, this takes time and is not a key feature for my small sample of users on the ward, therefore I did not implement this function.



Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

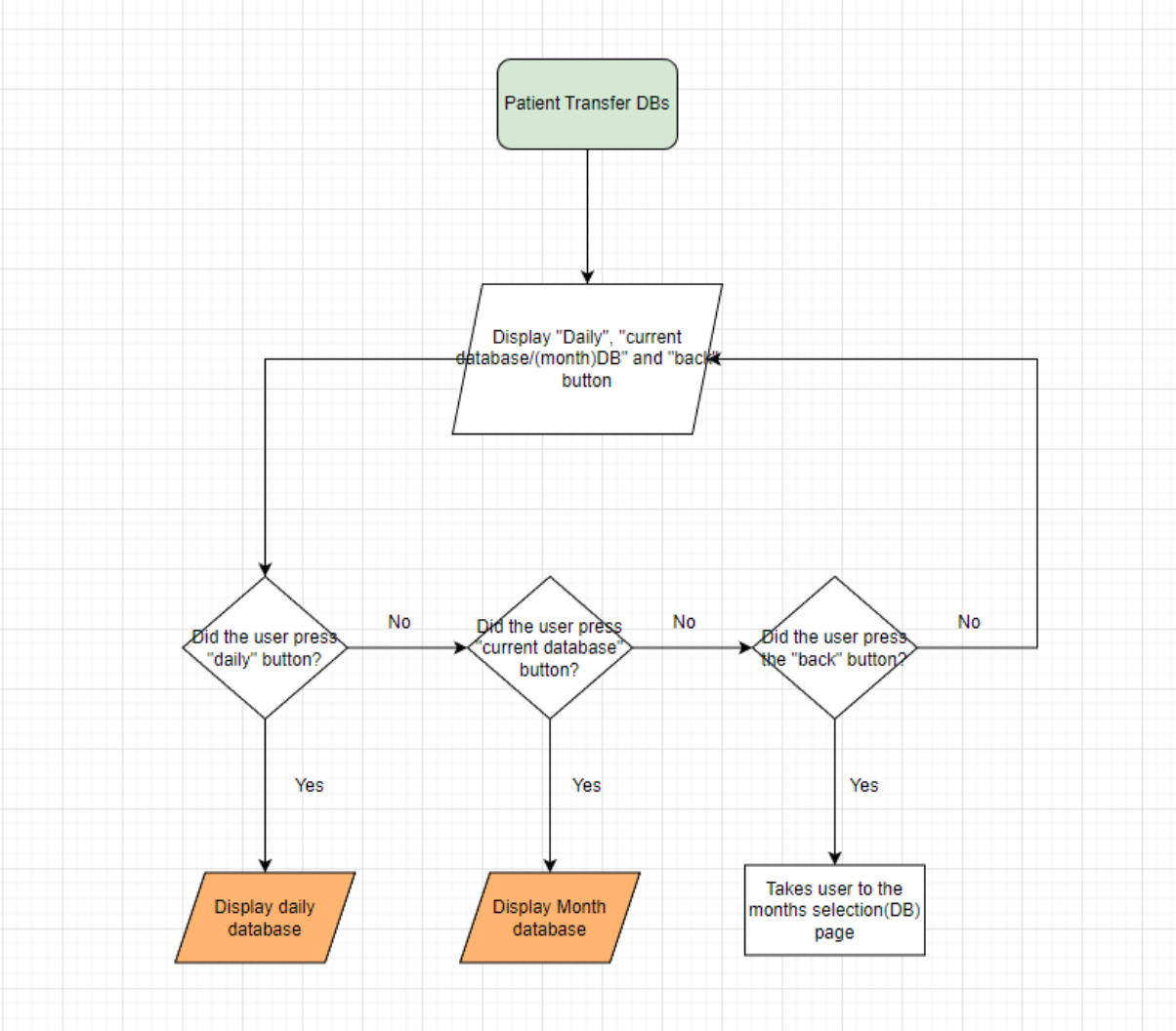
Main menu



Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

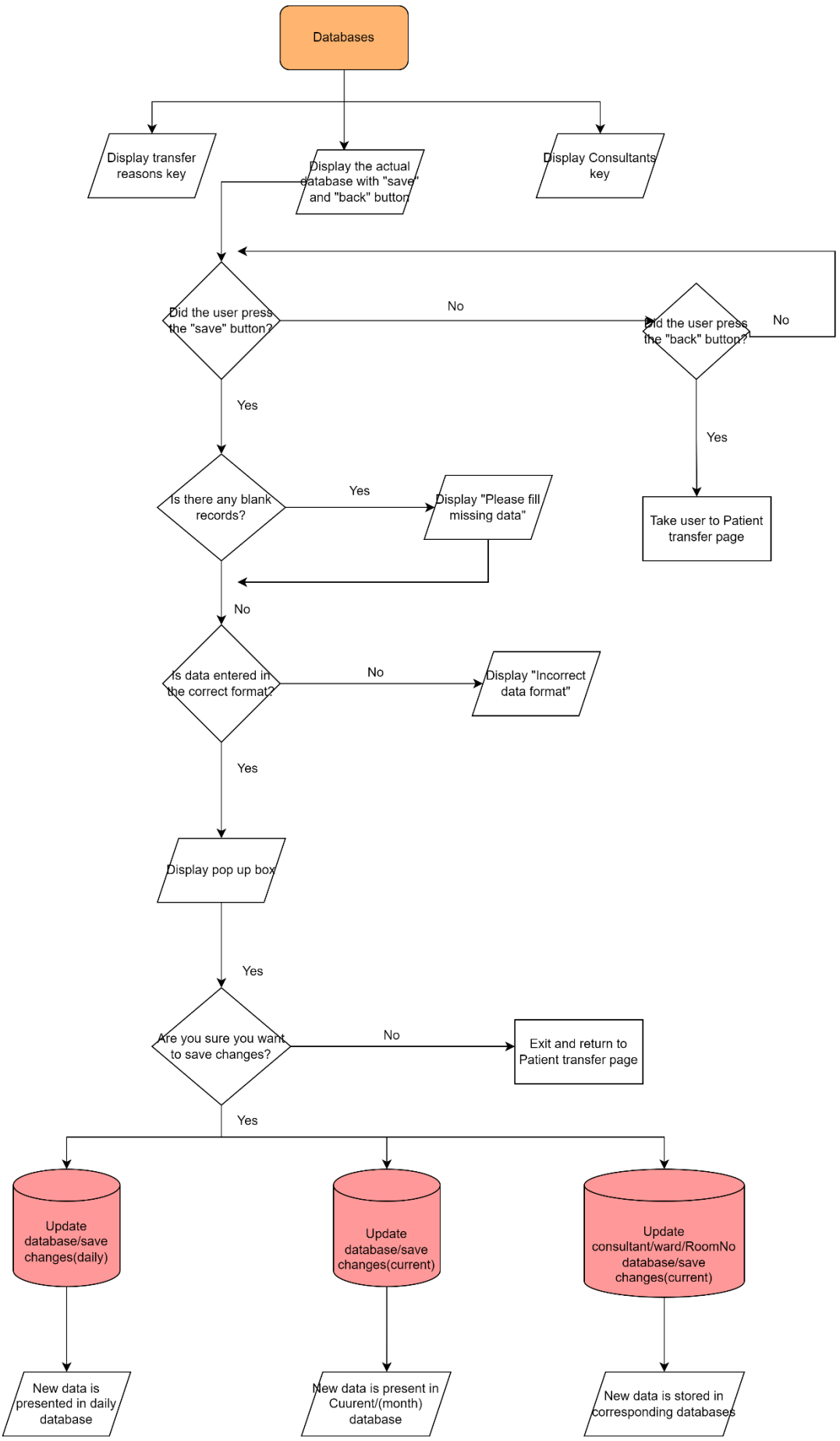
Here is the main menu three widgets are displayed where the user can be directed to the two main purposes of the program or they can logout. Once a decision is made from the two main options the Year page will be presented following the month page. Once a month is chosen the current and daily database is loaded. I have configured the menu in this way because it follows a logical order suitable for any of the nurses to understand. Another way I could have done this, is placing the year and month of the same screen to reduce the time to access the databases/graphs. Although, to follow a chronological pattern that makes sense I am going to keep the year and month selection screen separate.

Patient Transfer sub-menu & Databases



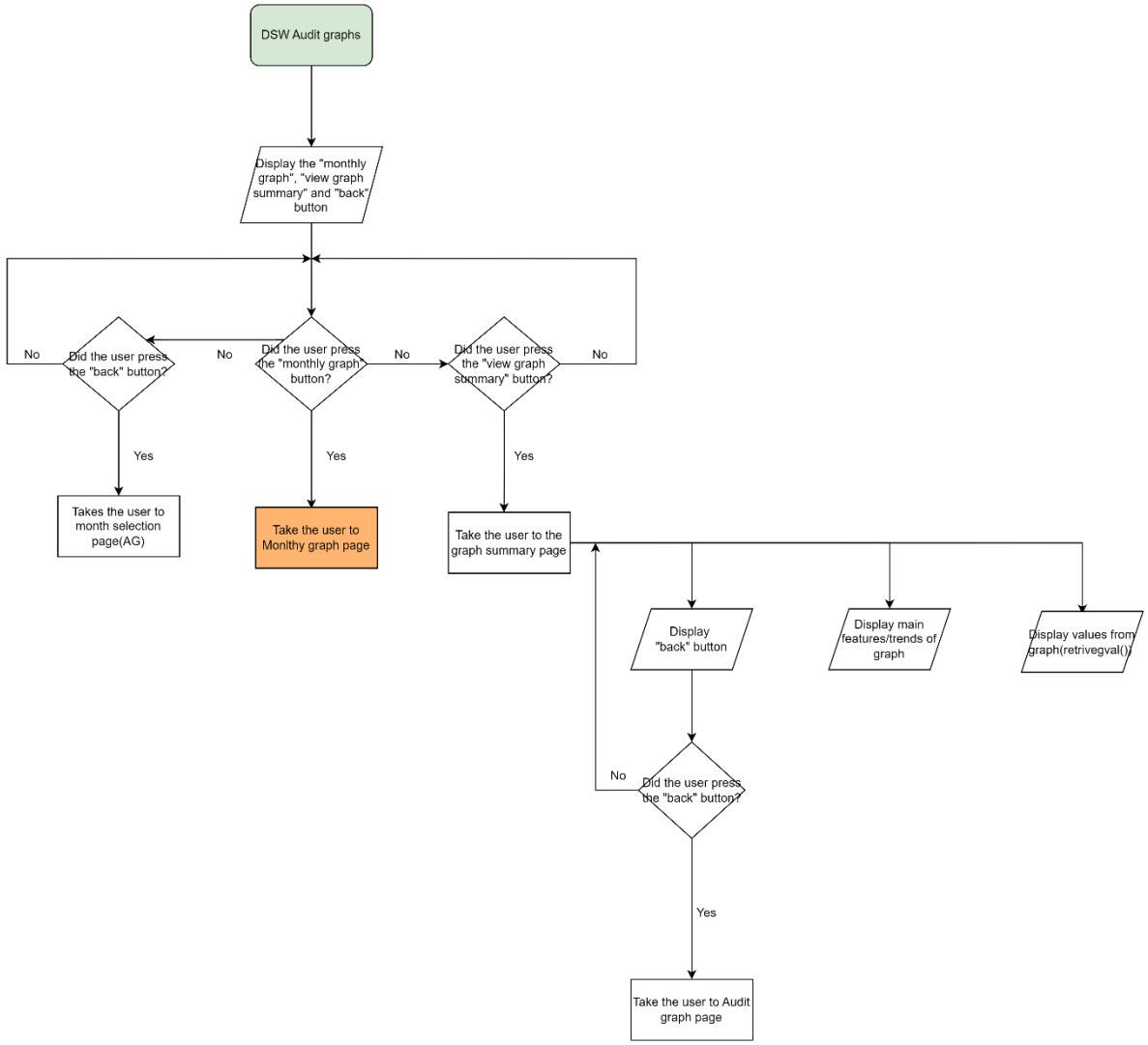
This is the Patient transfer databases screen which contains the daily, current database and back button. Each leading to displaying the databases or going back to the month selection page. I could have had only the current database on this page as this accounts for changes made anytime during the month. Despite this, for specificity I have used a daily database to spread the data out and so patient transfers from each day can be easily identified.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**



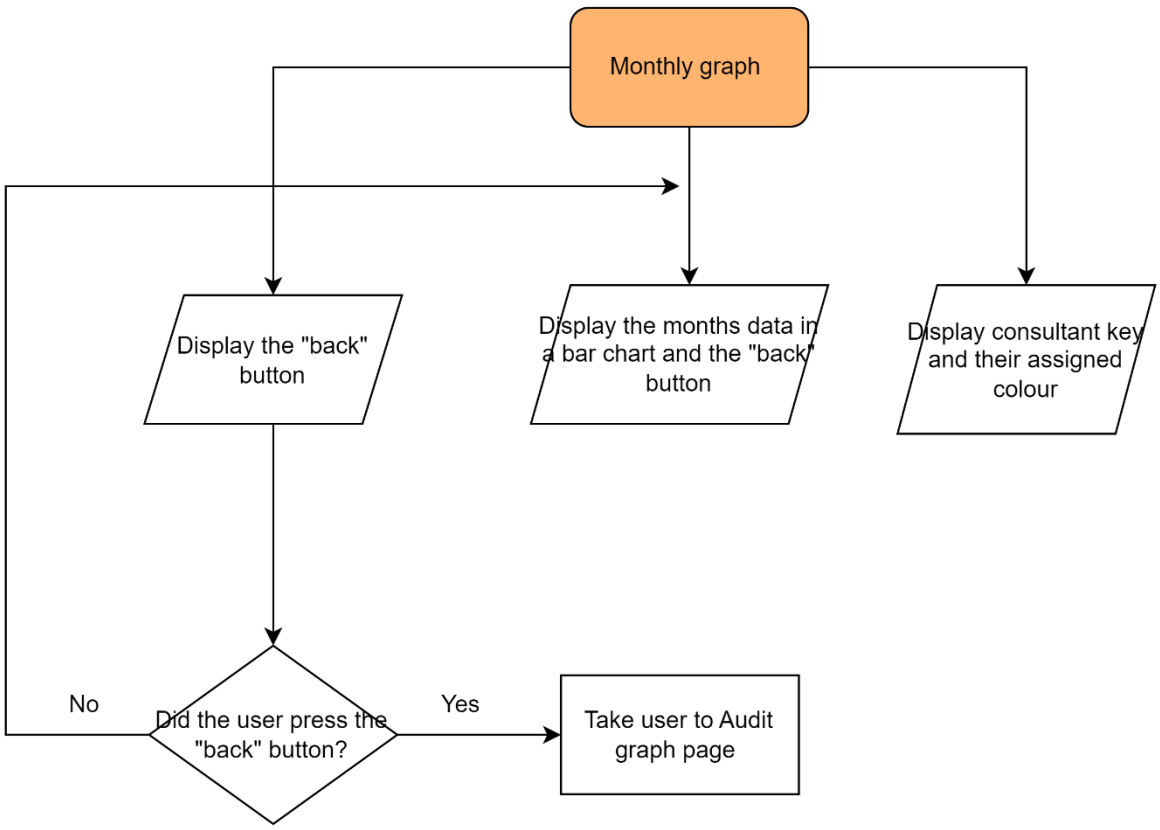
This is the actual databases where the user can edit and add any information. They will be displayed with two widgets save and back. When the database is saved by the user its format is checked otherwise an error message will be displayed. The new data is saved to the database in use and copied to consultant, ward, RoomNo database. I could have made the user input data into a table then save It to a database so the database wouldn't have to be loaded every time a user wanted to enter something. To save programming time, the actual database is editable to the user.

DSW Audit Graph sub-menu and summary



This is the monthly graph page which displays the main widgets for the monthly graph and the graph summary. This allows the user to visually interpret the months patient transfers and any key characteristics are highlighted in the graph summary page. Another way I could have shown this is only having one button for the graph and its characteristics, decreasing programming time. However, to make the user interface more user-friendly I have separated the two, also making it easier for older users to navigate the program for the desired information as they may not distinguish between the two if it was on one screen.

DSW Monthly Graph

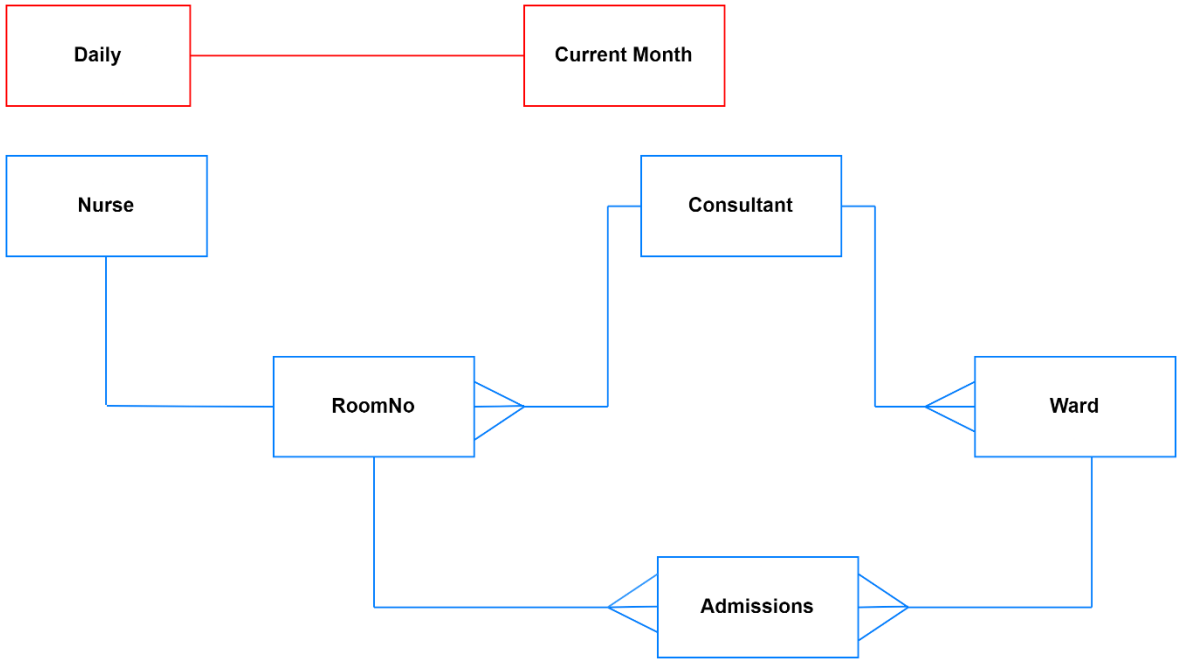


Here is the monthly graph page which displays a bar chart containing all the necessary information for a specific month. (number of transfers on y axis, transfer reasons on x axis) and a back button. There is only one operation which can take place on this page which is when the back button is clicked the user is taken back to the audit graph page. Therefore, there is not an alternative layout of this screen as it is already simplified enough for the user to understand.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Database design

Entity – Relationship diagram



In my program I will have 7 different database tables and 4 relations made up of one-to-tone, one to many and one-to-many, many-to-one relationships. This will be needed as only one nurse can be assigned one room as they can only be on one ward. Only nurses who login in with the correct credentials can be assigned a room(optional as this is not main purpose of the program). However, as the consultant actually takes care of the patient, they can have many rooms on many wards in the hospital. As the rooms and wards of patients can change multiple times, each change needs to be documented as one admission. Therefore, one ward and one room can have many admissions.

The nurse table will already be filled with their credentials as the accounts will be made before the software is created. As a result, the admin usernames and passwords will already be entered into the nurse table. All the other tables will be filled while the system is used by the admins logged in such as the entering the consultants' details. For example, the consultant table will be filled up as soon as the users enter the required consultant details into the daily/current month table.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Entities (and attributes) for the whole system

tblDaily

- ConsultantID (string) PRIMARY KEY
- WardID (varchar)
- RoomNo (integer)
- TimeOfAdmission (integer in time format HH:MM)
- TransferDate (string in date format
- DD/MM/YY)
- TransferTime (integer in time format HH:MM)
- ReasonForTransfer (string)
- Comments (varchar)

tblCurrentMonth

- ConsultantID (string) PRIMARY KEY
- WardID (varchar)
- RoomNo (integer)
- TimeOfAdmission (integer)
- TransferDate (string in date format
- DD/MM/YY)
- TransferTime (integer)
- ReasonForTransfer (string)
- Comments (varchar)

tblNurse

- NurseID (integer) PRIMARY KEY
- Username (string)
- Password (string)

tblWard

- WardID (varchar) PRIMARY KEY
- ConsultantID (string) FOREIGN KEY
- Wardname(speciality) (string)

tblConsultant

- ConsultantID (string) PRIMARY KEY
- Speciality (string)
- RoomNumber (integer)

TblRoomNo

- RoomNo (integer) PRIMARY KEY
- ConsultantID (string) FOREIGN KEY
- NurseID (string) FOREIGN KEY
- TimeOfAdmission (integer in time format HH:MM)

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

tblAdmission

- WardID (varchar) Foreign key
- RoomNo (integer) Foreign key
- TransferDate (string in date format
- DD/MM/YY)
- TransferTime (integer in time format HH:MM)
- Reason for transfer (integer)

Pseudocode

Login

subroutine forgotlogin():

If user presses “forgot login” button then

Display Pop up box

Output “*my contact details*”

else

pass

subroutine login():

UserinProgram(boolean) = 0

usernameExists(boolean) = 0

Display username and password textboxes

Display “login” and “forgot login” buttons

while the user has not pressed any buttons:

Username = username textbox

Password = password textbox

```
end while
if user presses the “forgot login” button then
        run subroutine forgotlogin()
else then
        UserinProgram(boolean) = is there a user already in the program? (going to be false always as not program not linked to a server)

end if

while Username OR Password == “  “
        output(“Please fill in blank fields”)
        run subroutinelogin()
end while


while UserinProgram(boolean) == TRUE
        run subroutinelogin()
else then
        pass

if Username!= 6 char then
        output(“Username is in invalid format”)
else then
        pass
end if
if Password!= 8 char then
        output(“Password is in invalid format”)
else then
        pass
end if
```

```
end while
If “username” exists in the entity “Nurse”
        usernameExists(boolean) == TRUE
else
        usernameExists(boolean) == FALSE

while usernameExists == FALSE
        output(“Username or password is incorrect”)
        run subroutinelogin()
else then
        hide subroutinelogin()
        run subroutineMainmenu()
```

Main menu

```
subroutine Mainmenu():
        Display “Patient data”, “Audit Graphs” and “back” buttons
        While no buttons are pressed:
                wait
        end while
if “Patient Data” button has been pressed then
        run subroutine YearPD()
else if “Audit Graphs” button has been pressed then
        run subroutine YearAG()
else if “back” button has been pressed then
        run subroutine login()
else then
        pass
```

Year Page (PD)

subroutine YearPD ()

Display “2023”,2024”,”2025”, “2026” and “back” buttons

If “2023” ,2024”,”2025”, “2026”, button has been pressed then

run subroutine MonthPD()

else if “back” button has been pressed then

run subroutine Mainmenu()

hide subroutine YearPD()

else then

pass

end if

DailyDatabase = 0

CurrentMonth = 0

Month Page

subroutine MonthPD()

Display “01/YY”, “02/YY”,”03/YY”,”04/YY”,”05/YY”,”06/YY”,”07/YY”,”08/YY”,”09/YY”,”10/YY”,”11/YY”,”12/YY” and “back” buttons

If “DD/YY(any)” button has been pressed then

DailyDatabase = “Daily” entity for that year and month (“DD/YY”)

CurrentDatabase = “CurrentMonth” entity for that year and month(“DD/YY”)

run subroutine PatientData()

else if “back” button has been pressed then

run subroutine YearPD()

hide subroutine MonthPD()

else then

```
                pass
end if
```

Patient Databases page

```
subroutine PatientData()
        Display “Daily” , “Current/month database” and “back” buttons
If “Daily” button has been pressed then
                Display DailyDatabase
                Display “save” and “back” button
                run subroutine DatabaseChange()
else if “Current/month database” button has been pressed then
                Display CurrentMonth
                Display “save” and “back” button
                run subroutine DatabaseChange()
else if “back” button has been pressed then
                run subroutine MonthPD()
                hide subroutine PatientData()
else then
        pass
```

```
subroutine DatabaseChange()
while the user has not pressed any of the buttons:
        wait
end while
if user pressed the “back” button then
        run subroutine PatientData()
else then
```

```
        pass

while user pressed the “save” button
        output “Are you sure you want to save the changes?”
        Display “Yes” button
        Display “No” button
end if

If “Yes” button is pressed then
        run subroutine CorrectFormat()
        run subroutine TransferCount()
        Update changed record in “daily”/”current month database” entity
        Update “consultant”/”ward”/”roomNo” entity
        output “changes have been saved successfully”
else then
        run subroutine DatabaseChange()
```

Subroutine CorrectFormat()

Format(Boolean) == FALSE

ConsultantKey = [LAI , BHG, EL , WO, DUG, AHM, WEST, CRAW, ZAK, HEN, SAW, PRI, HAM, EAS, FERG, MIT, LIM]

If consultant field from entity “Daily” or “CurrentMonth” != ConsultantKey

        Format(Boolean) = FALSE

        output “Table not in valid format”

else then

Format(Boolean) == TRUE

end if

while date field =! DD/MM/YY OR time field =! HH/MM am/pm from entity “Daily” or “CurrentMonth” then

Format(Boolean) == FALSE

output “Table not in valid format , time and date is not correct”

else then

Format(Boolean) == TRUE

end if

if ward field != string from entity “Daily” or “CurrentMonth” then

Format(Boolean) == FALSE

output “Table not in valid format, ward needs to be characters only”

else then

Format(Boolean) == TRUE

end if

if reason field from entity “Daily” or “CurrentMonth” 1<= AND reason >=6 then

Format(Boolean) == TRUE

else then

Format(Boolean) == FALSE

output “Table not in valid format, reason field is not correct”

if Format(Boolean) == TRUE

```
                end()

else then
        run subroutine CorrectFormat()
```

```
mostCommon = 0

subroutine TransferCount()
countMed = 0
countReq = 0
countLDis = 0
countCreq = 0
 countSoc = 0
 countOth = 0
totalcountMed = 0
totalcountReq = 0
totalcountLDis = 0
totalcountCreq = 0
totalcountSoc = 0
totalcountOth = 0

if int in “reason” field in “daily”/”current month database” entity == 1 then
        totalcountMed = countMed + 1
```

```
else if int in “reason” field in “daily”/”current month database” entity == 2 then
        totalcountReq = countReq + 1
else if int in “reason” field in “daily”/”current month database” entity == 3 then
        totalcountLDis = countLDis + 1
else if int in “reason” field in “daily”/”current month database” entity == 4 then
        totalcountCreq = countCreq + 1
else if int in “reason” field in “daily”/”current month database” entity == 5 then
        totalcountSoc = countSoc + 1
else if int in “reason” field in “daily”/”current month database” entity == 6 then
        totalcountOth = countOth + 1
else then
        pass

mostCommon = max(totalcountMed, totalcountReq, totalcountLDis, totalcountCreq,
totalcountOth)
```

Year page (AG)

```
subroutine YearAG()

                Display “2023”,2024”,”2025”, “2026” and “back” buttons
```

If “2023” ,2024”,”2025”, “2026”, button has been pressed then

run subroutine MonthAG()

else if “back” button has been pressed then

run subroutine Mainmenu()

hide subroutine YearAG()

else then

pass

end if

Month Page (AG)

subroutine MonthAG()

Display “01/YY”, “02/YY”,”03/YY”,”04/YY”,”05/YY”,”06/YY”,”07/YY”,”08/YY”,”09/YY”,”10/YY”,”11/YY”,”12/YY” and “back” buttons

If “DD/YY(any)” button has been pressed then

run subroutine MonthGraph = graph for that year and month (“DD/YY”) (make bar graph in matolib – consultant = a colour and total for each reason on y axis, reasons on x axis) (plot graphs module)

run subroutine AuditGraphs()

else if “back” button has been pressed then

run subroutine YearAG()

hide subroutine MonthAG()

else then

pass

end if

Audit Graph Page

subroutine AuditGraphs()

Display “(month)graph” , “view graph summary” and “back” buttons

If “(month)graph” button has been pressed then

run subroutine MonthGraph()

else if “view graph summary” button has been pressed then

run subroutine GraphSummary()

else if “back” button has been pressed then

run subroutine MonthAG()

(Month)Graph page

subroutine MonthGraph()

import schedule

import time

schedule to end of every month

if time>= 8pm then

run subroutine TransferCount()

colour consultants on (MATOLIB)

totalcount… displayed on graph (MATOLIB)

output “Medical reason”, “Patient Request” , “Late Discharge”, “consultant request”, “social reason” , “other” on y axis (library MATOLIB code)

Display MonthGraph()

else then

```
        pass
Display “back” button
if “back” button has been pressed then
        run subroutine AuditGraphs()
else then
        pass
```

Graph summary page

```
subroutine GraphSummary()
Display “back” button
if “back” button has been pressed
        run subroutine AuditGraphs()
else then
        pass
output = “highest number of transfers is ” +  mostCommon
output = “Most common consultant” +  (library MATPLOTLIB code)
output = “Highest category of transfers” + (library MATPLOTLIB code)
```

```
subroutine time()
import schedule
import time
schedule everyday
while time !>= 8pm then
        pass
```

else then

Copy “Daily” entity to “current month database” entity”

DROP “Daily” entity

CREATE new table “Daily” entity

Variable table

Name	Data Type	Scope	Purpose	Validation	Sanitization	Subroutines it is used in
UserinProgram	Boolean	Local	Variable is true when there is a user currently in the audit which prevents new users logging in	-	-	Login
usernameExists	Boolean	Local	Stored as TRUE if username exists in the Nurse entity	-	-	Login
Username	String	Local	Holds the users inputted username	Must be 6 characters long and match username in tblNurse	First letter is capital	Login
Password	String	Local	Holds the users inputted password	Must be 6 characters long and match username in tblNurse	First letter is capital	Login
Format	Boolean	Local	Remains false when data entered by the user into the database is in the wrong format	-	-	CorrectFormat
DailyDatabase	String	Global	Holds the daily database for a specific month and year which can be used in all subroutines	-	-	MonthPD PatientData
CurrentMonth	String	Global	Holds the Months database for the month selected by the user which can be used in all subroutines	-	-	MonthPD PatientData

totalcountMed	Integer	Local	Stores total number of medical reasons for patient transfers in the month database	-	-	TransferCount
totalcountReq	Integer	Local	Stores total number of patient requests for patient transfers in the month database	-	-	TransferCount
totalcountLDis	Integer	Local	Stores total number of late discharges for patient transfers in the month database	-	-	TransferCount
totalcountCreq	Integer	Local	Stores total number of consultant requests for patient transfers in the month database	-	-	TransferCount
totalcountSoc	Integer	Local	Stores total number of social reasons for patient transfers in the month database	-	-	TransferCount
totalcountOth	Integer	Local	Stores total number of other reasons for patient transfers in the month database	-	-	TransferCount
countMed	Integer	Local	Allows for the addition of each medical reason in the month database (+1)		+ 1 each time	TransferCount
countReq	Integer	Local	Allows for the addition of each patient request in the month database (+1)		+ 1 each time	TransferCount
countLDis	Integer	Local	Allows for the addition of each late discharge in the month database (+1)		+ 1 each time	TransferCount
countCreq	Integer	Local	Allows for the addition of each consultant request in the month database (+1)		+ 1 each time	TransferCount
countSoc	Integer	Local	Allows for the addition of each social reason in the month database (+1)		+ 1 each time	TransferCount
countOth	Integer	Local	Allows for the addition of each other reason in the month database (+1)		+ 1 each time	TransferCount
mostCommon	integer	Global	Calculates and stores the option for the highest number of patient transfers in the month	-	-	TransferCount GraphSummary

			which can be used in all subroutines			
Schedule	String	Local	Allows the daily database to refresh at the same time everyday	-	-	Time
Time	Integer	Local	Stores current time of the system clock	-	24-hour clock	Time

Types of testing

In my program I will carry out alpha, beta, black box and white box testing.

White Box testing

I will use white box testing to test the behaviour of my code and how it responds to valid and invalid inputs ensuring the security of my program. For example, the correct output message should be displayed if the user enters an invalid username during logging into the software.

Black Box testing

Black box testing will be used to experience the program from the user's point of view and this will be done by the testing each subroutine of the program as well as making sure all the buttons are displayed, which when clicked, produce the desired output such as moving between different screens.

Alpha testing

I will perform alpha testing myself during the production of my code, testing it throughout so problems can be solved before it cascades to other functions also saving time when the program is finished as less amendment will need to be made.

Beta testing

I will use beta testing by giving my program to my client to try and navigate the system while they decide if it is suitable for its purpose. This will simulate how a user will interact with my program and any errors can be picked by them that I would not have noticed in development. My program would also be given to the other nurses on my clients ward to experiment with, so I can get informative feedback by a variety of users.

Acceptance testing

Acceptance testing will be carried out throughout development such as on each subroutine and then on the whole program. After development I will look back at my success criteria and my client will examine my final program if I have met the necessary criteria, helping my client to determine if it suits her needs well enough.

Test Table

White Box Testing – Testing text inputs

Input	Valid	Invalid	Boundary
Username on login	Admin1 Admin2 Admin3 Only these usernames will match the usernames stored in the database (normal)	Admins Adminsss Scooby12 BrianMelow Makes sure username only has 6 characters and is a valid username otherwise they will not be valid.	Hooooooooooooo The program will stop inputs that are not 6 characters long
Password on login	London12 Clinic99 Ensures only valid passwords are allowed which are stored in the database	hospital23 doggo123 theLondonClinc London14 Makes sure password starts with a capital letter otherwise will go against validation rules. Makes sure only correct passwords are valid.	Shellerssss Bacteriaaaa TheLondonClinc Tests that only passwords which are 8 characters long are allowed by the validation rules as this is the maximum of characters the user can have.
Consultant Key in the database	BC TW FD	@##& /// @@@@	TWC OPP CLM

	Makes sure only initials of the consultant is allowed to be entered into the databases	Makes sure erroneous data cannot be entered and initials are only alphabetical	Allows only up to 3 characters to be entered by the validation rules.
What ward in the database	4 5 6 Ensures ward numbers can be inputted by the user	sixth floor “ ” Makes sure there is validation for the ward field as only numbers can be entered as ward IDs.	66 15 2 Ensures users can enter all possible digits if needed
Room number in the database	303 444 555 Allows room valid numbers to be inputted by the user	Addissdfsf “ ” Ensures room numbers are digits and there are no letters being valid or empty characters.	1000 -1 78 4 Room numbers can only be 3 digits
Date in the database	07/12/22 13/01/23 14/05/24 Date entered in DD/MM/YY format	7/6/22 00/1/22 /// 12.03.22 Invalid dates to be recognised by the validation rules and it must contain integers	133/4/22 4/4/4 0/0/0 Tests that the date is in correct format with two integers between each /
Time in the database	10:23 12:04 18:12 All ways of entering time are valid.	7 30 6:15 12 Makes sure the time is entered in only the HH:MMam/pm format as it is clear and concise for the user to read.	00:01 23:59 Tests for the 24-hour clock period as it is the range for which data can be inputted as.
Reason in the database	1 2	Edfinslfd @@	12 3

	5 6 Only integers are valid.	() “” Ensures only numbers are entered as valid inputs as erroneous data is prevented.	4 6 7 Tests reasons can only be single digits and maximum 6
Comments in the database	Late discharge after 6 FD request Allows all inputs of letters and numbers	There is no validation as the user can add any extra information required for that patient transfer	Gggggggggghg Looooooool Supports that any data can be entered as no validation rule here

Test to do	**Who will carry it out?**	**What should be expected?**	**What could go wrong?**
Go through the screens from options in the main menu and by pressing the “back” buttons	The developer (me)	Then user should seamlessly move through screens without any problems after a button is pressed and when then “back” button is pressed the user should be taken to the page they were previously on.	The user may not move to another screen when a button is pressed because of a coding error(pages may not link in main program)
Wait on main menu while no button is pressed	The developer (me)	Nothing should happen as no buttons have been selected and program should wait in the loop.	Runtime error may be displayed if no buttons are clicked and loop is incorrect.
Pop up box if forgot login button is pressed	The developer(me)	A pop-up box should be displayed containing my contact details for more information	Pop up box may not be displayed due to error in the code or information about me may be incorrect

		about the admin's login	due to human error when programming.
Daily database is created every day at 8pm and its content moved to monthly database	The developer (me)	All the data from the daily database should be carried over to the current month database and a new daily database should be created also deleting the previous one.	All data may not transfer to current month database due to format errors or data being lost. Daily database may not delete before making a new one because the refresh loop may be incorrect such as syntax errors.
Data from current month database should be displayed in graph format	The developer(me)	At the end of the month the data from the current month database should be transferred into a graph for users to visualise and spot trends.	Data may be entered in the graph in the correct format due to errors in the program or the loop may be incorrect as there are different number of days in each month so data may be retrieved on the wrong day.
Summary of results from graph should be made	The developer(me)	Summary page should be created highlighting trends and key features of the graph	Wrong data may be interpreted as the bar chart contains a lot of information

Black box testing

Test to do	Who will carry it out?	What should be expected?	What could go wrong?
Check if a user can login to the program	My client	Should be able to login successfully with the correct credentials stored in the Nurse entity	My client could forget her password and has to contact me which wastes valuable time or there is logic errors that I have made when calling or writing to the database causing the login to fail.
Check if a user can enter data with ease	Any of the staff on the ward	Easily enter data without any hassle and when the database is saved patient transfers should update and display a successful message.	May be format errors when entering data due to syntax errors in the program or for older users who are less able with technology may get overwhelmed with the information on the screen so then the software would need to be made more user friendly. Data may not save if I made any errors in development.
Check if user can logout	Any of the staff on the ward	User should be taken back to the login page and user should no be in the program anymore. Data should not be editable.	User may not be directed to the login page if there is a syntax error as the correct subroutine may not run. If login page takes too long to load the user may hit a runtime error and the program could crash.
Check if user can press each button and it gives the desired output	The developer (me)	After each button is pressed the correct output is displayed or the correct	There could be a logical error if the incorrect databases are displayed

		subroutine is run. For example, choosing a month and year should bring up the data for the patient transfers of that time period.	because the code doesn't call the correct databases for that specific time period. Some buttons could run the wrong subroutine if I had made that mistake in the program.
Check the correct keys are displayed on the page where data needs to be entered into a database	The developer (me)	When any of the database are clicked the consultant key and the transfer reasons key should be displayed around the page as well as them being colour coordinated.	The consultants' keys may be assigned the wrong colour due to human error while I am making the software. In addition, the layout of the page could be set out incorrectly, reducing its user-friendliness making it unnecessarily difficult for busy staff to enter patient data.

Development

Development Plan

1) Firstly, I need to create my relational database. This will be done by using SQLite , which can directly link with the main python program. The NurseID table will be filled with the login credentials created by me and all the other fields and the different tables will be filled as the system is used.

2) **First Prototype**
   Secondly, I will make my first prototype where I will create the GUI of my software so I can visualise how my program will operate by importing the AppJar library to python.

3) **Second Prototype**
   For my second prototype I will create the fundamentals of my program by making the main modules such as (login, main menu, Patient Transfer Databases, Audit Graph page, Year page and Month page, databases page, graph page). I will also create my monthly graph by importing the Matolib library to python.

4) **Third prototype**
   Finally, my third prototype will consist of the summary page from the trends of the monthly graph and if I have time, I will try add a pre-loading function to pages so important information can load quicker for the nurses.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

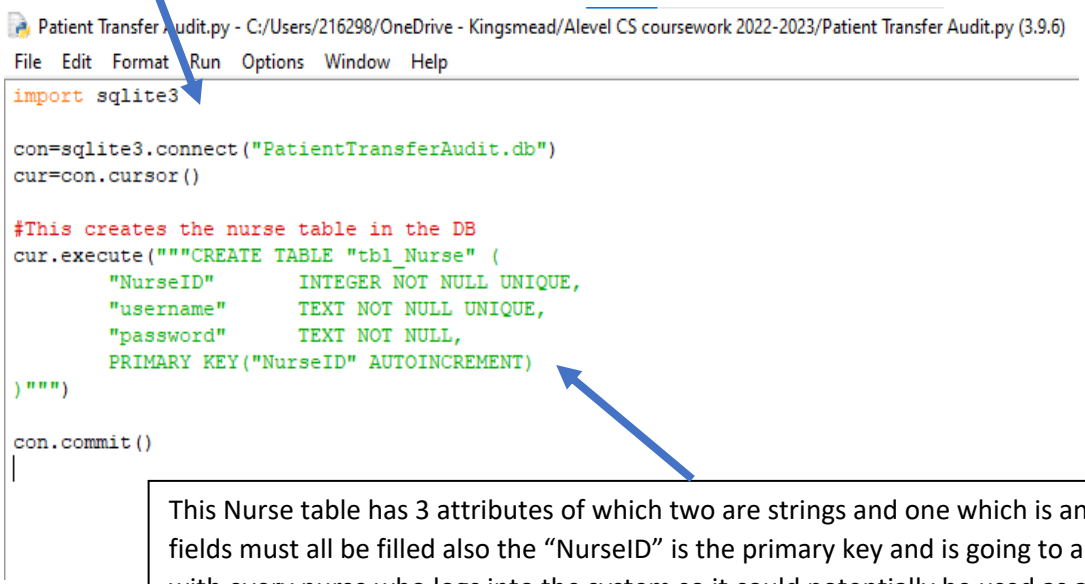
Database Setup

To set up my databases I am going to be using SQLite and storing the tables in DB browser. This is the most suitable strategy for my project as my program is localised to one PC and does not need to be accessed by multiple users from different terminals, that are all connected to one server.

Patient Transfer Audit database

tbl_Nurse

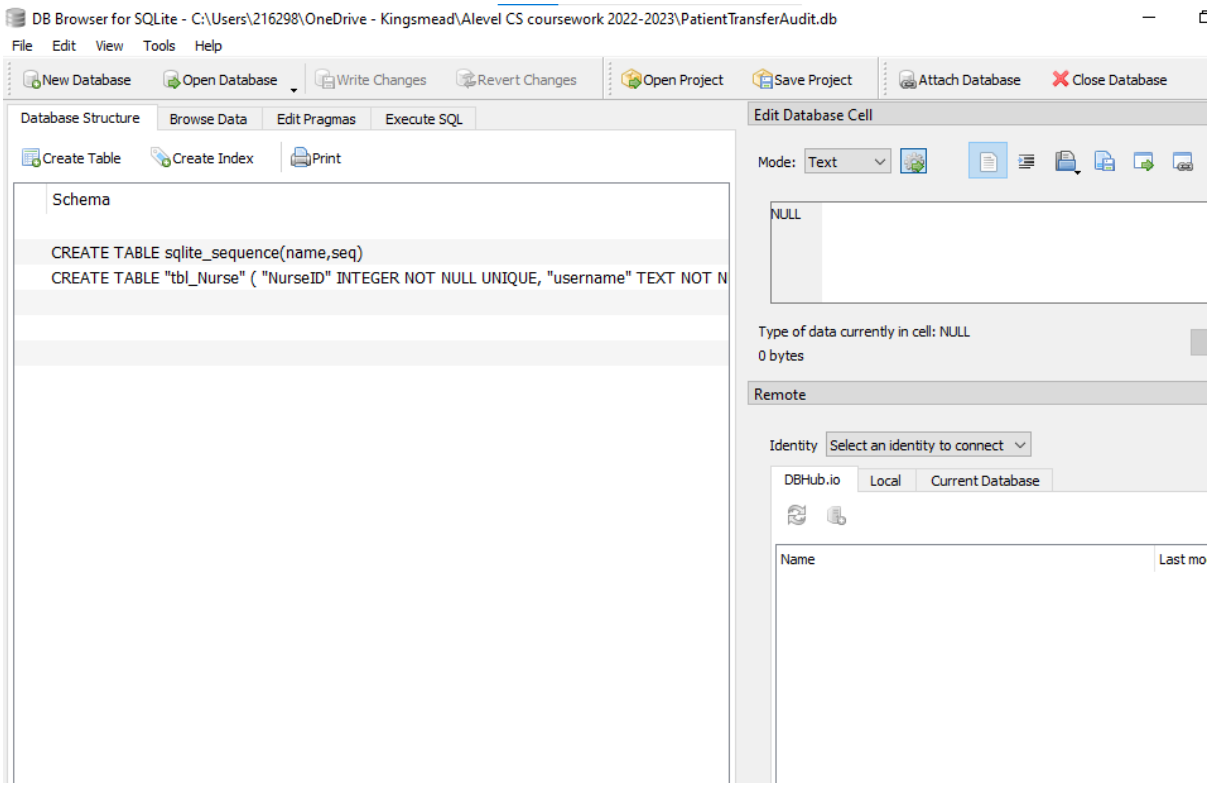
This is where SQLite3 is imported and the SQL file "PatientTransferAudit.db" is connected to the python file. The cursor is stored the variable "cur" allowing the database to be edited.



Here the nurse table has been created where the nurse's login details are stored.

This Nurse table has 3 attributes of which two are strings and one which is an integer. These fields must all be filled also the "NurseID" is the primary key and is going to autoincrement with every nurse who logs into the system so it could potentially be used as an access log in the future if needed. The "username" field will be unique as each username will be different making it easier to identify the nurse who has been using the software. The Nurses credentials are also stored in plaintext which may go against some of the staff's standards of security of sensitive information. Therefore, in the future I could encrypt the staff's passwords so the tables are filled with encrypted data, meaning the software is more secure to any data leakage.

This proves the table has been created successfully.



When I ran the code again this error popped up:

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/216298/OneDrive - Kingsmead/Alevel CS coursework 2022-2023/P
atient Transfer Audit.py
Traceback (most recent call last):
  File "C:/Users/216298/OneDrive - Kingsmead/Alevel CS coursework 2022-2023/Pati
ent Transfer Audit.py", line 7, in <module>
    cur.execute("""CREATE TABLE "tbl_Nurse" (
sqlite3.OperationalError: table "tbl_Nurse" already exists
>>>
```

To fix this error I placed the created nurse table in a function so it can be created again and repeated this for all my other tables in my database.

Patient Transfer Audit.py - C:/Users/216298/OneDrive - Kingsmead/Alevel

File Edit Format Run Options Window Help

```
import sqlite3

con=sqlite3.connect("PatientTransferAudit.db")
cur=con.cursor()

#This creates the nurse table in the DB
def setupDB():
    cur.execute("""CREATE TABLE "tbl_Nurse" (
        "NurseID"       INTEGER NOT NULL UNIQUE,
        "username"      TEXT NOT NULL UNIQUE,
        "password"      TEXT NOT NULL,
        PRIMARY KEY("NurseID" AUTOINCREMENT)
)""")

con.commit()
```

Tbl_Daily

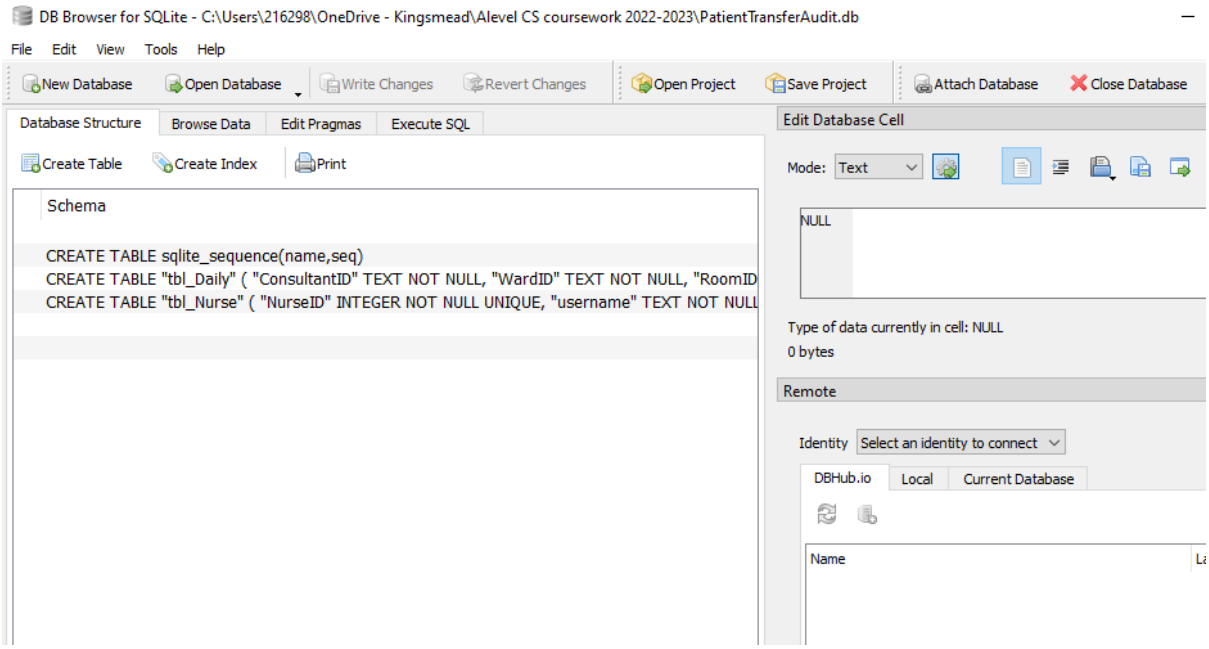
```
    con.commit()
#This creates the Daily audit table in the DB
    cur.execute("""CREATE TABLE "tbl_Daily" (
            "ConsultantID"      TEXT NOT NULL,
            "WardID"    TEXT NOT NULL,
            "RoomID"    TEXT NOT NULL,
            "TimeOfAdmission"   time NOT NULL,
            "TransferDate"      date NOT NULL,
            "TransferTime"      time NOT NULL,
            "ReasonForTransfer" INTEGER NOT NULL,
            "Comments"  TEXT NOT NULL,
            PRIMARY KEY("ConsultantID")
            )""")
    con.commit()
```

The daily audit table has been created where all information that needs to be filled by the nurses about a patient transfer will be held.

Here each field has to be filled and I found out that in SQLite3 fields can be stored as what they are as long as they are being shown. For example, “TransferDate” is where a date is required so it can be stored as a date data type indicating to the viewer what exact format that field is in. The “ReasonForTransfer” field is stored as an integer because the user can choose between 1-6 to select a reason for why the patient is being transferred to a different ward/room.

This shows that the daily table has been created successfully.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**



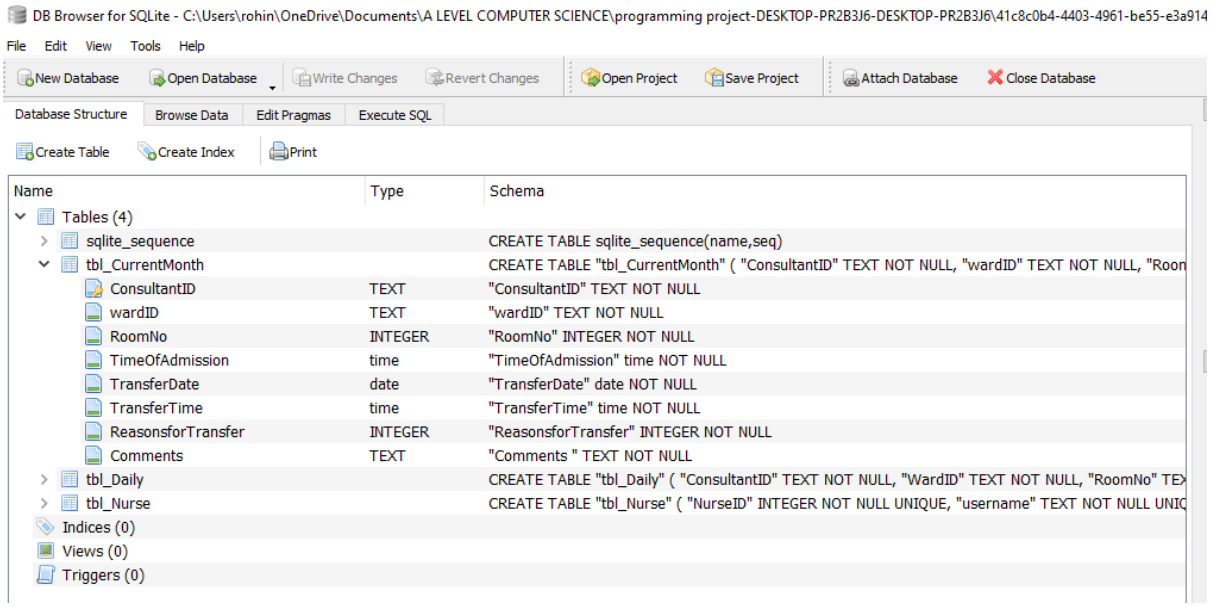
tbl_CurrentMonth

```
#This creates the CurrentMonth table in the DB
    cur.execute("""CREATE TABLE "tbl_CurrentMonth" (
        "ConsultantID"  TEXT NOT NULL,
        "wardID"        TEXT NOT NULL,
        "RoomNo"        INTEGER NOT NULL,
        "TimeOfAdmission"       time NOT NULL,
        "TransferDate"  date NOT NULL,
        "TransferTime"  time NOT NULL,
        "ReasonsforTransfer"    INTEGER NOT NULL,
        "Comments "     TEXT NOT NULL,
        PRIMARY KEY("ConsultantID")
        )""")
    con.commit()
```

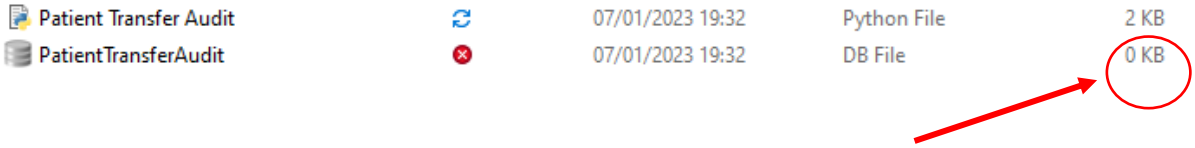
The CurrentMonth table has been made where again all information that needs to be filled by the nurses about a patient transfer will be held.

The CurrentMonth table is the same as the daily table as this is where data that may need to accessed on the same day, is stored. This table will be empty for now but fill up as more users enter data about patients.

This shows that the CurrentMonth table has been created successfully.

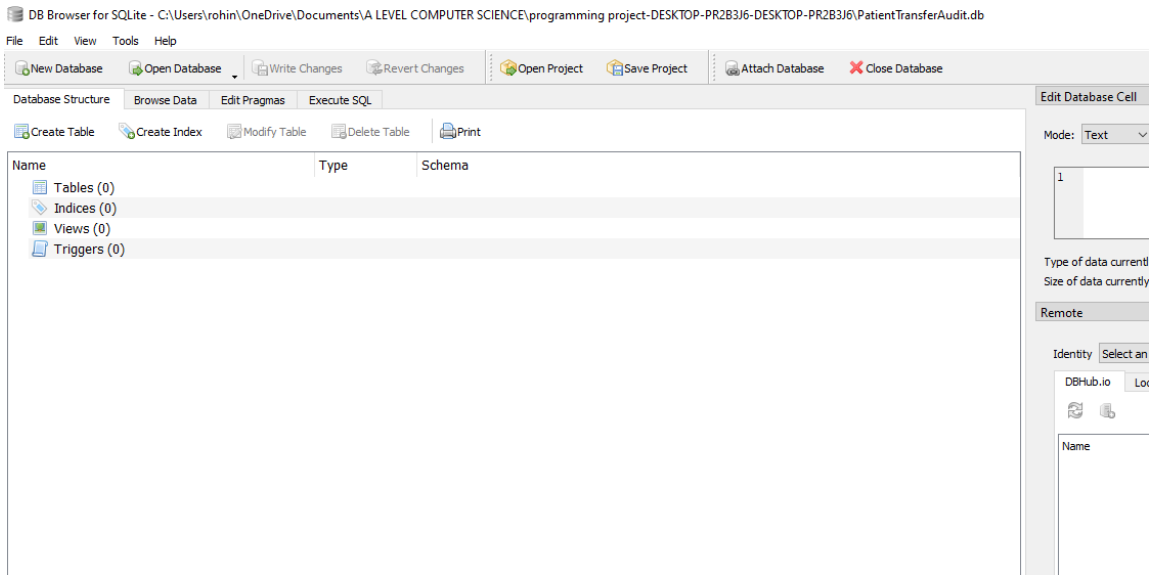


All of a sudden while I tried creating my other tables and saving it to the same database file it wasn't appearing. Therefore, I deleted the database file and ran the program again but now the database file made was 0KB and had nothing in it.



Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

The blank database



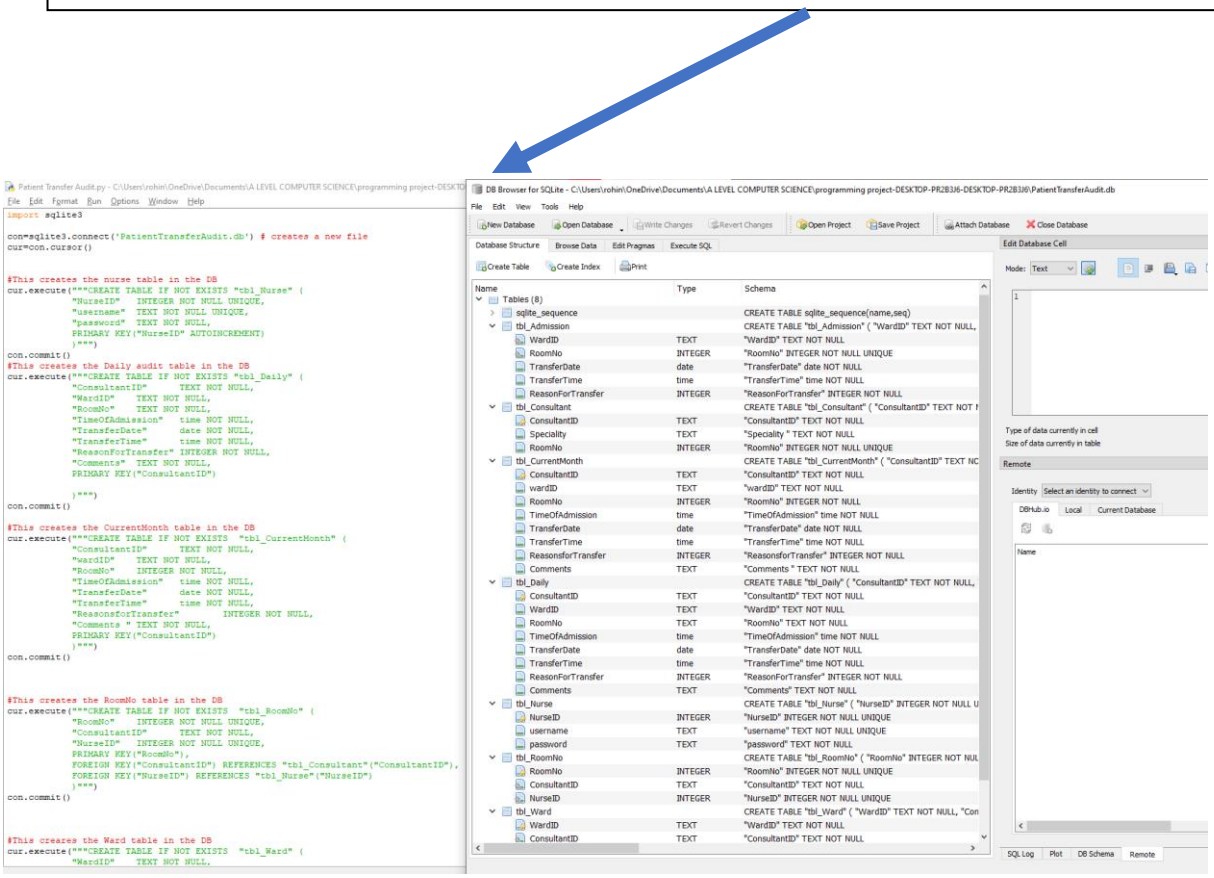
To solve this i removed the **"setupDB()"** function and instead added **"IF NOT EXISTS"** to each table code in the program. I ran the program again and this time all the tables showed up in my database, furthermore allowing me to create the tables again without any errors. Although this was not highly important, as this code only needs to be run once to create all the necessary tables, it still provides correctness to my program.

Patient Transfer Audit.py - C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming project-DESKT(

File Edit Format Run Options Window Help

```
import sqlite3

con=sqlite3.connect('PatientTransferAudit.db') # creates a new file
cur=con.cursor()


#This creates the nurse tabl
cur.execute("""CREATE TAB E IF NOT EXISTS tbl_Nurse" (
            "NurseID"   INT  NOT NULL UNIQUE,
            "username"  TEXT NOT NULL UNIQUE,
            "password"  TEXT NOT NULL,
            PRIMARY KEY("NurseID" AUTOINCREMENT)
            )""")
con.commit()
#This creates the Daily audi
cur.execute("""CREATE TAB E IF NOT EXISTS "tbl_Daily" (
            "ConsultantID"      NOT NULL,
            "WardID"    TEXT NOT NULL,
            "RoomNo"    TEXT NOT NULL,
            "TimeOfAdmission"   time NOT NULL,
            "TransferDate"      date NOT NULL,
            "TransferTime"      time NOT NULL,
            "ReasonForTransfer" INTEGER NOT NULL,
            "Comments"  TEXT NOT NULL,
            PRIMARY KEY("ConsultantID")

            )""")
con.commit()

#This creates the CurrentMo
cur.execute("""CREATE TAB E IF NOT EXISTS "tbl_CurrentMonth" (
            "ConsultantID"      NOT NULL,
            "wardID"    TEXT NOT NULL,
            "RoomNo"    INTEGER NOT NULL,
            "TimeOfAdmission"   time NOT NULL,
            "TransferDate"      date NOT NULL,
            "TransferTime"      time NOT NULL,
            "ReasonsforTransfer"        INTEGER NOT NULL,
            "Comments " TEXT NOT NULL,
            PRIMARY KEY("ConsultantID")
            )""")
con.commit()



#This creates the RoomNo
cur.execute("""CREATE TAB E IF NOT EXISTS tbl_RoomNo" (
            "RoomNo"    INT  NOT NULL UNIQUE,
            "ConsultantID"      TEXT NOT NULL,
            "NurseID"   INTEGER NOT NULL UNIQUE,
            PRIMARY KEY("RoomNo"),
            FOREIGN KEY("ConsultantID") REFERENCES "tbl_Consultant"("ConsultantID"),
            FOREIGN KEY("NurseID") REFERENCES "tbl_Nurse"("NurseID")
            )""")
con.commit()



#This creates the Ward table in the DB
```

This screenshot shows the rest of the tables have been coded into my python file and the tables have been created successfully as shown in DB browser



In this screenshot, all the other tables are present. In tbl_RoomNo there is a foreign key of ConsultantID and NurseID which gains the data present in the Consultant table and Nurse table. The consultantID is needed to identify which patients they hold; this also creates a one-to-many relationship with from the consultantID table. The nurseID is needed to show that a nurse has logged a consultant to a patient that has also been assigned a room. Tbl _Ward requires the consultant which is taking of the patient on that ward so its consutantID is a foreign key to reference the consultantID in tbl_RoomNo.

The foreign keys in tbl_Admission makes a one-to-many-many-to-one relationship between tbl_RoomNo and tbl_WardID. One room can have many admissions that could be on many wards. Therefore to maintain this information tbl_Admission stores the RoomNo and WardID from the different tables. Henceforward, if the nurses on the ward want to access any information on a patient I can adapt some changes to make the database available for nurses to view the different admissions, allowing them to keep a record for patients who may come back to the hospital again for a check-up.

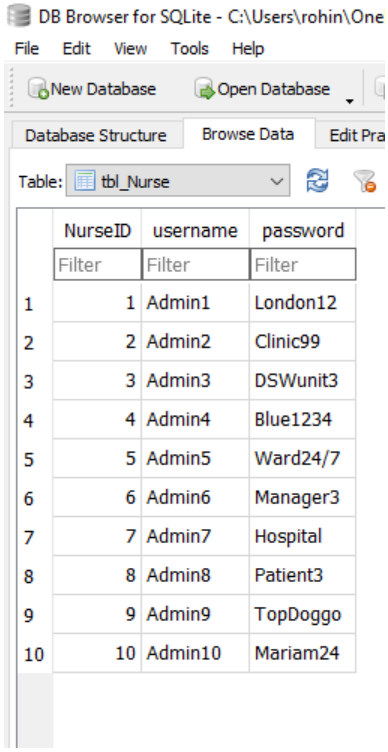
Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Field entries into my tables

To check if my tables work, I filled my tables with data from csv files. This ensured each record could be filled without any errors.

```
#populates Nurse table with logins
NurseCredentialsFile=open("NurseCredentials.csv","r")#Information added from NurseCrendentials csv file
for line in NurseCredentialsFile:
    line=line.strip()#removes \n
    userName,passWord=line.split(",")
    cur.execute("INSERT INTO tbl_Nurse(username,password)VALUES (?,?)",[userName,passWord])

con.commit()#all changes are commited
print("Database Setup Complete")
```

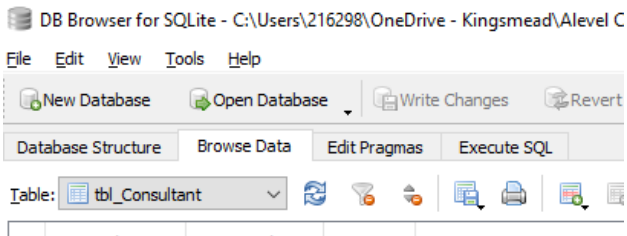
In this section of code, the CSV file is opened and the cursor is used to execute the **INSERT** command so the nID, userName, passWord can be added to the table nurses. No code has to be written to the file here as theses usernames and passwords will be fixed and cannot be altered by user inputs. Here the NurseID is an autoincrementing integer so this does not have to be inputted, therefore it will automatically increment giving a ID to each user as they enter their username and password.



	NurseID	username	password
	Filter	Filter	Filter
1	1	Admin1	London12
2	2	Admin2	Clinic99
3	3	Admin3	DSWunit3
4	4	Admin4	Blue1234
5	5	Admin5	Ward24/7
6	6	Admin6	Manager3
7	7	Admin7	Hospital
8	8	Admin8	Patient3
9	9	Admin9	TopDoggo
10	10	Admin10	Mariam24

```
#populates Consultant table with their initials and their speciality
ConsultantDetailsFile=open("ConsultantDetails.csv","r")#Information added from ConsultantDetails csv file
for line in ConsultantDetailsFile:
    line=line.strip()#removes\n
    cID,Speciality,roomNo=line.split(",")
    cur.execute("INSERT INTO tbl_Consultant VALUES (?,?,?)",[cID,Speciality,roomNo])

    con.commit()#all changes are commited
    print("Database Setup Complete")
```

The other table that needs to have information pre-inputted, is the consultant table. My client has given me a list of some of the consultants that patients are under care of and their specialities to input into the table. However, the RoomNo field in tbl_consultant will be filled later as the nurses use the program, therefore this column is empty.



	ConsultantID	Speciality	RoomNo
	Filter	Filter	Filter
1	RH	ophthalmology	-
2	RL	ophthalmology	-
3	RSW	ophthalmology	-
4	SJ	ophthalmology	-
5	GR	ophthalmology	-
6	IGD	ophthalmology	-
7	JJ	ophthalmology	-
8	MB	ophthalmology	-
9	MW	ophthalmology	-
10	FA	ophthalmology	-

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

```
IDLE Shell 3.9.6
File Edit Shell Debug Options Window Help
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming project-DESKTOP-PR2B3J6-DESKTOP-PR2B3J6\Patient Transfer Audit.py
Traceback (most recent call last):
  File "C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming project-DESKTOP-PR2B3J6-DESKTOP-PR2B3J6\Patient Transfer Audit.py",
    cur.execute("INSERT INTO tbl_Nurse VALUES (?,?,?)",[nID,userName,passWord])
sqlite3.IntegrityError: UNIQUE constraint failed: tbl_Nurse.NurseID
>>>
```

```
IDLE Shell 3.9.6
File Edit Shell Debug Options Window Help
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\216298\OneDrive - Kingsmead\Alevel CS coursework 2022-2023\P
atient Transfer Audit.py
Traceback (most recent call last):
  File "C:\Users\216298\OneDrive - Kingsmead\Alevel CS coursework 2022-2023\Pati
ent Transfer Audit.py", line 117, in <module>
    cur.execute("INSERT INTO tbl_Consultant VALUES (?,?,?)",[cID,Speciality,room
No])
sqlite3.IntegrityError: UNIQUE constraint failed: tbl_Consultant.RoomNo
>>>
```

These integrity errors came up when I tried running the database. This is when the nurse and consultant table cannot be run again because the table has already created. To solve this problem, I placed all my code in a function so the database can be created multiple times without any errors. Earlier when I put my code in a function the database went blank but now, I realised that I forgot to commit the changes to the database. Therefore, now when I integrated the function **setupDB()** operates with my code correctly. The solution is shown below.

```
Patient Transfer Audit.py - C:\Users\216298\OneDrive - Kingsmead\Alevel CS coursework 2022-2023\
File Edit Format Run Options Window Help
import sqlite3

con=sqlite3.connect('PatientTransferAudit.db') # creates a new file
cur=con.cursor()

def setupDB():

    #This creates the nurse table in the DB
    cur.execute("""CREATE TABLE IF NOT EXISTS "tbl_Nurse" (
                "NurseID"       INTEGER NOT NULL UNIQUE,
                "username"      TEXT NOT NULL UNIQUE,
                "password"      TEXT NOT NULL,
                PRIMARY KEY("NurseID" AUTOINCREMENT)
    )""")

    con.commit()
    #This creates the Daily audit table in the DB
    cur.execute("""CREATE TABLE IF NOT EXISTS "tbl_Daily" (
                    "ConsultantID"      TEXT NOT NULL,
                    "WardID"    TEXT NOT NULL,
                    "RoomNo"    TEXT NOT NULL,
                    "TimeOfAdmission"   time NOT NULL,
                    "TransferDate"      date NOT NULL,
                    "TransferTime"      time NOT NULL,

    #populates Nurse table with logins
    NurseCredentialsFile=open("NurseCredentials.csv","r")#Information addec
    for line in NurseCredentialsFile:
        line=line.strip()#removes \n
        userName,passWord=line.split(",")
        cur.execute("INSERT INTO  tbl_Nurse (username,password) VALUES (?,?

        con.commit()#all changes are commited
        print("Database Setup Complete")


    #populates Consultant table with their initials and their speciality
    ConsultantDetailsFile=open("ConsultantDetails.csv","r")#Information adc
    for line in ConsultantDetailsFile:
        line=line.strip()#removes\n
        cID,Speciality,roomNo=line.split(",")
        cur.execute("INSERT INTO tbl_Consultant VALUES (?,?,?)",[cID,Specia

        con.commit()#all changes are commited
        print("Database Setup Complete")


setupDB()
```

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

First prototype

Creating my interphase

Here I imported AppJar so I could use the GUI and store it in the global variable app so that could be called throughout the whole program.

Patient Transfer Audit.py - C:\Use\216298\OneDrive - Kingsmead\Alevel CS coursework 2022-2023\Patient Transfe

File Edit Format Run Options Window Help

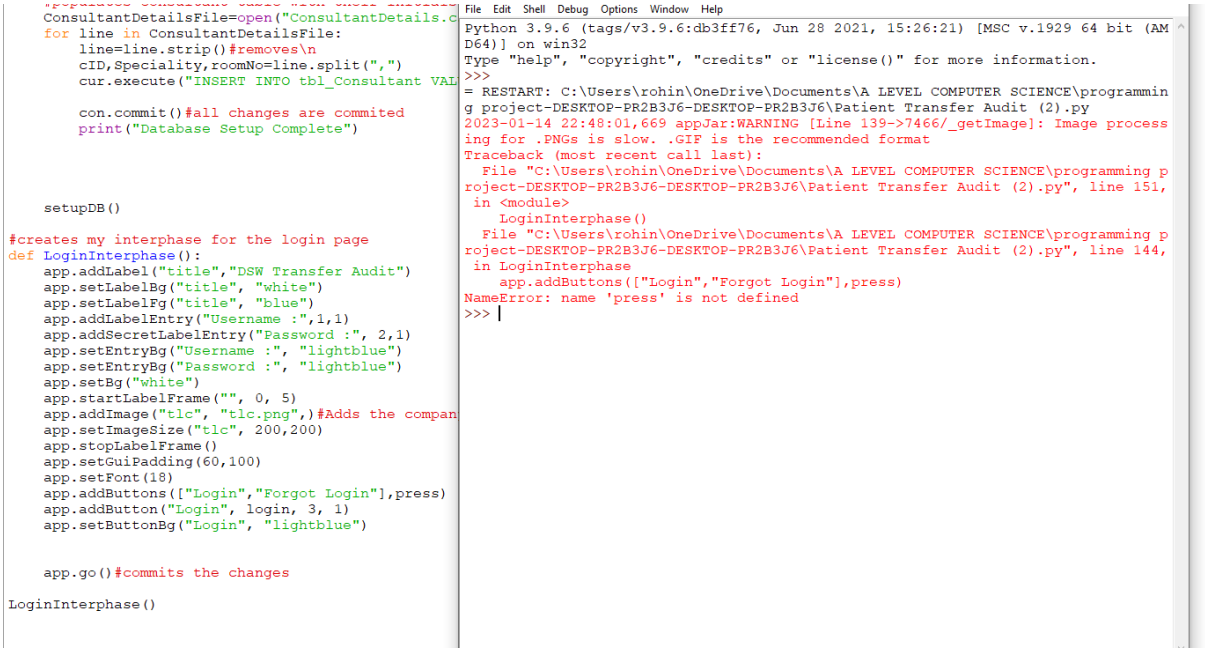
```
import sqlite3
from appJar import gui
con=sqlite3.connect('PatientTransferAudit.db') # creates a new file
cur=con.cursor()

app=gui("PatientTransferAudit")

def setupDB():

    #This creates the nurse table in the DB
    cur.execute("""CREATE TABLE IF NOT EXISTS "tbl_Nurse" (
                "NurseID"       INTEGER NOT NULL UNIQUE,
                "username"      TEXT NOT NULL UNIQUE,
                "password"      TEXT NOT NULL,
                PRIMARY KEY("NurseID" AUTOINCREMENT)
```

To begin with, I split my coded into two sections where in the press function all my buttons for the whole program will present each of them having a specific role. The createInterphase function will hold all the onscreen developments such as the layout design of each page.

```
#creates functions for each of my buttons on every page of my program
def press(button):
    if button=="Login":
        app.hide()
        app.showSubWindow("MainMenu")#takes user to main menu


    elif button=="Forgot Login":
        app.infoBox("Help","Contact 216298@kingsmead.org for more informa



#creates my interphase for the program
def createInterphase():
####Login page########
    app.addLabel("title","DSW Transfer Audit")
    app.setLabelBg("title", "white")
    app.setLabelFg("title", "blue")
    app.addLabelEntry("Username :",1,1)
    app.addSecretLabelEntry("Password :", 2,1)
    app.setEntryBg("Username :", "lightblue")
    app.setEntryBg("Password :", "lightblue")
    app.setFocus("Username :")
    app.setBg("white")
    app.startLabelFrame("", 0, 5)
    app.addImage("tlc", "tlc.png",)#Adds the company logo
    app.setImageSize("tlc", 200,200)
    app.stopLabelFrame()
    app.setGuiPadding(60,100)
    app.setFont(18)
    app.addButtons(["Login","Forgot Login"],press, 3, 1)
    app.setButtonBg("Login", "lightblue")
    app.setButtonBg("Forgot Login", "lightblue")
####Main Menu#####
    app.startSubWindow("MainMenu")
    app.addLabel("title2","Main Menu")


    app.stopSubWindow()
    app.go()#commits the changes

createInterphase()
```

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

I was trying to make my “login” and “forgot login” buttons but then this error popped up where press was not defined. I then quickly realised by researching into appJar that I need to make the function **press** to actually add functionality to my buttons and then from this my buttons could be added to the interphase.



```
    ConsultantDetailsFile=open("ConsultantDetails.c
    for line in ConsultantDetailsFile:
        line=line.strip()#removes\n
        cID,Speciality,roomNo=line.split(",")
        cur.execute("INSERT INTO tbl_Consultant VAL

        con.commit()#all changes are commited
        print("Database Setup Complete")



    setupDB()

#creates my interphase for the login page
def LoginInterphase():
    app.addLabel("title","DSW Transfer Audit")
    app.setLabelBg("title", "white")
    app.setLabelFg("title", "blue")
    app.addLabelEntry("Username :",1,1)
    app.addSecretLabelEntry("Password :", 2,1)
    app.setEntryBg("Username :", "lightblue")
    app.setEntryBg("Password :", "lightblue")
    app.setBg("white")
    app.startLabelFrame("", 0, 5)
    app.addImage("tlc", "tlc.png",)#Adds the compan
    app.setImageSize("tlc", 200,200)
    app.stopLabelFrame()
    app.setGuiPadding(60,100)
    app.setFont(18)
    app.addButtons(["Login","Forgot Login"],press)
    app.addButton("Login", login, 3, 1)
    app.setButtonBg("Login", "lightblue")


    app.go()#commits the changes

LoginInterphase()
```

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programmin
g project-DESKTOP-PR2B3J6-DESKTOP-PR2B3J6\Patient Transfer Audit (2).py
2023-01-14 22:48:01,669 appJar:WARNING [Line 139->7466/_getImage]: Image process
ing for .PNGs is slow. .GIF is the recommended format
Traceback (most recent call last):
  File "C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming p
roject-DESKTOP-PR2B3J6-DESKTOP-PR2B3J6\Patient Transfer Audit (2).py", line 151,
 in <module>
    LoginInterphase()
  File "C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming p
roject-DESKTOP-PR2B3J6-DESKTOP-PR2B3J6\Patient Transfer Audit (2).py", line 144,
 in LoginInterphase
    app.addButtons(["Login","Forgot Login"],press)
NameError: name 'press' is not defined
>>>
```

This is the solution to the error I encountered.

```
    setupDB()

#creates my interphase for the login page
def LoginInterphase():
    app.addLabel("title","DSW Transfer Audit")
    app.setLabelBg("title", "white")
    app.setLabelFg("title", "blue")
    app.addLabelEntry("Username :",1,1)
    app.addSecretLabelEntry("Password :", 2,1)
    app.setEntryBg("Username :", "lightblue")
    app.setEntryBg("Password :", "lightblue")
    app.setFocus("Username :")
    app.setBg("white")
    app.startLabelFrame("", 0, 5)
    app.addImage("tlc", "tlc.png",)#Adds the company logo
    app.setImageSize("tlc", 200,200)
    app.stopLabelFrame()
    app.setGuiPadding(60,100)
    app.setFont(18)
    def press(button):
        if button=="Login":
            app.hide("PatientTransferAudit")

        else:
            button=="ForgotLogin"
            app.hide("PatientTransferAudit")

    app.addButtons(["Login","Forgot Login"],press, 3, 1)
    app.setButtonBg("Login", "lightblue")
    app.setButtonBg("Forgot Login", "lightblue")


    app.go()#commits the changes

LoginInterphase()
```

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Here, an error occurred when I tried to link the Main Menu with the login page via a SubWindow. It suggested that there were two widgets which had the same name. After some research, I found out that when I start a new SubWindow I have to “show” it in the previous window button function (LoginInterphase). I renamed both of my SubWindows to resolve this problem.

```
    setupDB()

#creates my interphase for the login page
def LoginInterphase():
    app.addLabel("title","DSW Transfer Audit")
    app.setLabelBg("title", "white")
    app.setLabelFg("title", "blue")
    app.addLabelEntry("Username :",1,1)
    app.addSecretLabelEntry("Password :", 2,1)
    app.setEntryBg("Username :", "lightblue")
    app.setEntryBg("Password :", "lightblue")
    app.setFocus("Username :")
    app.setBg("white")
    app.startLabelFrame("", 0, 5)
    app.addImage("tlc", "tlc.png",)#Adds the company logo
    app.setImageSize("tlc", 200,200)
    app.stopLabelFrame()
    app.setGuiPadding(60,100)
    app.setFont(18)
    def press(button):
        if button=="Login":
            app.hide("PatientTransferAudit")
            app.startSubWindow("MainMenu")#takes user to main men


        else:
            button=="ForgotLogin"
            app.hide("PatientTransferAudit")
            app.infoBox("Help","Contact 216298@kingsmead.org for

    app.addButtons(["Login","Forgot Login"],press, 3, 1)
    app.setButtonBg("Login", "lightblue")
    app.setButtonBg("Forgot Login", "lightblue")

LoginInterphase()

def MainMenuInterphase():
    app.startSubWindow("MainMenu")
    app.addLabel("title2","Main Menu")


    app.go()#commits the changes


MainMenuInterphase()
```

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programmin
g project-DESKTOP-PR2B3J6-DESKTOP-PR2B3J6\Patient Transfer Audit (2).py
2023-01-14 23:25:54,722 appJar:WARNING [Line 140->7466/_getImage]: Image process
ing for .PNGs is slow. .GIF is the recommended format
2023-01-14 23:25:54,802 appJar:WARNING [Line 167->1821/go]: No stopContainer cal
led on: SubWindow
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\rohin\AppData\Local\Programs\Python\Python39\lib\tkinter\__init
__.py", line 1892, in __call__
    return self.func(*args)
  File "C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming p
roject-DESKTOP-PR2B3J6-DESKTOP-PR2B3J6\appJar\appjar.py", line 3783, in <lambda>
    return lambda *args: funcName(param)
  File "C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming p
roject-DESKTOP-PR2B3J6-DESKTOP-PR2B3J6\Patient Transfer Audit (2).py", line 148,
 in press
    app.startSubWindow("MainMenu")#takes user to main menu
  File "C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming p
roject-DESKTOP-PR2B3J6-DESKTOP-PR2B3J6\appJar\appjar.py", line 5680, in startSub
Window
    self.widgetManager.verify(WIDGET_NAMES.SubWindow, name)
  File "C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming p
roject-DESKTOP-PR2B3J6-DESKTOP-PR2B3J6\appJar\appjar.py", line 15862, in verify
    raise ItemLookupError("Duplicate widgetName: " + widgetName)
appJar.appjar.ItemLookupError: Duplicate widgetName: MainMenu
>>>
```

```
#creates functions for each of my buttons on every page of my program
def press(button):
    if button=="Login":
        app.hide()
        app.showSubWindow("MainMenu")#takes user to main menu

    elif button=="Forgot Login":
        app.infoBox("Help","Contact 216298@kingsmead.org for more information or any quieries regarding your credentials")



#creates my interphase for the program
def createInterphase():
####Login page#########
    app.addLabel("title","DSW Transfer Audit")
    app.setLabelBg("title", "white")
    app.setLabelFg("title", "blue")
    app.addLabelEntry("Username :",1,1)
    app.addSecretLabelEntry("Password :", 2,1)
    app.setEntryBg("Username :", "lightblue")
    app.setEntryBg("Password :", "lightblue")
    app.setFocus("Username :")
    app.setBg("white")
    app.startLabelFrame("", 0, 5)
    app.addImage("tlc", "tlc.png",)#Adds the company logo
    app.setImageSize("tlc", 200,200)
    app.stopLabelFrame()
    app.setGuiPadding(60,100)
    app.setFont(18)
    app.addButtons(["Login","Forgot Login"],press, 3, 1)
    app.setButtonBg("Login", "lightblue")
    app.setButtonBg("Forgot Login", "lightblue")
####Main Menu#####
    app.startSubWindow("MainMenu")
    app.addLabel("title2","Main Menu")


    app.stopSubWindow()
    app.go()#commits the changes

createInterphase()
```

I fixed this problem by renaming the SubWindows to show and hide when clicked and separated my code into buttons and the main design through functions as previously mentioned. The pages in my **createInterphase** have also been spaced out my comments so they are easy to identify if a change is needed.

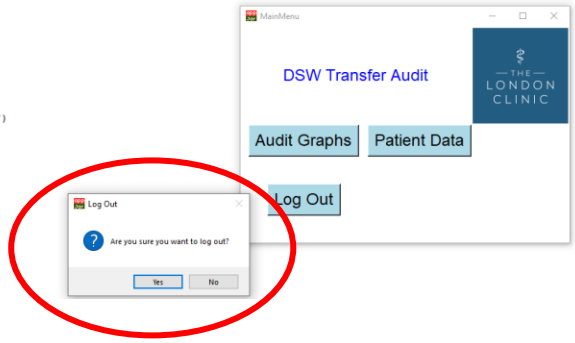
```
    elif button=="Log Out":
        app.questionBox("Log Out","Are you sure you want to log out?")
        app.hide("MainMenu")




#creates my interphase for the program
def createInterphase():
####Login page#########
    app.addLabel("title","DSW Transfer Audit" ,0,1)
    app.setLabelBg("title", "white")
    app.setLabelFg("title", "blue")
    app.addLabelEntry("Username :",1,1)
    app.addSecretLabelEntry("Password :", 2,1)
    app.setEntryBg("Username :", "lightblue")
    app.setEntryBg("Password :", "lightblue")
    app.setFocus("Username :")
    app.setBg("white")
    app.startLabelFrame("", 0, 2)
    app.addImage("tlc", "tlc.png", 0,2)#Adds the company logo
    app.setImageSize("tlc", 200,200)
    app.setGuiPadding(10,10)
    app.stopLabelFrame()
    app.setFont(18)
    app.addButtons(["Login","Forgot Login"],press, 3, 1)
    app.setButtonBg("Login", "lightblue")
    app.setButtonBg("Forgot Login", "lightblue")
####Main Menu#####
    app.startSubWindow("MainMenu")
    app.addLabel("title2","DSW Transfer Audit")
    app.addImage("tlc2", "tlc.png", 0,2)#Adds the company logo
    app.addNamedButton("Patient Data","ButtonPD",press, 2, 1)
    app.addNamedButton("Audit Graphs","ButtonAG",press, 2, 0)
    app.setNamedButtonBg("Patient Data", "lightblue")
    app.setNamedButtonBg("Audit Graphs", "lightblue")
    app.addButtons(["Log Out"],press)
    app.setButtonBg("Log Out", "lightblue")
```

When I went to logout from the main menu screen, the login page would load quicker than the actual question box where the program asks the user if they want to logout of the system. To solve this, I needed to create a separate subroutine for the logout process.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Now the login page does not show up straight away when the user clicks the “Log out” button but waits for the user to choose an option, to then proceed with a desired output.

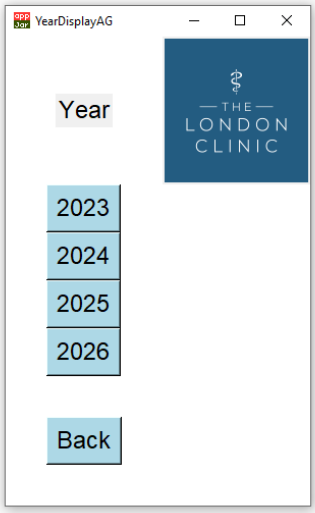
```
        app.stopSubWindow()




#creates a function so the user can logout
def logout(leave):

    leave = app.questionBox("Log Out","Are you sure you want to log out?")
    if leave==True:
        app.hideSubWindow("MainMenu")
        app.showSubWindow("Login")


    else:
        app.showSubWindow("MainMenu")


def NewRow(btn):
    print(btn)
    app.addGridRow("g1"," ")
```



Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

This is the code for the year display pages and I had some trouble laying out these pages but I quickly discovered that there was an appJar tutorial on the website which guided me on this development.

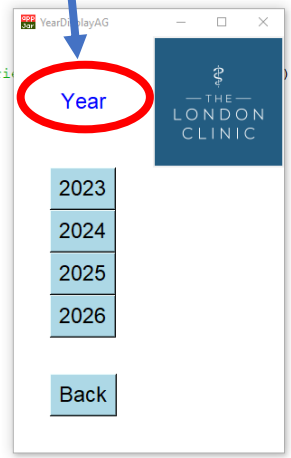
```
####Year Display AG ####
    app.startSubWindow("YearDisplayAG")
    app.addLabel("title3","Year")
    app.addImage("tlc3", "tlc.png", 0,2)#Adds the company logo
    app.setFont(18)
    app.setBg("white")
    app.setStretch("columns")
    app.addNamedButton("2023","2023_AG",press,1,0)
    app.addNamedButton("2024","2024_AG",press,2,0)
    app.addNamedButton("2025","2025_AG",press,3,0)
    app.addNamedButton("2026","2026_AG",press,4,0)
    app.setButtonBg("2023_AG", "lightblue")
    app.setButtonBg("2024_AG", "lightblue")
    app.setButtonBg("2025_AG", "lightblue")
    app.setButtonBg("2026_AG", "lightblue")
    app.setSticky("sw")
    app.setPadding([40,40])
    app.addNamedButton("Back","backAG",press)
    app.setButtonBg("backAG", "lightblue")



    app.stopSubWindow()


####Month Display AG####
    app.startSubWindow("MonthDisplayAG")
    app.addLabel("title4","Months")



####Year Display PD####
    app.startSubWindow("YearDisplayPD")
    app.addLabel("title5","Year")
    app.addImage("tlc5", "tlc.png", 0,2)#Adds the company logo
    app.setFont(18)
    app.setBg("white")
    app.setStretch("columns")
    app.addNamedButton("2023","2023_PD",press,1,0)
    app.addNamedButton("2024","2024_PD",press,2,0)
    app.addNamedButton("2025","2025_PD",press,3,0)
    app.addNamedButton("2026","2026_PD",press,4,0)
    app.setButtonBg("2023_PD", "lightblue")
    app.setButtonBg("2024_PD", "lightblue")
    app.setButtonBg("2025_PD", "lightblue")
    app.setButtonBg("2026_PD", "lightblue")
    app.setSticky("sw")
    app.setPadding([40,40])
```

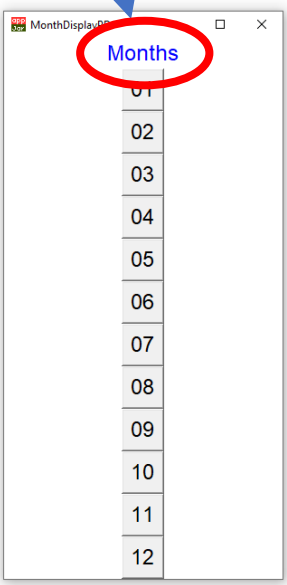


Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

When the a year was clicked from the different pages (YearPD and YearAG) only one of the assigned month pages would pop up and that would be the Patient Data month page, as it was above the Audit Graph month code in this press() function. Therefore, I needed to fix this to prevent users being taken to the wrong pages which may confuse them, delaying their time to tend to patients on the ward.

```
#creates functions for each of my buttons on every page of my program
def press(button):
    if button=="Login":
        app.hideSubWindow("Login")
        app.showSubWindow("MainMenu")#takes user to main menu


    elif button=="Forgot Login":
        app.infoBox("Help","Contact 216298@kingsmead.org for more information or any quieri

    elif button=="ButtonPD":
        app.hideSubWindow("MainMenu")
        app.showSubWindow("YearDisplayPD")

    elif button=="ButtonAG":
        app.hideSubWindow("MainMenu")
        app.showSubWindow("YearDisplayAG")


    elif button=="backYPD":
        app.hideSubWindow("YearDisplayPD")
        app.showSubWindow("MainMenu")


    elif button=="backYAG":
        app.hideSubWindow("YearDisplayAG")
        app.showSubWindow("MainMenu")

    elif button=="backMAG":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("YearDisplayAG")

Allows each year selected to direct to the same month page of Patient Data
    elif button=="2023_PD" or "2024_PD" or "2025_PD" or "2026_PD":
        app.hideSubWindow("YearDisplayPD")
        app.showSubWindow("MonthDisplayPD")

Allows each year selected to direct to the same month page of audit graphs
    elif button=="2023_AG" or "2024_AG" or "2025_AG" or "2026_AG":
        app.hideSubWindow("YearDisplayAG")
        app.showSubWindow("MonthDisplayAG")


    elif button=="backMAG":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("YearDisplayAG")
```





```
        app.showSubWindow("YearDisplayAG")

#Allows each year selected to direct to the same month page of Patient Data
    elif button=="PD_2023" or button=="PD_2024" or button=="PD_2025" or button=="PD_2026":
        app.hideSubWindow("YearDisplayPD")
        app.showSubWindow("MonthDisplayPD")

#Allows each year selected to direct to the same month page of audit graphs
    elif button=="AG_2023" or button=="AG_2024" or button=="AG_2025" or button=="AG_2026":
        app.hideSubWindow("YearDisplayAG")
        app.showSubWindow("MonthDisplayAG")

    elif button=="backMPD":
        app.hideSubWindow("MonthDisplayPD")
        app.showSubWindow("YearDisplayPD")


    elif button=="backMAG":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("YearDisplayAG")
```

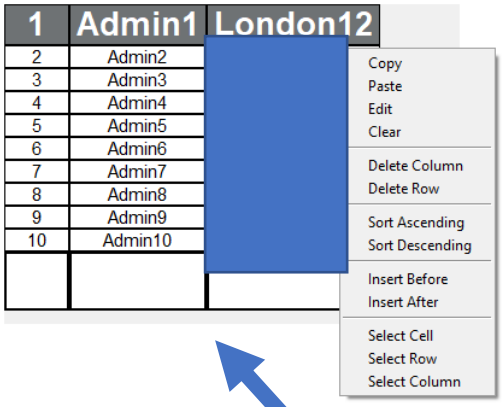
I found out that all the each button has to be declared by the parameter for the program to link the pages.

```
####Patient Transfer Databases####
    app.startSubWindow("PatientTransferDatabases")
    app.setSize("fullscreen")
    app.addLabel("title7","Patient Transfer Databases",0,0,2,2)
    app.setLabelBg("title7", "white")
    app.setLabelFg("title7", "blue")
    app.setFont(18)
    app.setBg("white")
    app.addButton("Daily",press,1,0)
    app.addButton("Current Month Database",press,1,1)
    app.setButtonBg("Daily","lightblue")
    app.setButtonBg("Current Month Database","lightblue")
    app.setSticky("sw")
    app.setPadding([40,40])
    app.addNamedButton("Back","BackPTD",press)
    app.setButtonBg("BackPTD","lightblue")
    app.stopSubWindow()
```

After the user selects a year and month, they would like to view for patient data, this screen pops up. This allows for the user to select which table they want to edit/view(daily or for the whole month). In creating this page the layout was not how I wanted so I corrected positions of the widgets using grid references, giving the page a more neat look.

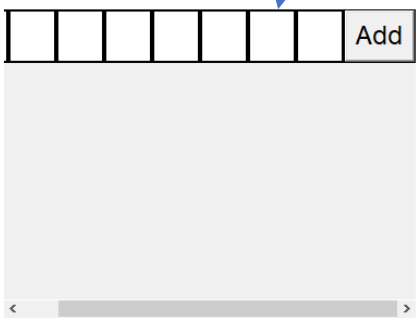
Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Daily Table



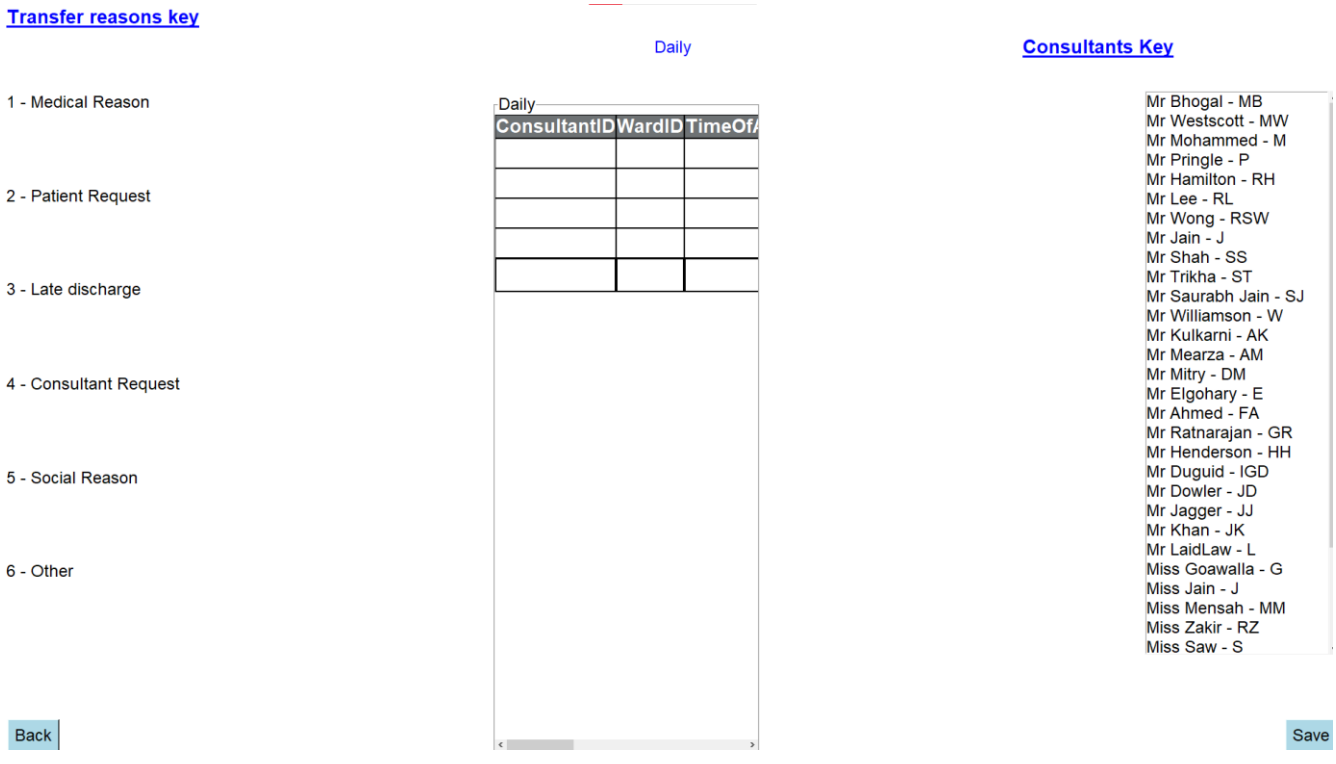

```
def printthe(btn):
    print(app.getGridRow("g1", btn))
    app.setFont(20)
    app.addGrid("g1",[["PatientTransferAudit.db","tbl_Nurse"]],  action=printhe, showMenu=True,actionButton ="Update",addRow=NewRow)


def NewRow(btn):
    print(btn)
    app.addGridRow("g1","")
```

Initially when creating my Daily table, I tried importing my actual database to display it in the program and I used the **tbl_Nurse** as an example because it was the only one that had data in. This worked but when I tried running the same thing with my **tbl_Daily** nothing was displayed as shown below. I then quickly realised there was nothing in the database to be displayed. To solve this dilemma, I decided to create a grid on the daily table page that would allow for inputs for users which will then be stored to be viewed by any other user.



Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

```
####Daily Database####
    app.startSubWindow("DailyDatabase",0,1)
    app.setSize("fullscreen")
    app.addLabel("title8","Daily",0,1)#Title of the table
    app.setLabelBg("title8", "white")
    app.setLabelFg("title8", "blue")
    app.setFont(18)
    app.setBg("white")
    app.addLabel("title9","Consultants Key",0,10,8).config(font=("","20", "bold","underline"))
    app.setSticky("e")
    app.setFont(10)
    #Adds all the consultants under the consultants key
    app.addListBox("Consultants", ["Mr Bhogal - MB",
                                   "Mr Westscott - MW",
                                   "Mr Mohammed - M",
                                   "Mr Pringle - P",
                                   "Mr Hamilton - RH",
                                   "Mr Lee - RL",
                                   "Mr Wong - RSW",
                                   "Mr Jain - J",
                                   "Mr Shah - SS",
                                   "Mr Trikha - ST",
                                   "Mr Saurabh Jain - SJ",
                                   "Mr Williamson - W",
                                   "Mr Kulkarni - AK",
                                   "Mr Mearza - AM",
                                   "Mr Mitry - DM",
                                   "Mr Elgohary - E",
                                   "Mr Ahmed - FA",
                                   "Mr Ratnarajan - GR",
                                   "Mr Henderson - HH",
                                   "Mr Duguid - IGD",
                                   "Mr Dowler - JD",
                                   "Mr Jagger - JJ",
                                   "Mr Khan - JK",
                                   "Mr LaidLaw - L",
                                   "Miss Goawalla - G",
                                   "Miss Jain - J",
                                   "Miss Mensah - MM",
                                   "Miss Zakir - RZ",
                                   "Miss Saw - S",
                                   "Mr El-Amir - AEA",
                                   "Mr Mearza - AM",
                                   "Mr Mitry - DM",
                                   "Mr Elgohary - E",
                                   "Mr F Ahmed - FA"],1,10,10,8)
    app.setLabelBg("title9", "white")
    app.setLabelFg("title9", "blue")
    app.setSticky("nw")
    app.addLabel("title10","Transfer reasons key",0,0).config(font=("","20", "bold","underline"))
    app.setLabelBg("title10", "white")
    app.setLabelFg("title10", "blue")
    app.addLabel("medical reason","1 - Medical Reason",1,0)
```



Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

```
                                    "Mr F Ahmed - FA"],1,10,10,8)
    app.setLabelBg("title9", "white")
    app.setLabelFg("title9", "blue")
    app.setSticky("nw")
    app.addLabel("title10","Transfer reasons key",0,0).config(font=("","20", "bold","underline"))
    app.setLabelBg("title10", "white")
    app.setLabelFg("title10", "blue")
    app.addLabel("medical reason","1 - Medical Reason",1,0)
    app.addLabel("patient request","2 - Patient Request",2,0)
    app.addLabel("late discharge","3 - Late discharge",3,0)
    app.addLabel("consultant request","4 - Consultant Request",4,0)
    app.addLabel("social reason","5 - Social Reason",5,0)
    app.addLabel("other","6 - Other",6,0)
    #addTableDaily()
    app.setFont(20)
    #creates a grid where data can be inputted
    app.startLabelFrame("Daily",row=1,column=1,rowspan=14,colspan=14,sticky="nesw")
    app.setGuiPadding(50,50)
    app.addGrid("g1",

    [["ConsultantID", "WardID", "TimeOfAdmission","TransferDate","TransferTime","ReasonForTransfer","Comments"],

    ["","" ,""],

    ["","" , ""],

    ["","" , ""],

    ["","", ""]],
#Allows for the features of the grid to exist and when user right-clicks on an entry box a sub-menu pops up which allows for changes to the grid
    action=printthe, showMenu=True,actionButton ="save",addRow=addRow2)
    app.stopLabelFrame()
    app.setSticky("se")
    app.addButton("Save",press,10,10)
    app.setButtonBg("Save","lightblue")
    app.setSticky("sw")
    app.addNamedButton("Back","BackDD",press,10,0)
    app.setButtonBg("BackDD","lightblue")
```

This is the final result of the Daily table which will store information about patient transfers and refresh daily. The transfer reasons and consultants key are separated to allow for easy viewing of the data. The Daily table is editable and can store information but the functionality of it being stored even after the program is closed will not be implemented until prototype 2. The only problem I faced was the positioning of this page because there is a lot on it but with grid references and appjar's "sticky" function I was able to construct the page how I envisioned in my final designs.

Current Month Table

This is the current Month table which holds information about patient transfers from the entire current month. This works the same as the Daily table and there were only minor errors when creating this page, most of them being Label name duplicates. In addition, here again the implementation for the functionality of the table will be implemented in prototype 2, allowing data to be deleted or stored permanently in the table when inputted by the user.

```
####Current Month Database####
    app.startSubWindow("CurrentMonthDatabase",0,1)
    app.setSize("fullscreen")
    app.addLabel("title11","Current Month",0,1)#Title of the table
    app.setLabelBg("title11", "white")
    app.setLabelFg("title11", "blue")
    app.setFont(18)
    app.setBg("white")
    app.addLabel("title12","Consultants Key",0,10,8).config(font=("","20", "bold","underline"))
    app.setLabelBg("title12", "white")
    app.setLabelFg("title12", "blue")
    app.setSticky("e")
    app.setFont(10)
    #Adds all the consultants under the consultants key
    app.addListBox("ConsultantsMonth", ["Mr Bhogal - MB",
                                        "Mr Westscott - MW",
                                        "Mr Mohammed - M",
                                        "Mr Pringle - P",
                                        "Mr Hamilton - RH",
                                        "Mr Lee - RL",
                                        "Mr Wong - RSW",
                                        "Mr Jain - J",
                                        "Mr Shah - SS",
                                        "Mr Trikha - ST",
                                        "Mr Saurabh Jain - SJ",
                                        "Mr Williamson - W",
                                        "Mr Kulkarni - AK",
                                        "Mr Mearza - AM",
                                        "Mr Mitry - DM",
                                        "Mr Elgohary - E",
                                        "Mr Ahmed - FA",
                                        "Mr Ratnarajan - GR",
                                        "Mr Henderson - HH",
                                        "Mr Duguid - IGD",
                                        "Mr Dowler - JD",
                                        "Mr Jagger - JJ",
                                        "Mr Khan - JK",
                                        "Mr LaidLaw - L",
                                        "Miss Goawalla - G",
                                        "Miss Jain - J",
                                        "Miss Mensah - MM",
                                        "Miss Zakir - RZ",
                                        "Miss Saw - S",
                                        "Mr El-Amir - AEA",
                                        "Mr Mearza - AM",
                                        "Mr Mitry - DM",
                                        "Mr Elgohary - E",
                                        "Mr F Ahmed - FA"],1,10,10,8)

    app.setSticky("nw")
    app.addLabel("title13","Transfer reasons key",0,0).config(font=("","20", "bold","underline"))
    app.setLabelBg("title13", "white")
```

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

```
                                    MR P Ahmed - FA "],1,10,10,0)

    app.setSticky("nw")
    app.addLabel("title13","Transfer reasons key",0,0).config(font=("","20", "bold","underline"))
    app.setLabelBg("title13", "white")
    app.setLabelFg("title13", "blue")
    app.addLabel("medical reason (month)","1 - Medical Reason",1,0)
    app.addLabel("patient request (month)","2 - Patient Request",2,0)
    app.addLabel("late discharge (month)","3 - Late discharge",3,0)
    app.addLabel("consultant request (month)","4 - Consultant Request",4,0)
    app.addLabel("social reason (month)","5 - Social Reason",5,0)
    app.addLabel("other (month)","6 - Other",6,0)
    #addTableDaily()
    app.setFont(20)
    #creates a grid where data can be inputted
    app.startLabelFrame("CurrentMonth",row=1,column=1,rowspan=14,colspan=14,sticky="nesw")
    app.setGuiPadding(50,50)
    app.addGrid("g2",

    [["ConsultantID", "WardID", "TimeOfAdmission","TransferDate","TransferTime","ReasonForTransfer","Comments"],

    ["","" ,""],

    ["","" , ""],

    ["","" , ""],

    ["","", ""]],
#Allows for the features of the grid to exist and when user right-clicks on an entry box a sub-menu pops up which allows for changes to the grid
    action=printthe, showMenu=True,actionButton ="save",addRow=addRow3)
    app.stopLabelFrame()
    app.setSticky("se")
    app.addNamedButton("Save","SaveButton",press,10,10)
    app.setButtonBg("SaveButton","lightblue")
    app.setSticky("sw")
    app.addNamedButton("Back","BackCD",press,10,0)
    app.setButtonBg("BackCD","lightblue")


    app.stopSubWindow()
```

Transfer reasons key

1 - Medical Reason

2 - Patient Request

3 - Late discharge

4 - Consultant Request

5 - Social Reason

6 - Other

Current Month

CurrentMonth

ConsultantID	WardID	TimeOf/

Consultants Key

Mr Bhogal - MB
Mr Westscott - MW
Mr Mohammed - M
Mr Pringle - P
Mr Hamilton - RH
Mr Lee - RL
Mr Wong - RSW
Mr Jain - J
Mr Shah - SS
Mr Trikha - ST
Mr Saurabh Jain - SJ
Mr Williamson - W
Mr Kulkarni - AK
Mr Mearza - AM
Mr Mitry - DM
Mr Elgohary - E
Mr Ahmed - FA
Mr Ratnarajan - GR
Mr Henderson - HH
Mr Duguid - IGD
Mr Dowler - JD
Mr Jagger - JJ
Mr Khan - JK
Mr LaidLaw - L
Miss Goawalla - G
Miss Jain - J
Miss Mensah - MM
Miss Zakir - RZ
Miss Saw - S

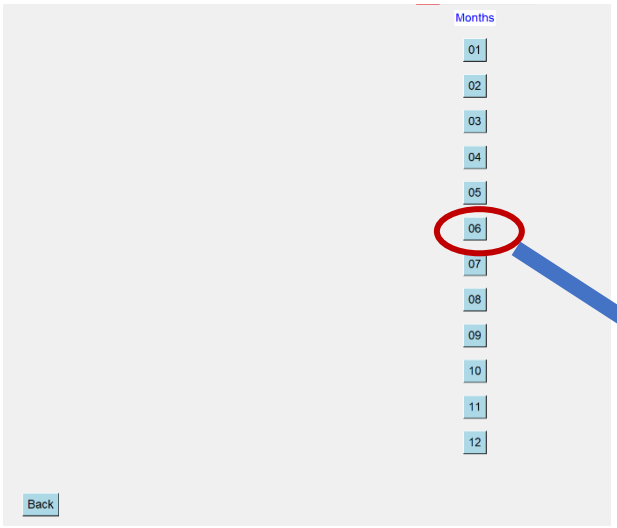
Back

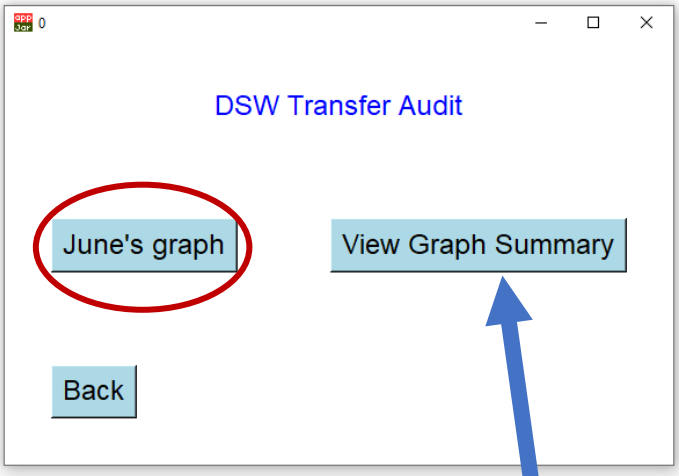
Save

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

This is the Audit graphs screen which pops up after the user selects a year and month. The correct month screen is shown here("June") when "06" is clicked. These corresponding buttons are assigned to the correct month through the **monthGraph()** function. In this example, the June's graph button is not going to have any functionality for now because in prototype 2 it is going to show a bar graph created from a MATOLIB library.

Here is also the code for the following Audit graph page and the only changes I made was to the layout of the buttons where I used appjars "stickiness" and grid references to make the layout as close to my final designs as possible.





In the future, this "View Graph Summary" button may have the corresponding month name title giving the page more clarity to the user.

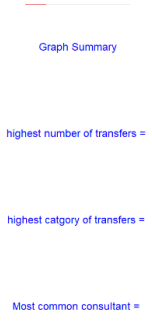
Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

```
#Allows the month clicked on the selection page to display the button required to present that graph
def monthGraph(button):

    if  button=="AG_January":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.setSticky("w")
        app.setPadding([40,40])
        app.addNamedButton("January's graph","January",press,1,0)
        app.setButtonBg("January", "lightblue")


    elif button=="AG_February":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.addNamedButton("February's graph","February",press,1,0)
        app.setButtonBg("February", "lightblue")

    elif button=="AG_March":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.addNamedButton("March's graph","March",press,1,0)
        app.setButtonBg("March", "lightblue")


    elif button=="AG_April":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.addNamedButton("April's graph","April",press,1,0)
        app.setButtonBg("April", "lightblue")

    elif button=="AG_May":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.addNamedButton("May's graph","May",press,1,0)
        app.setButtonBg("May", "lightblue")


    elif button=="AG_June":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.addNamedButton("June's graph","June",press,1,0)
        app.setButtonBg("June", "lightblue")


    elif button=="AG_July":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
```

```
!##Audit Graphs####
    app.startSubWindow("AuditGraphs",0,1)
    app.setSticky("n")
    app.setPadding([40,40])
    app.addLabel("title14","DSW Transfer Audit",0,0,2,1)
    app.setLabelBg("title14", "white")
    app.setLabelFg("title14", "blue")
    app.setSize("fullscreen")
    app.setFont(18)
    app.setSticky("e")
    app.setPadding([40,40])
    app.addNamedButton("View Graph Summary","VGS",press,1,1)
    app.setButtonBg("VGS","lightblue")
    app.setBg("white")
    app.setSticky("sw")
    app.setPadding([40,40])
    app.addNamedButton("Back","BackAGP",press)
    app.setButtonBg("BackAGP","lightblue")
```

```
####Graph Summary####
    app.startSubWindow("GraphSummary",0,1)
    app.addLabel("title15","Graph Summary",0,0)
    app.setLabelBg("title15", "white")
    app.setLabelFg("title15", "blue")
    app.setSize("fullscreen")
    app.addLabel("transfers","highest number of transfers = ")
    app.setLabelBg("transfers", "white")
    app.setLabelFg("transfers", "blue")
    app.addLabel("category","highest catgory of transfers = ")
    app.setLabelBg("category", "white")
    app.setLabelFg("category", "blue")
    app.addLabel("Common consultant","Most common consultant = ")
    app.setLabelBg("Common consultant", "white")
    app.setLabelFg("Common consultant", "blue")
    app.setFont(18)
    app.setBg("white")
    app.setSticky("sw")
    app.setPadding([40,40])
    app.addNamedButton("Back","BackGS",press)
    app.setButtonBg("BackGS","lightblue")

    app.stopSubWindow()



createInterphase()
```

This is the graph summary page where the user will be able to view the graphs trends/important data in text. As of now, there Is nothing shown on this page as the graph hasn't n been created yet also it will be filled as the program is used. No errors were made here as this page is a simple layout with a few labels.



During the development of my GUI, I decided to have a function where my user can view their hidden password if needed. This could help in identifying errors made by the user and quicken login time to the program.

```
def showPass():

    app.getEntry("Password :")
    app.hideButton("eye")
    app.showButton("eyehide")


def hidePass():


    app.hideButton("eyehide")
    app.showButton("eye")
```

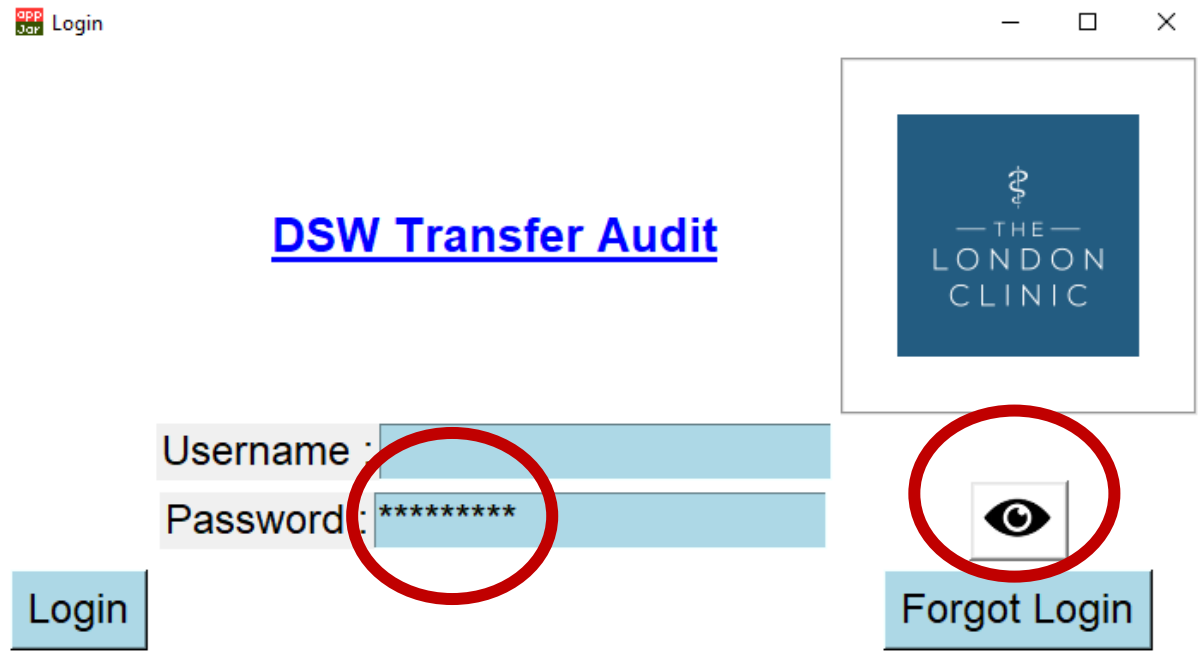
I got the show and hide images presenting and working correctly but the actual password was not able to be revealed or hidden when the show/hide button was clicked. To solve this problem, I created another entry box for the text of the password to be shown when the “eye” button was clicked.

```
#converts the hidden password to the actual password for the user to view
def checkPass():

    app.hideEntry("password:")
    app.hideButton("eyehide")
    app.showButton("eye")


def showPass(btn):
    app.hideEntry("Password :")
    password = app.getEntry("Password :")
    app.showEntry("password:")
    app.setEntry("password:",password)
    app.showButton("eyehide")
    app.hideButton("eye")


def hidePass(btn):

    app.showEntry("Password :")
    app.hideEntry("password:")
    app.showButton("eye")
    app.hideButton("eyehide")
```

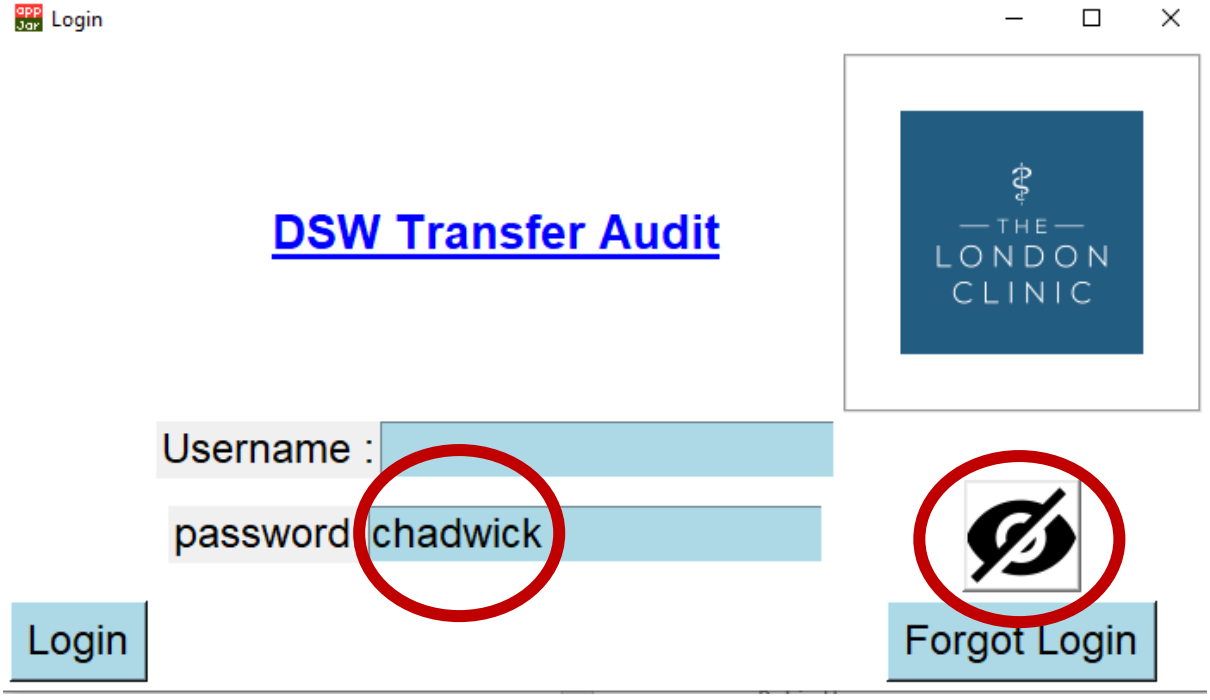
Here is the final show and hide password code. I created functions for the eyehide and eye button which just shows and hides when called in the **createInterphase** function. The entry to show the password text had to be in line with the original entry box by using grid references in appjar. I also made sure that all the adding of buttons/entries were in the **createInterphase** function while the showing/hiding of the buttons were declared in the password functions.

```
#creates my interphase for the program
def createInterphase():
####Login page#########
    app.startSubWindow("Login")
    app.addLabel("title","DSW Transfer Audit" ,0,1
    app.getLabelWidget("title").config(font=("","0", "bold","underline"))
    app.setLabelBg("title", "white")
    app.setLabelFg("title", "blue")
    app.addLabelEntry("Username :",1,1)
    app.addSecretLabelEntry("Password :",2,1)
    app.addLabelEntry("password:",2,1)
    app.addImageButton("eye",showPass,"eye.png",2,2)
    app.addImageButton("eyehide",hidePass,"eyehide.png",2,2)
    app.hideEntry("password:")
    app.hideButton("eyehide")
    checkPass()
    app.setEntryBg("Username :", "lightblue")
    app.setEntryBg("Password :", "lightblue")
    app.setEntryBg("password:", "lightblue")
    app.setFocus("Username :")
    app.setBg("white")
    app.startLabelFrame("", 0, 2)
    app.addImage("tlc", "tlc.png", 0,2)#Adds the company logo
    app.setImageSize("tlc", 200,200)
    app.setGuiPadding(10,10)
```

Hidden Password



Unhidden password

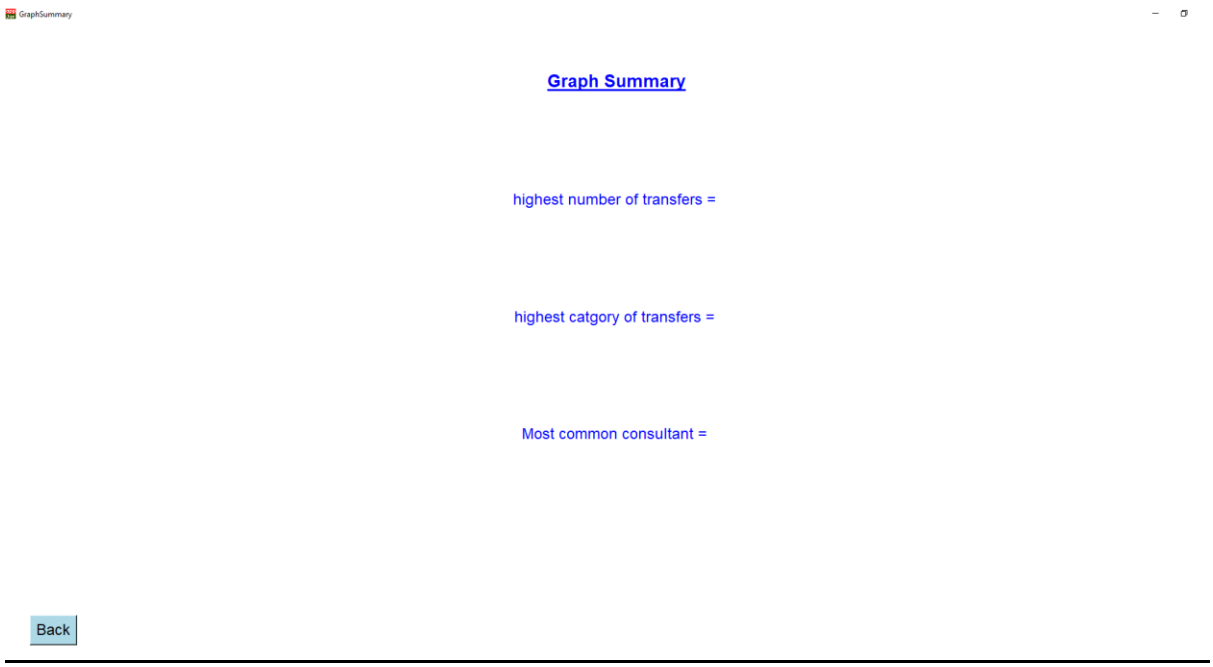


Graph Summary(extra)

This is the graph summary page where each months corresponding graph trends will be displayed and when making the page, I had zero errors because I was only adding a few lines of text to this page. Moreover, this page may not be fully functional until the third prototype as of my programming skill limitations and that adding this feature is not a necessity.

```
    app.setButtonBg("BackAGP", "lightblue")

####Graph Summary####
    app.startSubWindow("GraphSummary")
    app.addLabel("title15","Graph Summary",0,0).config(font=("","20", "bold","underline"))
    app.setLabelBg("title15", "white")
    app.setLabelFg("title15", "blue")
    app.setSize("fullscreen")
    app.addLabel("transfers","highest number of transfers = ",1,0)
    app.setLabelBg("transfers", "white")
    app.setLabelFg("transfers", "blue")
    app.addLabel("category","highest catgory of transfers = ",2,0)
    app.setLabelBg("category", "white")
    app.setLabelFg("category", "blue")
    app.addLabel("Common consultant","Most common consultant = ",3,0)
    app.setLabelBg("Common consultant", "white")
    app.setLabelFg("Common consultant", "blue")
    app.setFont(18)
    app.setBg("white")
    app.setSticky("sw")
    app.setPadding([40,40])
    app.addNamedButton("Back","BackGS",press)
    app.setButtonBg("BackGS","lightblue")

    app.stopSubWindow()



createInterphase()
```

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**



Prototype 1 finished

To add/improve for prototype 2:

1) The main addition I want to implement into my programs second prototype is the functionality of my code such as creating a login system that when a user inputs to log in the values stored in my database are checked. This will also account for the validity of the inputted data. Adding functions like these, will give a sense of realism to nurses when utilizing my program.

2) At this present time when data is added to the grid of either daily or current month table, it is displayed but after the program is closed the data disappears. Therefore, I would like to add a function where all the data is stored permanently in the grid and changes can be made to override the original values stored. This data need to also be transferred to the bar chart created using matplotlib displayed the patient transfers.

3) In addition, I would like to implement the feature of the daily table being able to refresh at a certain time, moving all its current data to the current month table. This will make my program more efficient by separating daily and the entire months data while also making it easier for the user to view information.

4) Lastly, to improve on the presentation of my program, I will adjust the layout of each grid to make it look more professional and rename some buttons to increases the user-friendliness of the different pages.

Number	**Success Criteria Description**	**Has it been met yet?**
1	My program should allow a user to login with the correct credentials.	Partially - I have added a login screen where details can be entered to login but the link between my login system and my database hasn't been set up yet, therefore lacking functionality.
2	The daycase to overnight patient data present in the database should be represented on a bar chart.	No – This function where patient transfers are displayed in bar graphs has not been implemented yet.
3	Information must be logically displayed on each page	Yes
4	My program must be able to display pages in a sensible time.	Yes
5	Data entered into the data base should be valid and stored in the correct format.	Partially – Data can be entered into tables but this is not checked for validation.
6	The colour scheme should be representable of The London Clinic: dark blue and white.	Yes
7	My program should be able to run on a standard desktop/laptop in the clinic.	Yes
8	Daily database for each day should refresh everyday to allow for new data.	No – Function for the Daily table to be refreshed at a certain time hasn't been added yet.
9	The program colour scheme should be same as the OS theme used.	No – Still optional
10	Data can be seen/altered at any time	No – Data cannot be saved after being edited yet
11	Make sure I have an updated database/tables	No – Functionality has not been incorporated yet.

Client Feedback From Working Prototype 1

Question	Client Feedback	How I will respond to this
Does the prototype provide a clear and easy way for nurses to make audits of patient transfers?	Yes, from what I've seen it seems to provide the information we want to gather but I would have to test this going forward with the upcoming prototypes.	I will continue to make sure making an audit for a patient transfer as easy as possible and will allow my client to test this statement between each prototype.
Are you happy with the final design?	Yes, the colour scheme matches the company's brand colours and it is clear and concise. However, I would like the months to be in word format. You could add some images in the background to make the software more user-friendly.	To resolve this, I will change all the month numbers to their actual names and discuss further with my client if any specific images she wants present on each page.
Are you happy with the security of the prototype?	I am happy of the idea of the login security system which operates by using a password for each nurse but I would have to see if there's validity in these security checks.	Acting on this, I will add functionality to the login system where data inputted is checked for blank spaces or character length to make sure the password matches the one stored in the database.
Are you happy with the accessibility and responsiveness of the program to the user?	Yes, I am happy with quick response time to the program.	I will continue to make sure the program flows smoothly and there is no delay in response time.
Are you happy with the current layout of the different table pages?	Yes it brings a basic concept of what it should look like but the transfers reasons key could be made more specific to "Reasons code" for easier understanding and the ID code being after the reason. You should increase the size of the table to read the different categories clearly. Some consultants are not longer	I plan to make the "transfers reasons key" a "Reasons ID code" and make the tables bigger so inputting data is a less challenging task. I also will remove any consultants that are not working at the clinic anymore and add any other consultants if requested by my client.

	practicing at the hospital so some should be removed as they are not participating in the audit.	
Is there anything you don't like about this prototype?	The title should not be abbreviated as "DSW" but show "Day Surgery Ward" and on the table pages make the consultants "key" a "codeID" making it easy to follow for different nurses.	I will make the following changes to the current labels in my program.
Is this prototype, so far, in line with the specification for the system you have requested?	Yes, the program does the job it intends to do and it provides a good level of professionalism.	From this feedback, I will carry on producing high level code making sure the program meets its purpose every time a change is made.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Prototype 2:

Linking the database to my Login system:

Now implementing functionality to my login system, I have created a function to test for a valid login. This compares the user's inputs to the username and passwords stored in the **tbl_Nurse** in my database.

All the passwords are selected from all the usernames in the nurse table. Here, when I entered the correct username and wrong password no error was occurring. To solve this is made sure that if both the username and password was incorrect or only one of them was incorrect it will still display an error pop up box.

In creating this, only when I entered the correct details in the visible mode was when the system allowed me to login, therefore I made it so if the password is visible or not it should still allow for valid inputs. In addition, the SELECT statement from the SQL here retrieved the password from my **tbl_Nurse** in the database based on the entered username allowing for comparison with the user's entered details.

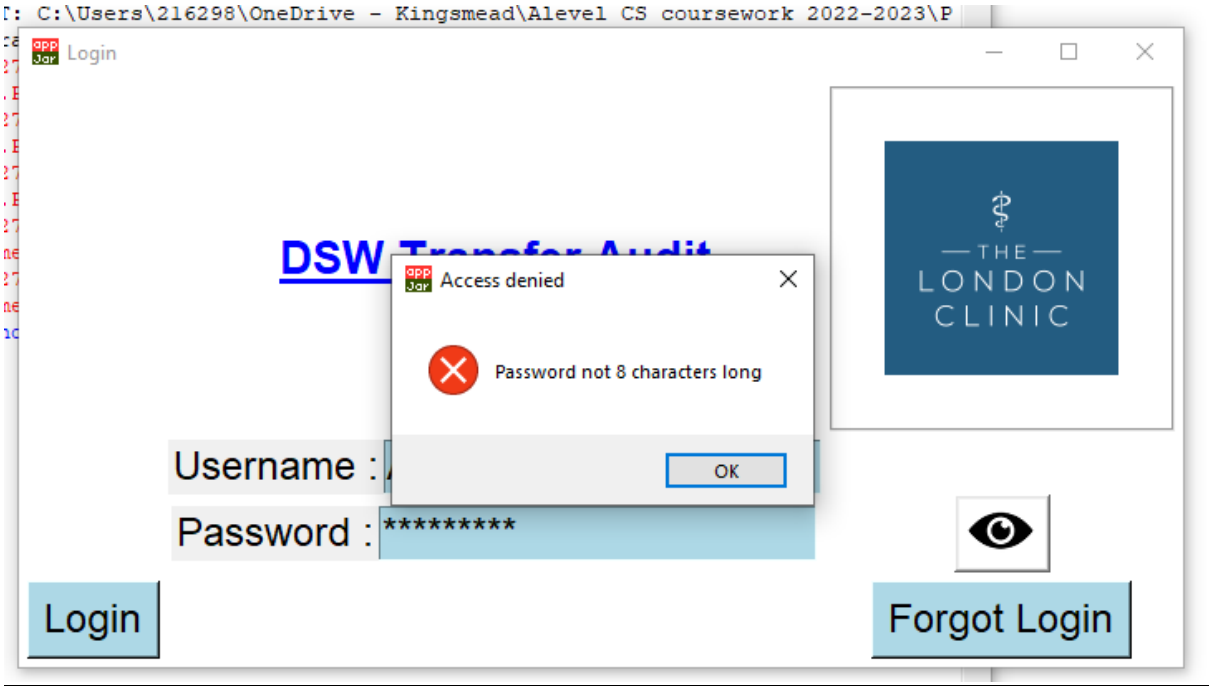
```
#is alphanum = alphabet and number
#is alpha = checks all characters is alphabet
#is digit = checks digit only

#mypassword[0].isupper()
#Allows user to only login if they enter valid inputs that match the value stored in the database
def validLogin():
    global hidden_password
    username = app.getEntry("Username :")#sets username variable to the username entered by the user
    #password can be written in either hidden or visible mode
    hidden_password = app.getEntry("hiddenPassword")#hidden  password
    visible_password = app.getEntry("visiblePassword")#visible password
    cur.execute("SELECT password FROM tbl_Nurse WHERE username=?", [username])#selects password from the database associated with the  entered username
    result=cur.fetchone()#fetches the passwords from the queried username in the database and stores it in the variable result
    print(result,hidden_password,visible_password)
    if len(hidden_password)==0 and len(visible_password)!=0:
        hidden_password=visible_password#if hidden password is empty visible password can be stored as the hidden password
    if len(visible_password)==0 and len(hidden_password)!=0:
        visible_password=hidden_password#if visible password is empty hidden password can be stored as the visible password

#Checks username is 6 characters long
    if len(username)!=6:
        app.errorBox("Access denied","Username not 6 characters long")
#If either password entry box is not 8 characters long user is denied
    elif len(hidden_password)!=8 and len(visible_password)!=8:
        app.errorBox("Access denied","Password not 8 characters long")

#Checks either password entry boxes have a capital letter

    elif hidden_password[0].islower() or visible_password[0].islower():
        app.errorBox("Access denied","Password not in correct format")

#Makes sure the only way to login is if the username and password matches exactly
    elif result==None or (result[0]==username and (result[0]!=hidden_password  or result[0]!=visible_password)):
         app.errorBox("Access denied","Username or Password is incorrect")

#if credentials are valid user can login successfully
    else:
        if result[0]==hidden_password  or result[0]==visible_password:
            app.hideSubWindow("Login")#User may login as details are correct
            app.clearEntry("visiblePassword")#clears password entry when user logs in
            app.clearEntry("hiddenPassword")
            app.showSubWindow("MainMenu")
```

In this section of the ValidLogin subroutine login credentials are checked different types of validity. Using the **len** function I made sure the username could only hold 6 characters and the password could only hold 8 characters linking back to where I mentioned about this in my test table. I then made sure the details were only correct if each username and password has a capital letter. If the result of the inputted data is not equal to the stored values in the database access will denied, making the user to try re-enter their details. However, if the users details match the ones in the database they will be able to login into the software successfully.

```
#if credentials are valid user can login successfully
    else:
        if result[0]==hidden_password  or result[0]==visible_password:
            app.hideSubWindow("Login")#User may login as details are correct
            app.clearEntry("Password :")#clears password entry when user logs in
            app.clearEntry("password:")
            app.showSubWindow("MainMenu")
```

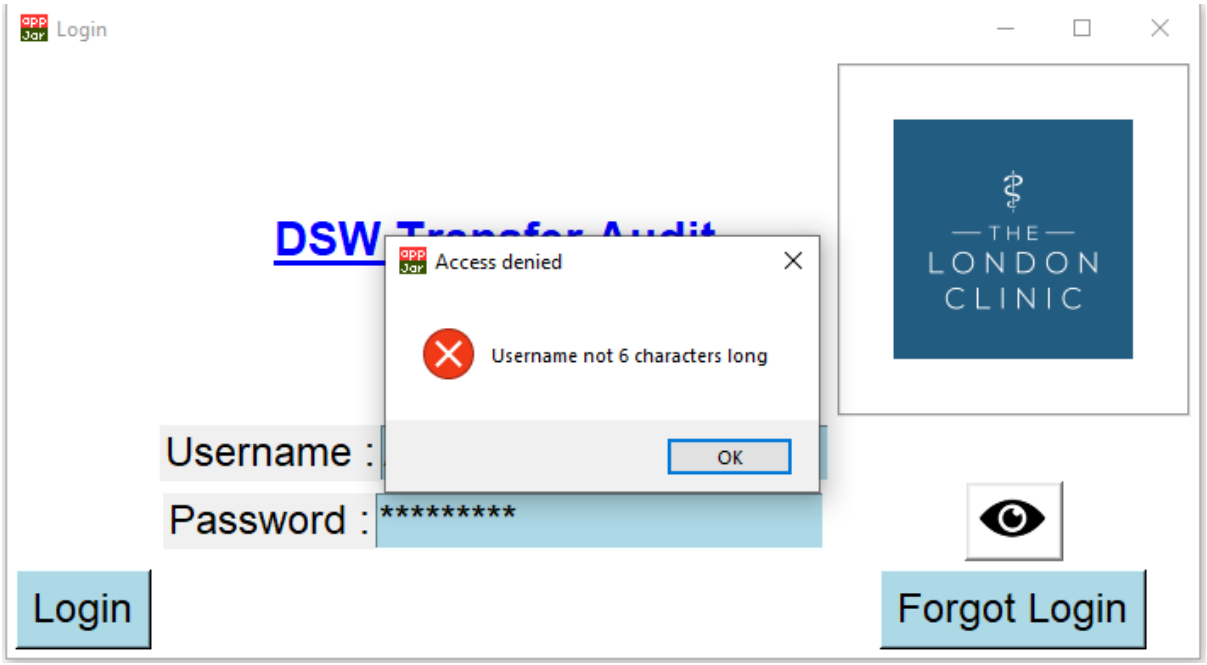
I added a subtle extra feature that when a user logs into the system whatever they types into the password field is deleted increasing security of data and to prevent other nurses who will use the system seeing the previous persons login details if they ever logout of the previous nurses account. This also makes the software look more cleaner as redundant data is not left uncared for and therefore provides the user interface with a more professional look.

Correct username and incorrect password are inputted:



If password is incorrect this message is displayed

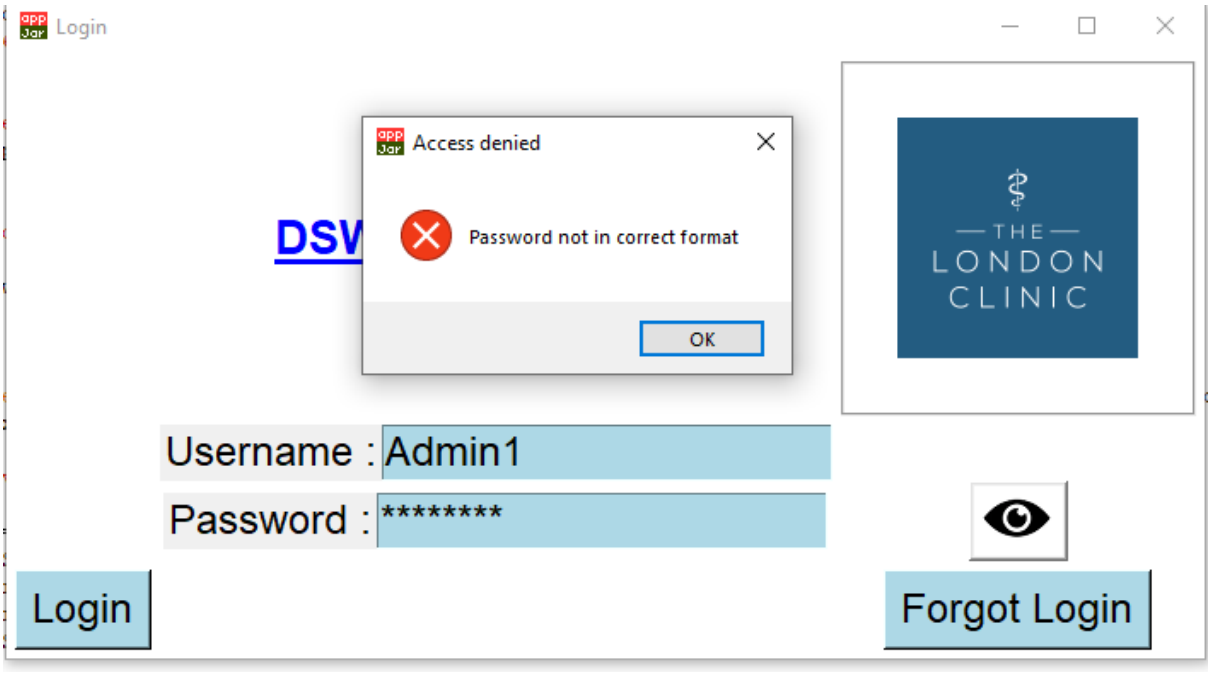
Incorrect username and correct password are inputted:



This is displayed when the incorrect username is inputted while a correct password is inputted.

Correct username and password which is incorrect:

This is displayed when the correct username and incorrect password is inputted even if it is 8 characters.



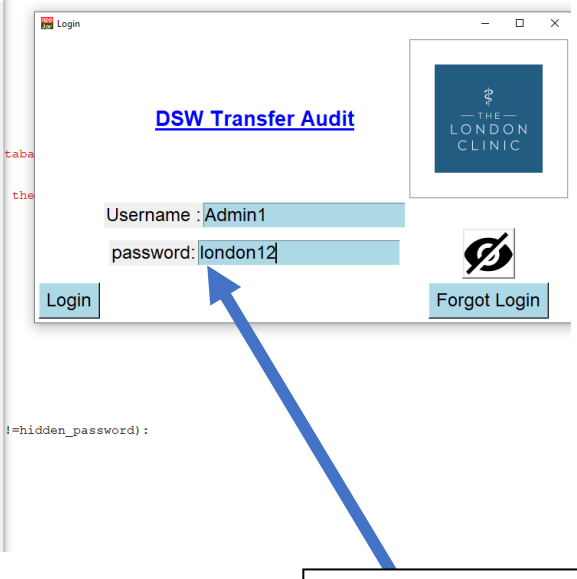
While coding I realised my variable names for my show and hide password subroutine were too confusing when creating the validity of my login page. Therefore, to make things clearer and easier for someone viewing my code to understand I changed the variable names to **"visiblePassword"** and **"hiddenPassword"** which also makes it easier for me to differentiate between the two.

```
#converts the hidden password to the actual password for the user to view
def checkPass():

    app.hideEntry("password:")
    app.hideButton("eyehide")
    app.showButton("eye")


def showPass(btn):
    app.hideEntry("Password :")
    password = app.getEntry("Password :")
    app.showEntry("password:")
    app.setEntry("password:",password)
    app.showButton("eyehide")
    app.hideButton("eye")


def hidePass(btn):


    app.showEntry("Password :")
    app.hideEntry("password:")
    app.showButton("eye")
    app.hideButton("eyehide")
```

```
#mypassword[0].isupper()
#Allows user to only login if they enter valid inputs that match the value stored in the database
def validLogin():
    global hidden_password
    username = app.getEntry("Username :")#sets username variable to the username entered by the user
    #password can be written in either hidden or visible mode
    hidden_password = app.getEntry("hiddenPassword")#hidden  password
    visible_password = app.getEntry("visiblePassword")#visible password
    cur.execute("SELECT password FROM tbl_Nurse WHERE username=?", [username])#Selects all usernames and passwords from the
    result=cur.fetchone()#fetches all the usernames and passwords from the database and stores it in the variable result
    print(result,hidden_password,visible_password)
    if len(hidden_password)==0 and len(visible_password)!=0:
        hidden_password=visible_password#if hidden password is empty visible password can be stored as the hidden password
    if len(visible_password)==0 and len(hidden_password)!=0:
        visible_password=hidden_password#if visible password is empty hidden password can be stored as the visible password
```

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

While creating my validLogin() subroutine I received this error when checking for capital letter in the password and this error was occurring because the program was checking the index of the other entry box. As this was, empty the program was returning an index out of error, so to fix this I made it so if the password is typed in the hidden password box and the visible password box is empty, when the login button is pressed whatever is stored in the hidden password box would be made **equal** to the visible password variable. This will be carried out vice versa. Therefore, the passwords will be the same in both input boxes and when the user logs in the correct process will occur.



Checks password for capital letter. If there is not a capital letter an invalid format error message is displayed.

```
#Allows user to only login if they enter valid inputs that match the value stored in the database
def validLogin():
    global hidden_password
    username = app.getEntry("Username :")#sets username variable to the username entered by the user
    #password can be written in either hidden or visible mode
    hidden_password = app.getEntry("hiddenPassword")#hidden  password
    visible_password = app.getEntry("visiblePassword")#visible password
    cur.execute("SELECT password FROM tbl_Nurse WHERE username=?", [username])#Selects all usernames and passwords from the
    result=cur.fetchone()#fetches all the usernames and passwords from the database and stores it in the variable result
    print(result,hidden_password,visible_password)
    if len(hidden_password)==0 and len(visible_password)!=0:
        hidden_password=visible_password#if hidden password is empty visible password can be stored as the hidden password
    if len(visible_password)==0 and len(hidden_password)!=0:
        visible_password=hidden_password#if visible password is empty hidden password can be stored as the visible password
```

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

```
2023-03-02 18:56:46,815 appJar:WARNING [Line 417->5210/getGridEntries]: Deprecated - grids renamed to tables
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\rohin\AppData\Local\Programs\Python\Python39\lib\tkinter\__init__.py", line 1892, in __call__
    return self.func(*args)
  File "C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming project-DESKTOP-PR2B3J6-DESKTOP-PR2B3J6\appJar\appjar.py", line 14530, in <lambda>
    command=lambda row=val, *args: self.action(row)
  File "C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming project-DESKTOP-PR2B3J6-DESKTOP-PR2B3J6\Patient Transfer Audit (2).py", line 420, in show
    consultant_id = row[0]#checks each row for data
IndexError: string index out of range
```

When I was creating the function to retrieve data from my table to store in the database, I came across this error where because the row was empty there was no indexes hence the index error, to fix this I made it so the program would continue if an empty space was encountered and only store the actual data in variables to be inserted into the database.

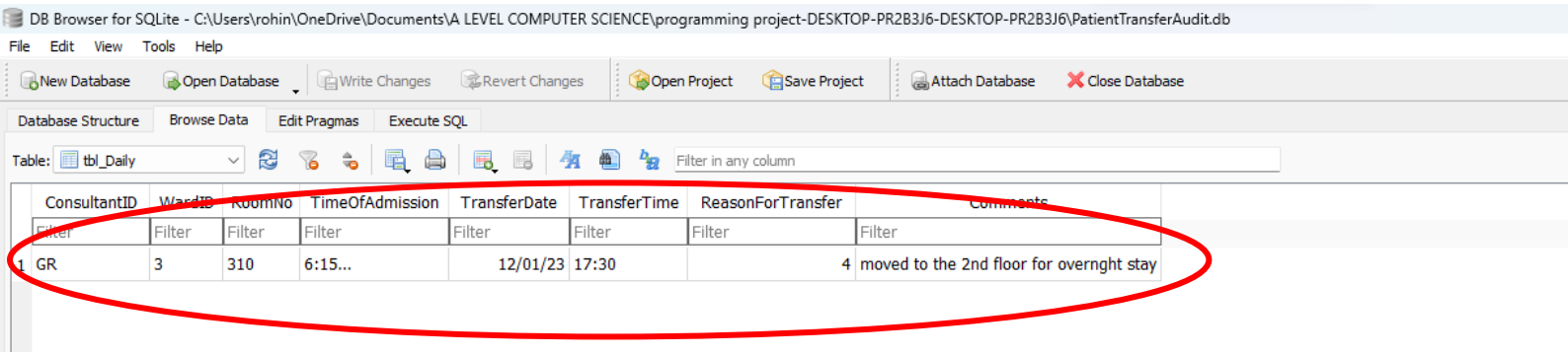
```
    return self.func(*args)
  File "C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming project-DESKTOP-PR2B3J6-DESKTOP-PR2B3J6\appJar\appjar.py", line 14530, in <lambda>
    command=lambda row=val, *args: self.action(row)
  File "C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming project-DESKTOP-PR2B3J6-DESKTOP-PR2B3J6\Patient Transfer Audit (2).py", line 442, in
    cur.execute("INSERT INTO tbl_Daily (ConsultantID, WardID, RoomNo, TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments) VALUES (?, ?, ?, ?,
_id, ward_id, Room_No, Time_Of_Admission, transfer_date, transfer_time, reason_for_transfer, comments))
sqlite3.IntegrityError: UNIQUE constraint failed: tbl_Daily.ConsultantID
```

While creating the function to save the data from my table and inserting it into my database, I came across this error where the consultantID field in my daily table was unique and it was not allowing inputted data to override unique data being stored because the inputted data may be repeated, thus making the field not a primary key. Therefore, to fix this I decided to make my Daily table and Current month table to have no primary keys. In doing so, this becomes a limitation to the maintenance to my data, however all I am to go to use these tables for is inserting all data to and retrieving all data from, which does not really require a primary key as data is being moved in and out of the database quickly. These tables in my database also have a one-to-one relationship and contain no foreign keys, reinforcing why I don't need a primary key in these tables.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

```
def SaveTable(btn):
    conn = sqlite3.connect('PatientTransferAudit.db')#connect db
    cur = conn.cursor()

    data = app.getGridRow("g1", btn)#gets inputs from the row and stores it in the variable data
    print(data)
    if data is None:
        return print("No data")#prints no data if row is empty
    else:
        for row in data:
            print(row)
            if any(row):#for any row in the table
                consultant_id = data[0]#assigns each row index to the corresponding field name
                ward_id = data[1]
                Room_No = data[2]
                Time_Of_Admission = data[3]
                transfer_date = data[4]
                transfer_time = data[5]
                reason_for_transfer = data[6]
                comments = data[7]


    cur.execute("INSERT INTO tbl_Daily (ConsultantID, WardID, RoomNo, TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments) VALUES (?, ?, ?, ?, ?, ?, ?, ?)", (consultan
    conn.commit()#commits all changes into the tbl_Daily in my database
    print("complete")
```

This is the final code for my SaveTableg1() function that retrieves the data from each row in the daily table stores each inputted data in a variable of the correct field name. This is then inserted into the tbl_Daily in my database where it can be permanently stores until data needs to be retrieved again.

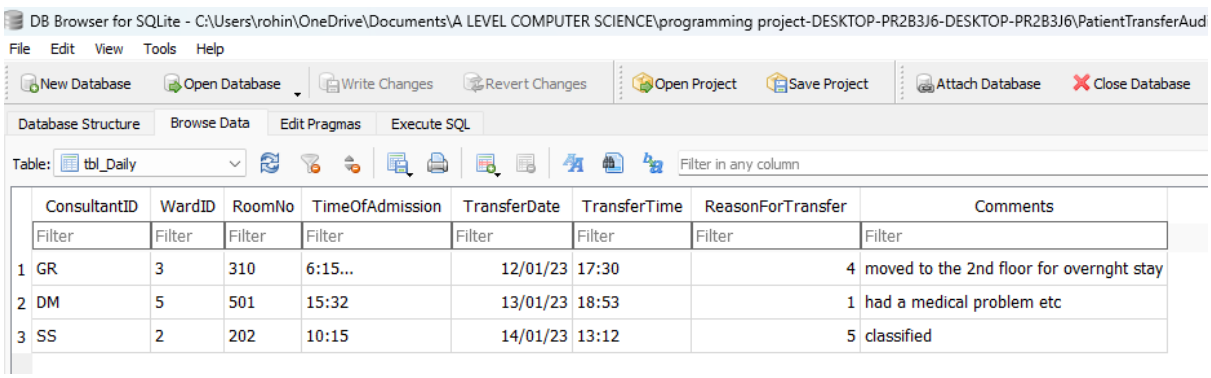


Daily

GR	3	310	6:15	12/01/23	17:30
dog	cat	3	12:45	8/01/23	14:56
RAGGY	5	6	rag time	sc00b	16:40

The data in the tbl_Daily then needs to be retrieved back into the daily grid in my program when the user tries to access the table. For this to work, I had to fetch all the data from the table and replace all the rows in the grid with the data. Although when I did this, it also replaced the important headers from the grid. To solve this problem, I stored the first row in a variable and added it to the end of the grid so the data was not lost maintaining data integrity. Finally, I re-added the table headers to the top of the table.

```
def retrive_tbl_Daily():
    conn = sqlite3.connect('PatientTransferAudit.db')#connect db
    cur = conn.cursor()
    cur.execute("SELECT* FROM tbl_Daily")#selects all data from the table
    rows = cur.fetchall()#stores all the data in the variable rows
    headers = ["ConsultantID", "WardID","RoomNo", "TimeOfAdmission","TransferDate","TransferTime","ReasonForTransfer","Comments"]#as
    app.replaceAllGridRows("g1",rows)#replaces all the rows in my grid with the retrived data
    first_row = app.getGridRow("g1",-1)#stores the first row that is currently in the header in the variable first_row
    app.setGridHeaders("g1",headers)#set the grid headers to the headers of the Daily grid
    app.addGridRow("g1",first_row)#adds the first row to the end of the existing grid
```



	ConsultantID	WardID	RoomNo	TimeOfAdmission	TransferDate	TransferTime	ReasonForTransfer	Comments
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	GR	3	310	6:15...	12/01/23	17:30	4	moved to the 2nd floor for overnght stay
2	DM	5	501	15:32	13/01/23	18:53	1	had a medical problem etc
3	SS	2	202	10:15	14/01/23	13:12	5	classified

This is the data in tbl_daily being displayed in the daily grid in my program.

Daily

ConsultantID	WardID	RoomN
DM	5	501
SS	2	202
GR	3	310

Transferring data from the Daily to current month table:

At 8pm every day, all the data from the daily table will be transferred into the current month table and the daily table will refresh ready for new data to be collected the next day.

```
def TransferDailyTable():
    seconds = time.time()#gets the current time in seconds
    local_time = time.ctime(seconds)#converts the time in seconds into a readable format
    print("Local time:", local_time)#prints the readable time
    current_time = time.localtime()#gets the struct_time function in format(year, month, day, hour, minute, second, weekday, day of the year, daylight saving)
    hour = time.strftime("%H:%M:%S",current_time)#puts the current time in H/M/S format
    str_hour = (str(hour))#makes hour variable a string
    str_hour=str_hour.strip()
    hour,minute,second=str_hour.split(":")#splits the time into single integers of hour,minute and second
    if hour >== "20":
        rows = app.getGridRowCount("g1")#gets number of rows with data
        print(rows)
        rowData=[]#empty list to transfer data
        for x in range(rows):
            data=app.getGridRow("g1",x)#for each row in the table store values in the variable data
            rowData.append([data])#3D array

        row_Num = 0
        for outer_list in rowData:
            for inner_list in outer_list:#retrieves the rows from the 3D list
                print(inner_list)
            app.addGridRow("g2",inner_list)#adds the rows from g1 to g2

    app.deleteAllGridRows("g1")
```

This is the code for transferring data. I used the `import time` library for the table to carry out this transfer function at 8pm. I converted the time into a string so that it could be split by (:) using the line.strip() command. If the hour was greater or equal to 20 (considering the 24-hour clock) the rest of the code would take place.

The function iterates over each row in “g1” and stores it in the variable data. This data is added to an empty list called rowData[] creating a 3D list. To retrieve the rows, I wanted I created a for loop which only took the “inner_list” and added it to the current month table. Once this transfer has been completed the data in the daily grid is cleared, ready for a new batch of patient transfers coming the following day.

#MAKE IT SO THE ADDED DATA GOES TO THE TOP OF THE GRID

However, when adding the time, I came across this error.

```
  File "C:\Users\216298\OneDrive - Kingsmead\Alevel CS coursework 2022-2023\Patient Tr
Audit.py", line 474, in TransferDailyTable
    if hour(int("%H:%M:%S")) == hour(20,00,00):
ValueError: invalid literal for int() with base 10: '%H:%M:%S'
```

After some research, I realised that the time must be in string format to proceed on removing the (:) from the time as you can only split data when they are string datatypes.

The solution:

```
str_hour = (str(hour))#makes hour variable a string
str_hour=str_hour.strip()
hour,minute,second=str_hour.split(":")#splits the time into single integers of hour,minute and second
```

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Before transfer

Daily

ConsultantID	WardID	RoomN
ST	4	401
W	1	111
GR	3	310

CurrentMonth

ConsultantID	WardID	RoomN

After transfer

Daily

ConsultantID	WardID	RoomN

CurrentMonth

ConsultantID	WardID	RoomN
ST	4	401
W	1	111
GR	3	310

```
###Month Display PD####
    app.startSubWindow("MonthDisplayPD")
    app.setSize("fullscreen")
    app.addLabel("title6","Months")
    app.setLabelBg("title6", "white")
    app.setLabelFg("title6", "blue")
    app.setFont(18)
    app.setBg("white")
    app.addNamedButton("January","PD_January",press,1,0)
    app.addNamedButton("February","PD_February",press,2,0)
    app.addNamedButton("March","PD_March",press,3,0)
    app.addNamedButton("April","PD_April",press,4,0)
    app.addNamedButton("May","PD_May",press,5,0)
    app.addNamedButton("June","PD_June",press,6,0)
    app.addNamedButton("July","PD_July",press,7,0)
    app.addNamedButton("August","PD_August",press,8,0)
    app.addNamedButton("September","PD_September",press,9,0)
    app.addNamedButton("October","PD_October",press,10,0)
    app.addNamedButton("November","PD_November",press,11,0)
    app.addNamedButton("December","PD_December",press,12,0)
    app.setButtonBg("PD_January", "lightblue")
    app.setButtonBg("PD_February", "lightblue")
    app.setButtonBg("PD_March", "lightblue")
    app.setButtonBg("PD_April", "lightblue")
    app.setButtonBg("PD_May", "lightblue")
    app.setButtonBg("PD_June", "lightblue")
    app.setButtonBg("PD_July", "lightblue")
    app.setButtonBg("PD_August", "lightblue")
    app.setButtonBg("PD_September", "lightblue")
    app.setButtonBg("PD_October", "lightblue")
    app.setButtonBg("PD_November", "lightblue")
    app.setButtonBg("PD_December", "lightblue")
    app.setSticky("sw")
    app.setPadding([40,40])
    app.addNamedButton("Back","backMPD",press)
    app.setButtonBg("backMPD", "lightblue")

    app.stopSubWindow()
```

Considering my clients opinion, I am changing all the month buttons from numbers to the actual name of the months improving my user interface.

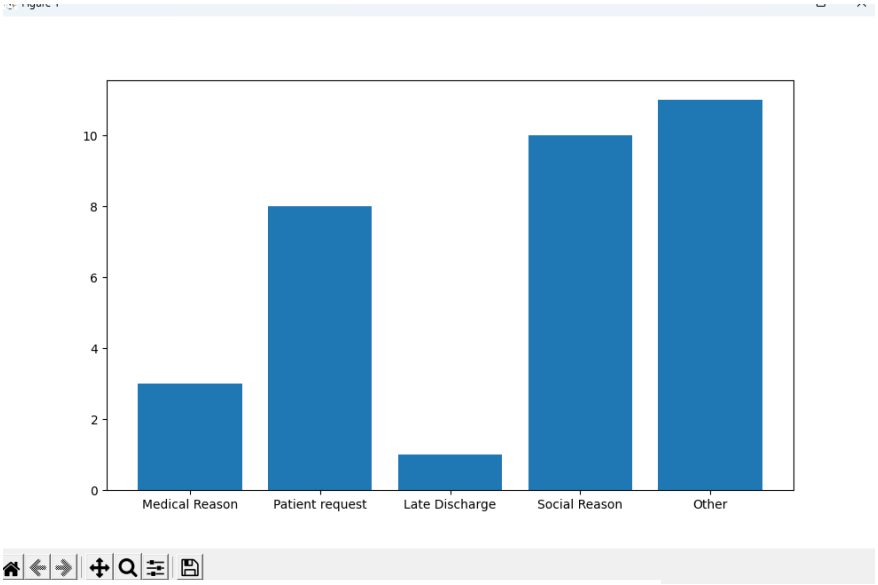
Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Creating the Month Graph

Imported the matplotlib library into my python program so I can start creating the bar graph which holds the tables data inputted by the nurses, tracking the number of patient transfers.

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
```

First, I checked if the library was working correctly by creating a bar graph with random values to see what it would eventually look like also allowing me to get used to the different library functions.




```
def JanurayMonthlyGraph():


#run graph only at the end of the month at 8pm and do this for each month (could use another subroutine for all this)
#need the month graph ready for each year potentially or leave it and say will be done in th future
    x = np.array(["Medical Reason", "Patient request", "Late Discharge", "Social Reason","Other"])#x-axis
    y = np.array([3, 8, 1, 10,11])#y-axis

    plt.figure(figsize=(10,6))#size of bar graph
    plt.bar(x,y)#plot bar graph
    plt.show()#show bar graph
```

To create the inputs for the bar graph I needed to total each different reason for patient transfers in the month. In doing so, I had to construct these total variables to have a global scope so they could be used within other subroutines in my code.

```
MedicalReason_Total = 0 #total for medical reasons
PatientRequest_Total = 0#total for patient requests
LateDischarge_Total = 0#total for late discharges
ConsultantRequest_Total = 0#total for consultant requests
SocialReason_Total = 0#total for social reasons
Other_Total = 0#total for other/unknown reasons
def AuditGraph():
    global MedicalReason_Total#refrence the global variables
    global PatientRequest_Total
    global LateDischarge_Total
    global ConsultantRequest_Total
    global SocialReason_Total
    global Other_Total
    rows = app.getGridRowCount("g2")
    print(rows)
    GraphData=[]#empty list to store data
    for x in range(rows):
        data=app.getGridRow("g2",x)#for each row in the table store values in the variable data
        if data:# check if the current row contains any non-empty elements
            consultant_id = data[0]#assigns each row index to the corresponding field name
            ward_id = data[1]
            Room_No = data[2]
            Time_Of_Admission = data[3]
            transfer_date = data[4]
            transfer_time = data[5]
            reason_for_transfer = data[6]
            comments = data[7]
            print(reason_for_transfer)


            if "1" in reason_for_transfer:
                MedicalReason_Total+=1 #increment medical reason tally by one

            elif "2" in reason_for_transfer:
                PatientRequest_Total+=1

            elif "3" in reason_for_transfer:
                LateDischarge_Total+=1

            elif "4" in reason_for_transfer:
                ConsultantRequest_Total+=1

            elif "5" in reason_for_transfer:
                SocialReason_Total+=1


            elif "6" in reason_for_transfer:
                Other_Total+=1


    print("total is " , SocialReason_Total)
```

This "AuditGrpah()" subroutine allows the sum of the different reasons of transfer to be calculated. This function works similar to my "SaveTableg1()" subroutine. Above, I have circled the global variables I mentioned before. Here, I have again taken all the data from all the rows but now in the current month grid. Each cell, in each row represents a specific field like consultant ID, WardID etc. If the cell contains data for any of the rows it is stored to the corresponding field named variable. As I wanted to focus on obtaining the reason of transfer information I made an if statement to count each of the reasons from every row and held that number in the global variables so they could be easily added into my **MonthlyGraph()** subroutines.

```
import datetime as dt
import schedule
```

To insert the values for the total of the different number of reasons from patient transfers I must make sure the values from the **correct** month retrieved and used to construct the audit graph. To do this I have added an extra field in my current month and daily table in my database and a column called "month" in my grids so this can store whether the grid is for January,February etc. For example, if this part of the grid contains "January" the totals for January will be retrieved from the grid and used to make the audit graph. This functionality has been created in the new function "check_month()".

```
con.commit()
#This creates the Daily audit table in the DB
cur.execute("""CREATE TABLE "tbl_Daily" (
    "ConsultantID"  TEXT NOT NULL,
    "WardID"        TEXT NOT NULL,
    "RoomID"        TEXT NOT NULL,
    "TimeOfAdmission"       time NOT NULL,
    "TransferDate"  date NOT NULL,
    "TransferTime"  time NOT NULL,
    "ReasonForTransfer"     INTEGER NOT NULL,
    "Comments"      TEXT NOT NULL,
    "Month" TEXT NOT NULL
)""")

con.commit()

#This creates the CurrentMonth table in the DB
cur.execute("""CREATE TABLE "tbl_CurrentMonth" (
    "ConsultantID"  TEXT NOT NULL,
    "wardID"        TEXT NOT NULL,
    "RoomNo"        INTEGER NOT NULL,
    "TimeOfAdmission"       time NOT NULL,
    "TransferDate"  date NOT NULL,
    "TransferTime"  time NOT NULL,
    "ReasonsforTransfer"    INTEGER NOT NULL,
    "Comments "     TEXT NOT NULL,
    "Month" TEXT NOT NULL

)""")
con.commit()
```

To add an extra column in my database I deleted my current created tables and added the new column of "Month" then ran the setupDB() function again to recreate the tables.

```
def retrive_tbl_Daily():
    conn = sqlite3.connect('PatientTransferAudit.db')#connect db
    cur = conn.cursor()
    cur.execute("SELECT ConsultantID, WardID, RoomID, TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments FROM tbl_Daily")#selects all rows with data from the table
    rows = cur.fetchall()
    headers = ["ConsultantID", "WardID","RoomID", "TimeOfAdmission","TransferDate","TransferTime","ReasonForTransfer","Comments"]#assigns all the titles for each column in the grid to the v
    app.replaceAllGridRows("g1",rows)#replaces all the rows in my grid with the retrived data
    first_row = app.getGridRow("g1",-1)#stores the first row that is currently in the header in the variable first_row
    app.setGridHeaders("g1",headers)#set the grid headers to the headers of the Daily grid
    app.addGridRow("g1",first_row)#adds the first row to the end of the existing grid
```

I had to change how the rows were retrieved from the database by now selecting all the fields apart from the Month field as it contains no data in it.

This is the following code for check_month**g2**() function:

```
def check_month(button):
    if button == "PD_January":
        rows = app.getGridRowCount("g2")#counts the numbers of rows in the grid
        Month_name = "January"
        for i in range(rows):
            print(rows)
            row = app.getGridRow("g2", i)#gets all the current row and stores it in variable row
            if not any(row):#skips the rows that are completely empty
                continue
            if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
                row[8] = Month_name
                app.replaceGridRow("g2", i, row)#Add the month_name to each of the rows
```

Here when the January button is clicked, the number of rows is stores in the variable **rows** and that data that needs to be inserted into the Month column in my grid is stored in **Month_name**. Then the program iterates through the grid storing the current row it is on in **current_row**. "Any()"returns the condition false if the row only contains false values so not any() means if the row contains only false values, in other words, it is completely empty, the row will be skipped, therefore making the condition true. As this may not be the first time the user has gone into the January table, the code checks if "January" has already been added to all the month column in my Grid. If not then row[8] will be set to "January" for all the rows and then the updated rows will be displayed. This is only for the January grid so I repeated the same code for all the other months and this can be shown below.

```
File Edit Format Run Options Window Help
    elif button == "PD_February":
        rows = app.getGridRowCount("g2")#counts the numbers of rows in the grid
        Month_name = "February"
        for i in range(rows):
            row = app.getGridRow("g2", i)#gets all the current row and stores it in variable row
            if not any(row):#skips the rows that are completely empty
                continue
            if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
                    row[8] = Month_name
                    app.replaceGridRow("g2", i, row)#Add the month_name to each of the rows


    elif button == "PD_February":
        rows = app.getGridRowCount("g2")#counts the numbers of rows in the grid
        Month_name = "February"
        for i in range(rows):
            row = app.getGridRow("g2", i)#gets all the current row and stores it in variable row
            if not any(row):#skips the rows that are completely empty
                continue
            if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
                    row[8] = Month_name
                    app.replaceGridRow("g2", i, row)#Add the month_name to each of the rows


    elif button == "PD_March":
        rows = app.getGridRowCount("g2")#counts the numbers of rows in the grid
        Month_name = "March"
        for i in range(rows):
            row = app.getGridRow("g2", i)#gets all the current row and stores it in variable row
            if not any(row):#skips the rows that are completely empty
                continue
            if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
                    row[8] = Month_name
                    app.replaceGridRow("g2", i, row)#Add the month_name to each of the rows



    elif button == "PD_April":
        rows = app.getGridRowCount("g2")#counts the numbers of rows in the grid
        Month_name = "April"
        for i in range(rows):
            row = app.getGridRow("g2", i)#gets all the current row and stores it in variable row
            if not any(row):#skips the rows that are completely empty
                continue
            if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
                    row[8] = Month_name
                    app.replaceGridRow("g2", i, row)#Add the month_name to each of the rows
```

```
elif button == "PD_May":
     rows = app.getGridRowCount("g2")#counts the numbers of rows in the grid
     Month_name = "May"
     for i in range(rows):
        row = app.getGridRow("g2", i)#gets all the current row and stores it in variable row
        if not any(row):#skips the rows that are completely empty
            continue
        if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
               row[8] = Month_name
               app.replaceGridRow("g2", i, row)#Add the month_name to each of the rows



elif button == "PD_June":
     rows = app.getGridRowCount("g2")#counts the numbers of rows in the grid
     Month_name = "June"
     for i in range(rows):
        row = app.getGridRow("g2", i)#gets all the current row and stores it in variable row
        if not any(row):#skips the rows that are completely empty
            continue
        if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
               row[8] = Month_name
               app.replaceGridRow("g2", i, row)#Add the month_name to each of the rows


elif button == "PD_July":
     rows = app.getGridRowCount("g2")#counts the numbers of rows in the grid
     Month_name = "July"
     for i in range(rows):
        row = app.getGridRow("g2", i)#gets all the current row and stores it in variable row
        if not any(row):#skips the rows that are completely empty
            continue
        if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
               row[8] = Month_name
               app.replaceGridRow("g2", i, row)#Add the month_name to each of the rows


elif button == "PD_August":
     rows = app.getGridRowCount("g2")#counts the numbers of rows in the grid
     Month_name = "August"
     for i in range(rows):
        row = app.getGridRow("g2", i)#gets all the current row and stores it in variable row
        if not any(row):#skips the rows that are completely empty
            continue
        if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
               row[8] = Month_name
               app.replaceGridRow("g2", i, row)#Add the month_name to each of the rows
```

```
elif button == "PD_September":
     rows = app.getGridRowCount("g2")#counts the numbers of rows in the grid
     Month_name = "September"
     for i in range(rows):
        row = app.getGridRow("g2", i)#gets all the current row and stores it in variable row
        if not any(row):#skips the rows that are completely empty
            continue
        if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
               row[8] = Month_name
               app.replaceGridRow("g2", i, row)#Add the month_name to each of the rows



elif button == "PD_October":
     rows = app.getGridRowCount("g2")#counts the numbers of rows in the grid
     Month_name = "October"
     for i in range(rows):
        row = app.getGridRow("g2", i)#gets all the current row and stores it in variable row
        if not any(row):#skips the rows that are completely empty
            continue
        if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
               row[8] = Month_name
               app.replaceGridRow("g2", i, row)#Add the month_name to each of the rows


elif button == "PD_November":
     rows = app.getGridRowCount("g2")#counts the numbers of rows in the grid
     Month_name = "November"
     for i in range(rows):
        row = app.getGridRow("g2", i)#gets all the current row and stores it in variable row
        if not any(row):#skips the rows that are completely empty
            continue
        if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
               row[8] = Month_name
               app.replaceGridRow("g2", i, row)#Add the month_name to each of the rows


elif button == "PD_December":
     rows = app.getGridRowCount("g2")#counts the numbers of rows in the grid
     Month_name = "December"
     for i in range(rows):
        row = app.getGridRow("g2", i)#gets all the current row and stores it in variable row
        if not any(row):#skips the rows that are completely empty
            continue
        if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
               row[8] = Month_name
               app.replaceGridRow("g2", i, row)#Add the month_name to each of the rows
```

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Following this I made an adjustment to the AuditGraph() subroutine. After some research using **w3resource**, I realised to allow for the totals of the different reasons for transfer to be calculated every month I would need to use the calendar library and previously used time library. Here, I integrated the import time library with the calendar function in my code.

```
ConsultantRequest_Total = 0#total for consultant requests
SocialReason_Total = 0#total for social reasons
Other_Total = 0#total for other/unknown reasons
def AuditGraph():
    global MedicalReason_Total#refrence the global variables
    global PatientRequest_Total
    global LateDischarge_Total
    global ConsultantRequest_Total
    global SocialReason_Total
    global Other_Total
    seconds = time.time()#gets the current time in seconds
    local_time = time.ctime(seconds)#converts the time in seconds into a readable format
    print("Local time:", local_time)#prints the readable time
    current_time = time.localtime()#gets the struct_time function in format(year, month, day, hour, minute, second, weekday, day of the year, daylight saving)
    hour = time.strftime("%H:%M:%S",current_time)#puts the current time in H/M/S format
    str_hour = (str(hour))#makes hour variable a string
    str_hour=str_hour.strip()
    hour,minute,second=str_hour.split(":")#splits the time into single integers of hour,minute and second
    today = datetime.date.today()#gets todays date
    last_day_of_month = calendar.monthrange(today.year,today.month)[1]#gets the exact number of days in the current month
    date = datetime.date(2023, 3, 31)
    if date.day == last_day_of_month and hour >= "20":#if the day is the last of the month and it is 8pm or later count the totals
        print("today is last day")
        rows = app.getGridRowCount("g2")
    for x in range(rows):
        data=app.getGridRow("g2",x)#for each row in the table store values in the variable data
        if data and (data[8]=="January" or data[8]=="February" or data[8]=="March" or data[8]=="April" or data[8]=="May" or data[8]=="June" or data[8]=="July" o
            consultant_id = data[0]#assigns each row index to the corresponding field name
            ward_id = data[1]
            Room_No = data[2]
            Time_Of_Admission = data[3]
            transfer_date = data[4]
            transfer_time = data[5]
            reason_for_transfer = data[6]
            comments = data[7]
            print(reason_for_transfer)


            if "1" in reason_for_transfer:
                MedicalReason_Total+=1 #increment medical reason tally by one

            elif "2" in reason_for_transfer:
                PatientRequest_Total+=1#increment patient request reason tally by one

            elif "3" in reason_for_transfer:
                LateDischarge_Total+=1#increment late discharge reason tally by one

            elif "4" in reason_for_transfer:
                ConsultantRequest_Total+=1#increment consultant request tally by one

            elif "5" in reason_for_transfer:
                SocialReason_Total+=1#increment social reason tally by one
```

This is the date I created to set it to the last day of the month as when I was made this subroutine it was not the end of the month. I also used the **monthrange()** function from the calendar library here to get the exact number days within the month. In this example, using this means the rest of the code will only run if the **date.day** I set (“31”) is equal the **last_day_of_month**(“31”) as this is for march.

Another condition which was checked here is that the time function had to be >=8pm to count the reason for transfers. I did this because this is around the time when the ward closes so no more patient transfers will need to be entered for the final day of the month.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Before adding these totals to be displayed on a bar graph I realised I need to make sure **correct data was going to displayed**, **corresponding to the correct month**. To do this I made it so when data is originally saved in the Daily grid it also saves the **month** column to the database. Therefore, when data is retrieved it only obtains the data for that specific month from the database. Doing this, will make sure there is no redundant data as the months carry on throughout the year.

While trying to add this feature I came across this error where the program was still try to retrieve data from an empty table in my database

```
    grid.addRows(data, scroll=False)
  File "C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming project-DESKTOP-PR2B3J6-DESKTOP-PR2B3J6\appJar\a
    self.numColumns = len(data[0])
IndexError: list index out of range
```

To fix this, looking at **W3schools** I found out there is a function where I could count how many rows in the database table by using the keyword **COUNT.** Implementing this feature, I counted the rows in my tbl_daily. If the number of rows were not equal to zero, only then would the subroutine run.

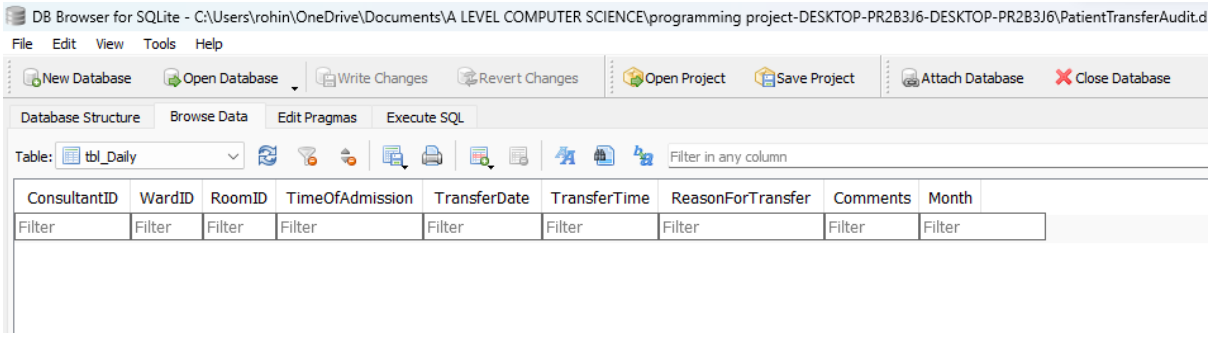


table is empty

```
def retrive_tbl_Daily():
    conn = sqlite3.connect('PatientTransferAudit.db')#connect db
    cur = conn.cursor()
    cur.execute("SELECT COUNT(*) FROM tbl_daily")#counts all the rows in the table
    row_count = cur.fetchone()[0]#stores number of rows in the variable row_count
    if row_count != 0:
        cur.execute("SELECT ConsultantID, WardID, RoomID, TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments FROM tbl_Daily")#
        rows = cur.fetchall()
        headers = ["ConsultantID", "WardID","RoomID", "TimeOfAdmission","TransferDate","TransferTime","ReasonForTransfer","Comments"]#assigns all th
        app.replaceAllGridRows("g1",rows)#replaces all the rows in my grid with the retrived data
        first_row = app.getGridRow("g1",-1)#stores the first row that is currently in the header in the variable first_row
        app.setGridHeaders("g1",headers)#set the grid headers to the headers of the Daily grid
        app.addGridRow("g1",first_row)#adds the first row to the end of the existing grid
    else:
        print("table is empty")
```

```
if row_count != 0:
    cur.execute("SELECT ConsultantID, WardID, RoomID, TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments, Month FROM tb
    rows = cur.fetchall()
    headers = ["ConsultantID", "WardID","RoomID", "TimeOfAdmission","TransferDate","TransferTime","ReasonForTransfer","Comments","Month"]#ass
```

I made sure that now the month field would also be retrieved from tbl_Daily in my database.

To add the **month_name** to all the end of all the rows currently present in the grid I had to make a similar piece of code to my check_monthg2() function called check_monthg1(). This subroutine does the same thing as the current month grid but for the daily grid.

```
def check_monthg1(button):

    if button == "PD_January":
       rows = app.getGridRowCount("g1")#counts the numbers of rows in the grid
       Month_name = "January"
       for i in range(rows):
           print(rows)
           row = app.getGridRow("g1", i)#gets all the current row and stores it in variable row
           if not any(row):#skips the rows that are completely empty
               continue
           if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
                  row[8] = Month_name
                  app.replaceGridRow("g1", i, row)#Add the month_name to each of the rows


    elif button == "PD_February":
           rows = app.getGridRowCount("g1")#counts the numbers of rows in the grid
           Month_name = "February"
           for i in range(rows):
               row = app.getGridRow("g1", i)#gets all the current row and stores it in variable row
               if not any(row):#skips the rows that are completely empty
                   continue
               if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
                      row[8] = Month_name
                      app.replaceGridRow("g1", i, row)#Add the month_name to each of the rows


    elif button == "PD_March":
            rows = app.getGridRowCount("g1")#counts the numbers of rows in the grid
            Month_name = "March"
            for i in range(rows):
               row = app.getGridRow("g1", i)#gets all the current row and stores it in variable row
               if not any(row):#skips the rows that are completely empty
                   continue
               if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
                      row[8] = Month_name
                      app.replaceGridRow("g1", i, row)#Add the month_name to each of the rows



    elif button == "PD_April":
            rows = app.getGridRowCount("g1")#counts the numbers of rows in the grid
            Month_name = "April"
            for i in range(rows):
               row = app.getGridRow("g1", i)#gets all the current row and stores it in variable row
               if not any(row):#skips the rows that are completely empty
                   continue
               if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
                      row[8] = Month_name
                      app.replaceGridRow("g1", i, row)#Add the month_name to each of the rows
```

Patient Transfer Audit.py - C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming project-DESKTOP-PR2B3J6-DESKTOP-PR2B3J6\Patient Transfer Audit.py (3.9.6)

ile Edit Format Run Options Window Help

```
elif button == "PD_May":
        rows = app.getGridRowCount("g1")#counts the numbers of rows in the grid
        Month_name = "May"
        for i in range(rows):
            row = app.getGridRow("g1", i)#gets all the current row and stores it in variable row
            if not any(row):#skips the rows that are completely empty
                continue
            if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
                    row[8] = Month_name
                    app.replaceGridRow("g1", i, row)#Add the month_name to each of the rows



elif button == "PD_June":
        rows = app.getGridRowCount("g1")#counts the numbers of rows in the grid
        Month_name = "June"
        for i in range(rows):
            row = app.getGridRow("g1", i)#gets all the current row and stores it in variable row
            if not any(row):#skips the rows that are completely empty
                continue
            if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
                    row[8] = Month_name
                    app.replaceGridRow("g1", i, row)#Add the month_name to each of the rows


elif button == "PD_July":
        rows = app.getGridRowCount("g1")#counts the numbers of rows in the grid
        Month_name = "July"
        for i in range(rows):
            row = app.getGridRow("g1", i)#gets all the current row and stores it in variable row
            if not any(row):#skips the rows that are completely empty
                continue
            if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
                    row[8] = Month_name
                    app.replaceGridRow("g1", i, row)#Add the month_name to each of the rows


elif button == "PD_August":
        rows = app.getGridRowCount("g1")#counts the numbers of rows in the grid
        Month_name = "August"
        for i in range(rows):
            row = app.getGridRow("g1", i)#gets all the current row and stores it in variable row
            if not any(row):#skips the rows that are completely empty
                continue
            if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
                    row[8] = Month_name
                    app.replaceGridRow("g1", i, row)#Add the month_name to each of the rows
```

Patient Transfer Audit.py - C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming project-DESKTOP-PR2B3J6-DESKTOP-PR2B3J6\Patient Transfer Audit.py (3.9.6)

File Edit Format Run Options Window Help

```
    elif button == "PD_September":
            rows = app.getGridRowCount("g1")#counts the numbers of rows in the grid
            Month_name = "September"
            for i in range(rows):
                row = app.getGridRow("g1", i)#gets all the current row and stores it in variable row
                if not any(row):#skips the rows that are completely empty
                    continue
                if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
                        row[8] = Month_name
                        app.replaceGridRow("g1", i, row)#Add the month_name to each of the rows



    elif button == "PD_October":
            rows = app.getGridRowCount("g1")#counts the numbers of rows in the grid
            Month_name = "October"
            for i in range(rows):
                row = app.getGridRow("g1", i)#gets all the current row and stores it in variable row
                if not any(row):#skips the rows that are completely empty
                    continue
                if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
                        row[8] = Month_name
                        app.replaceGridRow("g1", i, row)#Add the month_name to each of the rows


    elif button == "PD_November":
                rows = app.getGridRowCount("g1")#counts the numbers of rows in the grid
                Month_name = "November"
                for i in range(rows):
                    row = app.getGridRow("g1", i)#gets all the current row and stores it in variable row
                    if not any(row):#skips the rows that are completely empty
                        continue
                    if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
                            row[8] = Month_name
                            app.replaceGridRow("g1", i, row)#Add the month_name to each of the rows


    elif button == "PD_December":
            rows = app.getGridRowCount("g1")#counts the numbers of rows in the grid
            Month_name = "December"
            for i in range(rows):
                row = app.getGridRow("g1", i)#gets all the current row and stores it in variable row
                if not any(row):#skips the rows that are completely empty
                    continue
                if row[8]!= Month_name:#checks if the month name is already present in the last cell of the row
                        row[8] = Month_name
                        app.replaceGridRow("g1", i, row)#Add the month_name to each of the rows
```

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

I noticed, when I first added data into the **empty daily table** there was no **month name** in the Month column so it could not be saved to the database. This meant that when data was retrieved, the corresponding data for that month had not been specified causing **any data** to be added into the eg, "January data only" grid. Looking to the future of this program, adding the **month name** to the daily grid before any data entered would be the most suitable and efficient way of identifying the corresponding month data. In addition, the function will only needed be run for "g1" as it will be transferred as usual to "g2". **Therefore, I realised I would not need the checkmonthg1() and checkmonthg2() as its purpose is essentially the same thing. As a result of this, I removed these two redundant functions from my code.**

To overcome this obstacle, I created another function called addGridRow() where the subroutine would do two things add the **month name** to the end of each row that was currently in the table even if it was empty.

```
def addGridRow(btn):
    rows = app.getGridRowCount("g1")#count number of rows in g1
    for i in range(rows):
        row = [""] * 9 # create a list with empty strings for all columns
        row[8] =  month_in_new_row[8] # set the 9th column (index 8) to the current month
        app.replaceGridRow("g1", i,row)#replace each row[8] present in the grid with Month_name
    if rows == 0:
        app.addGridRow("g1", month_in_new_row)#add current month to row[8] if theres no rows at all
    else:
        for i in range(rows):
            row = app.getGridRow("g1", i)#gets each individual row
            if not any(row):#checks if each field row is empty
                continue#if the row is not empty then continue
            if row[8] != month_in_new_row[8]:#checks if row[8] already contains the current month
                row[8] = month_in_new_row[8]#if not assign row[8] to the current month
                app.replaceGridRow("g1", i, row)#Replaces the row with the updated changes


        else:
            app.addGridRow("g1", month_in_new_row)#if no rows have an empty row[8] add new row to grid with the corresponding month
```

Secondly, when the user would add a new row the correct month name would be displayed in the **month column** ready to be saved to the database. Each row is checked if it is empty and as now the grid should have some data in it due the code in the **red box**, the program would continue and assign row[8] to the correct month if it is not already present. Finally, the row data with the **corresponding month** will replace the **current row data**, updating the row. This enhances my programs usability and effectiveness.

Both pieces of code use the global variable month_in_new_row from another subroutine called set_month_to_add_row(). This global variable contains the corresponding month that should be inputted into the grid. The assignment of this variable occurs when the corresponding month button is pressed such as "PD_January" assigns "January" to the variable.

```
def set_month_to_add_row(button):
    global month_in_new_row#makes global variable for each month from press(button)
    if button == "PD_January":
        month_in_new_row = [""] * 9#create a list with empty strings for all columns
        month_in_new_row[8] = "January"#sets index[8] column to "January"
        btn = True#set btn = true so addGridRow() can now take place
        addGridRow(btn)
```

Once the month is assigned to the global variable I set the "btn" to add a new row to True so the addGridRow() function could occur and pass in the appropriate month. I did it this way as there are two buttons involved in the presentation of the month in the grid,(the button to choose the month and the button to add a new row). Therefore, I created these two separate functions to allow for both buttons to work together.

Now this was created, I had to make sure that when it is a certain month the correct data gets added to the correct grid without the data being added to the others months grid.

However, in the development of this I came across a problem where the data which was being fetched from the database was being overridden by the **red box** in addGridRow(). This is because the whole row was being **replaced** and displaying **ONLY** the month in row[8] due to the fact I set `row = [""] * 9` which made all the rows blank.

The solution to this was to make sure only row[8] was being replaced and the rest of the data in the row was retained. I did this by writing adjusting the **red box** in addGridRow() and removing that line of code just mentioned.

```
def addGridRow(btn):

    rows = app.getGridRowCount("g1")
    for i in range(rows):
        row = app.getGridRow("g1", i)
        row[8] = month_in_new_row[8] # set the 9th column (index 8) to the value of month_in_new_row
        app.replaceGridRow("g1", i, row)#replace original row with updated version
```

For the program to run smoothly, I need to synchronise the date with the ability to obtain the correct information from the tables in my database. To do this, I edited my retrive_tbl_Daily() function so that when the date is a certain month only the corresponding data will be fetched from the database.

```
def retrive_tbl_Daily():
    conn = sqlite3.connect('PatientTransferAudit.db')#connect db
    cur = conn.cursor()
    cur.execute("SELECT COUNT(*) FROM tbl_daily")#counts all the rows in the table
    row_count = cur.fetchone()[0]#stores number of rows in the variable row_count
    if row_count != 0:
        today = datetime.date.today()#gets todays date
        print(today)
        if today.month == 1:
            cur.execute("SELECT ConsultantID, WardID, RoomID, TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments, Month FROM tbl_Daily WHERE Month='January'")#selects
        elif today.month == 2:
            cur.execute("SELECT ConsultantID, WardID, RoomID, TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments, Month FROM tbl_Daily WHERE Month='February'")#selects
        elif today.month == 3:
            cur.execute("SELECT ConsultantID, WardID, RoomID, TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments, Month FROM tbl_Daily WHERE Month='March'")#selects al
        elif today.month == 4:
            cur.execute("SELECT ConsultantID, WardID, RoomID, TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments, Month FROM tbl_Daily WHERE Month='April'")#selects al
        elif today.month == 5:
            cur.execute("SELECT ConsultantID, WardID, RoomID, TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments, Month FROM tbl_Daily WHERE Month='May'")#selects all
        elif today.month == 6:
            cur.execute("SELECT ConsultantID, WardID, RoomID, TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments, Month FROM tbl_Daily WHERE Month='June'")#selects all
        elif today.month == 7:
            cur.execute("SELECT ConsultantID, WardID, RoomID, TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments, Month FROM tbl_Daily WHERE Month='July'")#selects all
        elif today.month == 8:
            cur.execute("SELECT ConsultantID, WardID, RoomID, TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments, Month FROM tbl_Daily WHERE Month='August'")#selects a
        elif today.month == 9:
            cur.execute("SELECT ConsultantID, WardID, RoomID, TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments, Month FROM tbl_Daily WHERE Month='September'")#select
        elif today.month == 10:
            cur.execute("SELECT ConsultantID, WardID, RoomID, TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments, Month FROM tbl_Daily WHERE Month='October'")#selects
        elif today.month == 11:
            cur.execute("SELECT ConsultantID, WardID, RoomID, TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments, Month FROM tbl_Daily WHERE Month='November'")#selects
        elif today.month == 12:
            cur.execute("SELECT ConsultantID, WardID, RoomID, TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments, Month FROM tbl_Daily WHERE Month='December'")#selects
```

```
    rows = cur.fetchall()
    headers = ["ConsultantID", "WardID","RoomID", "TimeOfAdmission","TransferDate","TransferTime","ReasonForTransfer","Comments","Month"]#as
    app.replaceAllGridRows("g1",rows)#replaces all the rows in my grid with the retrived data
    first_row = app.getGridRow("g1",-1)#stores the first row that is currently in the header in the variable first_row
    app.setGridHeaders("g1",headers)#set the grid headers to the headers of the Daily grid
    app.addGridRow("g1",first_row)#adds the first row to the end of the existing grid
    print(rows)
```

Here, I again made use of the time library where for example, if the month was equal to 1 only the data with the “January” month in my daily table will be taken from the database. Then the function will carry out as usual adding the data to the January daily grid.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

For all the months to work when transferring the data from the **daily grid** to **the current month grid**, I had to make sure when it was a **specific** month in the year only the data for that **specific** month could be transferred.

```
OP-PR2B3J6-DESKTOP-PR2B3J6\Patient Transfer Audit.py", line 696, in month_check
    row = app.getGridRow("g1", i)
  File "C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming project-DESK
OP-PR2B3J6-DESKTOP-PR2B3J6\appJar\appjar.py", line 5239, in getGridRow
    return self.getTableRow(title, rowNumber)
  File "C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming project-DESK
OP-PR2B3J6-DESKTOP-PR2B3J6\appJar\appjar.py", line 5173, in getTableRow
    return grid.getRow(rowNumber)
  File "C:\Users\rohin\OneDrive\Documents\A LEVEL COMPUTER SCIENCE\programming project-DESK
OP-PR2B3J6-DESKTOP-PR2B3J6\appJar\appjar.py", line 14390, in getRow
    for cell in self.cells[rowNumber+1]:
IndexError: list index out of range
```

Daily

ConsultantID	WardID	RoomID
JJ	5	520
IGD	1	114

When creating this function, this is the error I got when I pressed on a month that was not the current month and then went back to the current month. The data was added but there is an empty row between the data.

When I tried fixing this by running the retrive_tbl_daily again one the month Is clicked, I got another error of “cellNum” _addRow widg.grid(row=rowNum, column=cellNum + 1, sticky=N+E+S+W) UnboundLocalError: local variable 'cellNum' referenced before assignment which was to do with one of the functions in the appJar Library. After multiple attempts, of trying to fix this I could not find a suitable solution, therefore there is a limitation to my project, as the user can only select the month which is the current month in the calendar year. For example, if it is currently March only the march button can be clicked to edit/add patient transfers.

```
def month_check(button):
    rowsg1 = app.getGridRowCount("g1")
    today = datetime.date.today() #gets today's date
    print(today)
    if (button == "PD_January" or button == "PD_February" or button == "PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June" or
            for i in range(rowsg1):
                row = app.getGridRow("g1", i)#gets each row in the grid
                print(row)
                if all(val == "" for val in row):
                   continue # skip empty rows
                if row[8] != "January":
                    empty_row = [""] * 9#creates an empty list of 9 values
                    app.replaceGridRow("g1", i, empty_row#adds the empty rows to the grid if the row[8]!=January
                else:
                    return TransferDailyTable(button)
```

Here, is the final product of the function and I have named it month_check(). This checks the month at the current time of the year and in this case, it is “January” so

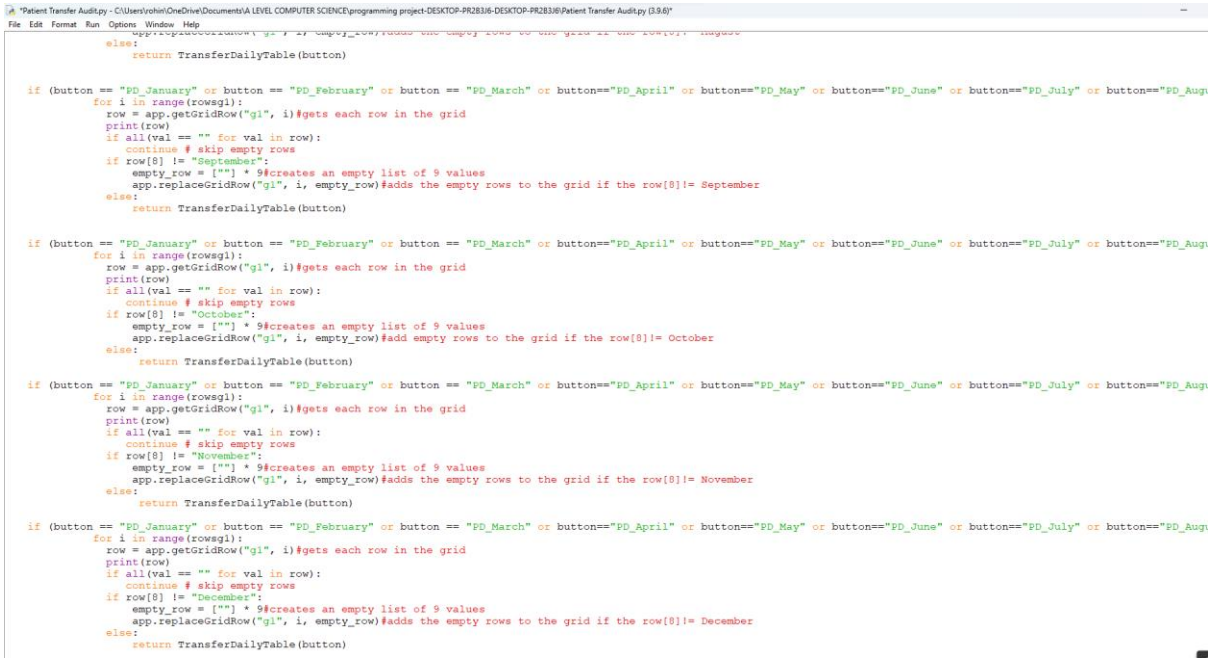
```
and today.month ==1:#checks if the current month is january
```

. The if statement iterates over the rows in the grid and if any of the row[8]!= “January” it is replaced with empty rows. This makes it easier when transferring data as the data has already been associated with the appropriate month. Finally after the rows have finished iterating the TransferDailyTable() function is run.

The rest of the code of the month_check() function:

```
if (button == "PD_January" or button == "PD_February" or button == "PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June" or button=="PD_July" or button=="PD_August" 
          for i in range(len(rowsg1)):
            row = app.getGridRow("g1", i)#gets each row in the grid
            print(row)
            if all(val == "" for val in row):
               continue # skip empty rows
            if row[8] != "February":
                empty_row = [""] * 9#creates an empty list of 9 values
                app.replaceGridRow("g1", i, empty_row)#adds the empty rows to the grid if the row[8]!=February
            else:
                return TransferDailyTable(button)

if (button == "PD_January" or button == "PD_February" or button == "PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June" or button=="PD_July" or button=="PD_August" 
        print("printing",rowsg1)
        for i in range(rowsg1):
          row = app.getGridRow("g1", i)#gets each row in the grid
          print(row)
          if all(val == "" for val in row):
            continue # skip empty rows
          if row[8] != "March":
              print("creating blank rows")
              empty_row = [""] * 9#creates an empty list of 9 values
              app.replaceGridRow("g1", i, empty_row)#adds the empty rows to the grid if the row[8]!=March
          else:

              return TransferDailyTable(button)


if (button == "PD_January" or button == "PD_February" or button == "PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June" or button=="PD_July" or button=="PD_August" 
          for i in range(rowsg1):
            row = app.getGridRow("g1", i)#gets each row in the grid
            print(row)
            if all(val == "" for val in row):
               continue # skip empty rows
            if row[8] != "April":
                empty_row = [""] * 9#creates an empty list of 9 values
```

```
            continue # skip empty rows
        if row[8] != "April":
            empty_row = [""] * 9#creates an empty list of 9 values
            app.replaceGridRow("g1", i, empty_row)#adds the empty rows to the grid if the row[8]!=April
        else:
            return TransferDailyTable(button)


if (button == "PD_January" or button == "PD_February" or button == "PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June" or button=="PD_July" or button=="PD_Augu
      for i in range(rowsg1):
        row = app.getGridRow("g1", i)#gets each row in the grid
        print(row)
        if all(val == "" for val in row):
            continue # skip empty rows
        if row[8] != "May":
            empty_row = [""] * 9#creates an empty list of 9 values
            app.replaceGridRow("g1", i, empty_row)#adds the empty rows to the grid if the row[8]!=May
        else:
            return TransferDailyTable(button)


if (button == "PD_January" or button == "PD_February" or button == "PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June" or button=="PD_July" or button=="PD_Augu
      for i in range(rowsg1):
        row = app.getGridRow("g1", i)#gets each row in the grid
        print(row)
        if all(val == "" for val in row):
            continue # skip empty rows
        if row[8] != "June":
            empty_row = [""] * 9#creates an empty list of 9 values
            app.replaceGridRow("g1", i, empty_row)#adds the empty rows to the grid if the row[8]!=June
        else:
            return TransferDailyTable(button)


if (button == "PD_January" or button == "PD_February" or button == "PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June" or button=="PD_July" or button=="PD_Augu
      for i in range(rowsg1):
        row = app.getGridRow("g1", i)#gets each row in the grid
        print(row)
        if all(val == "" for val in row):
            continue # skip empty rows
        if row[8] != "July":
            empty_row = [""] * 9#creates an empty list of 9 values
            app.replaceGridRow("g1", i, empty_row)#adds the empty rows to the grid if the row[8]!=July
        else:
            return TransferDailyTable(button)

if (button == "PD_January" or button == "PD_February" or button == "PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June" or button=="PD_July" or button=="PD_Augu
```



As this was completed I could now edit my TransferDailyTable() function so it only transfers acceptable month data.

```
def TransferDailyTable(button):
    seconds = time.time()#gets the current time in seconds
    local_time = time.ctime(seconds)#converts the time in seconds into a readable format
    print("Local time:", local_time)#prints the readable time
    current_time = time.localtime()#gets the struct_time function in format(year, month, day, hour, minute, second, weekday, day of the year, daylight saving)
    today = datetime.date.today() #gets today's date
    print(today)
    hour = time.strftime("%H:%M:%S",current_time)#puts the current time in H/M/S format
    str_hour = (str(hour))#makes hour variable a string
    str_hour=str_hour.strip()
    hour,minute,second=str_hour.split(":")#splits the time into single integers of hour,minute and second
    if hour >="19" and hour <= "20":#if hour is between 7pm and 8pm run the cuntion
        print("hey")
        rows = app.getGridRowCount("g1")#gets number of rows with data
        for i in range(rows):
            row = app.getGridRow("g1", i)
            print(row)
            print(rows)
            rowData=[]#empty list to transfer data
            for x in range(rows):
                data=app.getGridRow("g1",x)#for each row in the table store values in the variable data
                rowData.append([data])#2D list
            print("row data",rowData)


        if row[8] == "January" or row[8] == "Febraury" or row[8] == "March" or row[8] == "April" or row[8] == "May" or row[8] == "June" or row[8] == "July" or row[8] == "August" or row[
            for aList in rowData:#for each list in rowData
                app.addGridRow("g2",aList[0])#adds each list in the 2D list to current month

        print(rows)
        replace_g1 = app.getGridRowCount("g1")#count number of rows in g1
        for i in range(replace_g1):
            app.replaceGridRow("g1",i,"")#for each row replace g1 with empty values
```

This is the updated TransferDailyTable() function that adds the data from the daily grid into the current month grid everyday when the code is run anytime between 7pm and 8pm. I altered the time from 8pm to between 8pm and 9pm as I want to allow enough time for the AuditGraph() function to be checked, therefore allowing time slots for these functions will increase my programs efficiency when dealing with lots of data moving between grids at different times(prevents them running at the same time). The function then gets all the grid rows in the daily grid and adds it to a new 2D list called rowData. If the row index 8 from the **current** selected row is equal to **any month, each list within the list** will be added to bottom of the current month grid. Finally, all the rows which were in the daily grid are replaced with empty rows, therefore refreshing the daily grid.

I then next had to edit my subroutine for counting the totals of the reasons for transfer so it would only count them for the current month during the calendar year.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

MonthlyGraph() with Current Month Values:

```
def MonthlyGraph():
    current_time = time.localtime()#gets the struct_time function in format(year, month, day, hour, minute, second, weekday, day of the year, daylight sav
    hour = time.strftime("%H:%M:%S",current_time)#puts the current time in H/M/S format
    str_hour = (str(hour))#makes hour variable a string
    str_hour=str_hour.strip()
    hour,minute,second=str_hour.split(":")#splits the time into single integers of hour,minute and second
    today = datetime.date.today()#gets todays date
    last_day_of_month = calendar.monthrange(today.year,today.month)[1]#gets the exact number of days in the current month
    date = datetime.date(2023, 3, 31)
    if date.day == last_day_of_month and (hour >= "20" and hour <= "21"):#if the day is the last of the month and it is 8pm or later count the totals

        x = np.array(["Medical Reason", "Patient request", "Late Discharge","Consultant Request","Social Reason","Other"])#x-axis
        y = np.array([MedicalReason_Total, PatientRequest_Total, LateDischarge_Total,ConsultantRequest_Total,SocialReason_Total,Other_Total])#y-axis

        plt.figure(figsize=(10,6))#size of bar graph
        plt.bar(x,y)#plot bar graph
        plt.show()#show bar graph
Other_Total = 0#total for other/unknown reasons
def AuditGraph():
    global MedicalReason_Total#refrence the global variables
    global PatientRequest_Total
    global LateDischarge_Total
    global ConsultantRequest_Total
    global SocialReason_Total
    global Other_Total
    seconds = time.time()#gets the current time in seconds
    local_time = time.ctime(seconds)#converts the time in seconds into a readable format
    print("Local time:", local_time)#prints the readable time
    current_time = time.localtime()#gets the struct_time function in format(year, month, day, hour, minute, second, weekday, day of the year, daylight saving)
    hour = time.strftime("%H:%M:%S",current_time)#puts the current time in H/M/S format
    str_hour = (str(hour))#makes hour variable a string
    str_hour=str_hour.strip()
    hour,minute,second=str_hour.split(":")#splits the time into single integers of hour,minute and second
    today = datetime.date.today()#gets todays date
    last_day_of_month = calendar.monthrange(today.year,today.month)[1]#gets the exact number of days in the current month
    print(last_day_of_month)
    date = datetime.date(2023, 3, 31)
    if date.day == last_day_of_month and today.month == 1 and (hour >= "20" and hour <= "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the cur
            month = "January"
    elif date.day == last_day_of_month and today.month == 2 and (hour >= "20" and hour <= "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the c
            month = "February"
    elif date.day == last_day_of_month and today.month == 3 and (hour >= "20" and hour <= "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the c
            month = "March"
    elif date.day == last_day_of_month and today.month == 4 and (hour >= "20" and hour <= "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the c
            month = "April"
    elif date.day == last_day_of_month and today.month == 5 and (hour >= "20" and hour <= "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the c
            month = "May"
    elif date.day == last_day_of_month and today.month == 6 and (hour >= "20" and hour <= "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the c
            month = "June"
    elif date.day == last_day_of_month and today.month == 7 and (hour >= "20" and hour <= "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the c
            month = "July"
    elif date.day == last_day_of_month and today.month == 8 and (hour >= "20" and hour <= "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the c
            month = "August"
    elif date.day == last_day_of_month and today.month == 9 and (hour >= "20" and hour <= "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the c
            month = "September"
    elif date.day == last_day_of_month and today.month == 10 and (hour >= "20" and hour <= "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the
            month = "October"
    elif date.day == last_day_of_month and today.month == 11 and (hour >= "20" and hour <= "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the
            month = "November"
    elif date.day == last_day_of_month and today.month == 12 and (hour >= "20" and hour <= "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the
            month = "December"
```

Above is the first part of the new and improved AuditGraph() function. Again, all the totals for each reason of transfer are set to “0 “and are global variable as it needs to be used in another function. I get the current time and split it by (hour:minute:second) using the strip command. I also got the current date and last day of the month using the import time and import calendar library which I used **w3resource** to research for. In this case, the current date is **march 31st**.Therefore the variable month is set to **march**. All these if statements only occur if the time is between 8pm and 9 pm as this is when data will no longer be entered into any of the grids.

```
print("today is last day")
rows_g2 = app.getGridRowCount("g2")
print(rows_g2)
for x in range(rows_g2):
    data=app.getGridRow("g2",x)#for each row in the table store values in the variable data
    print(data)
    if data and (data[8]==month):
        consultant_id = data[0]#assigns each row index to the corresponding field name
        ward_id = data[1]
        Room_No = data[2]
        Time_Of_Admission = data[3]
        transfer_date = data[4]
        transfer_time = data[5]
        reason_for_transfer = data[6]
        comments = data[7]
        print(reason_for_transfer)


        if "1" in reason_for_transfer:
            MedicalReason_Total+=1 #increment medical reason tally by one

        elif "2" in reason_for_transfer:
            PatientRequest_Total+=1#increment patient request reason tally by one

        elif "3" in reason_for_transfer:
            LateDischarge_Total+=1#increment late discharge reason tally by one

        elif "4" in reason_for_transfer:
            ConsultantRequest_Total+=1#increment consultant request tally by one

        elif "5" in reason_for_transfer:
            SocialReason_Total+=1#increment social reason tally by one


        elif "6" in reason_for_transfer:
            Other_Total+=1#increment other reason tally by one


print("total is " , SocialReason_Total)
```

This second part of the AuditGraph() function is similar to the previous version of this function. Once the correct corresponding month has been selected the data is taken from each row in the current month grid. If the data[8] is equal to the appropriate month, each column should assigned an index that matches the grid. If the grid contains any of the reasons, each reason in each row would be added to their pre-defined totals. The code iterates through all the rows until it has reached, the end of the grid. Furthermore, I did the code this way so when the totals are inserted into the bar graph using the matplotlib library, only the **current months data** will be added to that graph.

This where the current month values are displayed in a bar graph using the matplotlib library as previously mentioned. I again used the import time and import calendar library here to get the last day of the month for the current month and a specific hour. I got the hour on its own by using the slip command again and separating the time between each (:). At the same time when the totals for the reasons for transfer are calculated, the bar graph is created with the corresponding data. However, the only difference is that this occurs when the respected “(month) graph” button is pressed for example,

```
elif button=="January":
    app.hideSubWindow("AuditGraphs")
    MonthlyGraph()
    app.showSubWindow("AuditGraphs")   # switch back to previous subwindow
```

There is already a **built in function** to save the bar graph to **files** in the matplotlib library if the nurses would like a copy of the graph but once the user has finished looking at the grid, after they exit, the bar graph page closes and the previous sub window is shown.

As I made a personalised addGridRow button I had to make another one for g2. I renamed the g1 add row function addGridRowg1() and called the g2 one addGridRowg2().

```
def addGridRowg2(btn):

    rowsg2 = app.getGridRowCount("g2")#count number of rows in g2
    row = app.getGridRow("g2", 0)#gets the first row in g2
    if row[8] != month_in_new_row[8] or row[8] == month_in_new_row[8]:#checks if row[8] already contains the current month or if it does not
            row[8] = month_in_new_row[8]#if not assign row[8] to value in month_in_new_row
            app.addGridRow("g2", month_in_new_row)#add grid row containing displaying the value in month_in_new_row in row[8]
```

This subroutine can get any row as the grid will either contain nothing or contain data that already has the correct month in row[8]. If row[8] contains the correct month or doesn't contain the correct month, its purpose would be the same. This is to add an empty new grid row to the end of the grid containing the appropriate month name.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

One of the last functions I needed to add in prototype 2 is the extra page in my program that gives the summary of results from the bar graph.

I noticed while trying to test data for this new subroutine that when data was transferred, if

```
if hour >="19" and hour <="20":#if hour is between 7pm and 8pm run the funtion
```

the function still runs when the hour continues into 8pm. As I did not want the function to run through this hour, I altered the if statement

```
if hour >="19" and hour <"20":#if hour is between 7pm and 8pm run the funtion
```

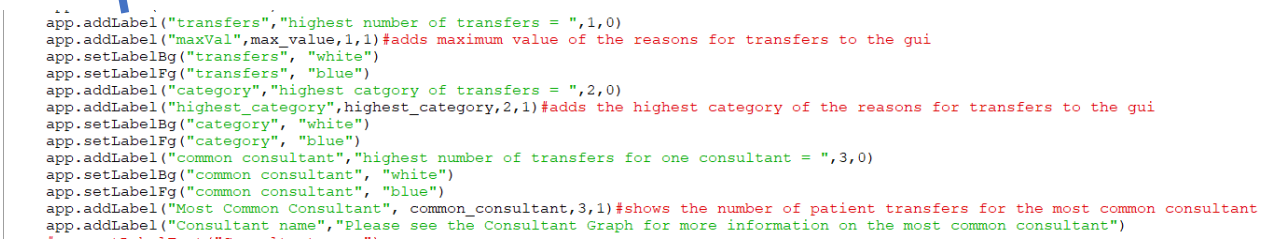
so that the function only runs within the hour of 7pm. I also repeated this for all my other functions which relied on the import time library.

The following code is for the graph summary page which was an addition to the main program.

```
max_value = 0#sets max_value to 0 so it can be used as a global variable
highest_category = 0#sets max_value to 0 so it can be used as a global variable
common_consultant = 0#sets max_value to 0 so it can be used as a global variable
def GraphSummary():
    global max_value#makes variables global
    global highest_category
    global common_consultant
    reasons_for_transfer = {"MedicalReason": MedicalReason_Total,
                            "PatientRequest": PatientRequest_Total,
                            "LateDischarge": LateDischarge_Total,
                            "ConsultantRequest": ConsultantRequest_Total,
                            "SocialReason": SocialReason_Total,
                            "Other": Other_Total}#dictionary is made to store the variables with its respected values

    highest_reason_for_transfer = max(reasons_for_transfer, key=reasons_for_transfer.get)#key which gets the name of the reason for transfer with the highest value
    highest_number = reasons_for_transfer[highest_reason_for_transfer]#gets the number where the reason for transfer contains the highest value

    max_value = highest_number#stores the highest_number in variable max_value
    highest_category = highest_reason_for_transfer#stores the highest_reason_for_transfer in highest_category
    common_consultant = max(new_reasonsList)
    print(max_value)
    print(highest_category)
    app.setLabel("maxVal", max_value)# updates max_value label
    app.setLabel("highest_category",highest_category)#updates highest category label
    app.setLabel("Most Common Consultant",common_consultant)#updates the Most Common Consultant label
```

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

This is the function which calculates the trends of the values worked out for each reason for transfer in AuditGraph(). As its most suitable, I used dictionaries here because I have values which need to assigned to variables so they can be displayed on the Graph summary page. With some help using **w3schools** I quickly realised that a key is needed to get the name of the variable assigned with the value, using **.get.** With knowledge I used the max function to find the **greatest** value in my dictionary and stored it in the variable highest_reason_for_transfer. Therefore, to obtain the highest category from the reasons I used this **highest_reason_for_transfer** to get back the name of the reason of transfer. The common consultant variable works out the consultant who has the highest number of transfers. All these values **updated** labels that were later added to the Graph Summary page. However, I realised I would need a **separate bar graph** for the individual consultant totals because how my program is built using appJar, it would be too long to allow the graph to hold 3 variables. Consequently, given the lack of time I went with the simpler option of creating a new bar graph.



```
app.addLabel("transfers","highest number of transfers = ",1,0)
app.addLabel("maxVal",max_value,1,1)#adds maximum value of the reasons for transfers to the gui
app.setLabelBg("transfers", "white")
app.setLabelFg("transfers", "blue")
app.addLabel("category","highest catgory of transfers = ",2,0)
app.addLabel("highest_category",highest_category,2,1)#adds the highest category of the reasons for transfers to the gui
app.setLabelBg("category", "white")
app.setLabelFg("category", "blue")
app.addLabel("common consultant","highest number of transfers for one consultant = ",3,0)
app.setLabelBg("common consultant", "white")
app.setLabelFg("common consultant", "blue")
app.addLabel("Most Common Consultant", common_consultant,3,1)#shows the number of patient transfers for the most common consultant
app.addLabel("Consultant name","Please see the Consultant Graph for more information on the most common consultant")
```

For the new bar graph I named the function to calculate the number of patient transfers for each consultant in the grid, ConsultantGraph(). This is shown below.

```
#if its end of january get reason for trasnfer for where the month column = january in my current month grid
new_consultantList = 0
new_reasonsList = 0
def ConsultantGraph():
    seconds = time.time()#gets the current time in seconds
    local_time = time.ctime(seconds)#converts the time in seconds into a readable format
    print("Local time:", local_time)#prints the readable time
    current_time = time.localtime()#gets the struct_time function in format(year, month, day, hour, minute, second, weekday, day of the year, daylight saving)
    hour = time.strftime("%H:%M:%S",current_time)#puts the current time in H/M/S format
    str_hour = (str(hour))#makes hour variable a string
    str_hour=str_hour.strip()
    hour,minute,second=str_hour.split(":")#splits the time into single integers of hour,minute and second
    today = datetime.date.today()#gets todays date
    last_day_of_month = calendar.monthrange(today.year,today.month)[1]#gets the exact number of days in the current month
    print(last_day_of_month)
    date = datetime.date(2023, 3, 31)
    global new_consultantList#refrences global variables to be used outisde the function
    global new_reasonsList#refrences global variables to be used outisde the function
    if  date.day == last_day_of_month and today.month == 1 and hour >= "20": and hour <"21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the cu
        month = "January"
    elif date.day == last_day_of_month and today.month == 2 and hour >= "20": and hour < "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the
        month = "February"
    elif date.day == last_day_of_month and today.month == 3 and hour >= "20": and hour < "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the
        print("hello")
        month = "March"
    elif date.day == last_day_of_month and today.month == 4 and hour >= "20": and hour < "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the
        month = "April"
    elif date.day == last_day_of_month and today.month == 5 and hour >= "20": and hour < "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the
        month = "May"
    elif date.day == last_day_of_month and today.month == 6 and hour >= "20": and hour < "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the
        month = "June"
    elif date.day == last_day_of_month and today.month == 7 and hour >= "20": and hour < "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the
        month = "July"
    elif date.day == last_day_of_month and today.month == 8 and hour >= "20": and hour < "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the
        month = "August"
    elif date.day == last_day_of_month and today.month == 9 and hour >= "20": and hour < "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the
        month = "September"
    elif date.day == last_day_of_month and today.month == 10 and hour >= "20": and hour < "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the
        month = "October"
    elif date.day == last_day_of_month and today.month == 11 and hour >= "20": and hour < "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the
        month = "November"
    elif date.day == last_day_of_month and today.month == 12 and hour >= "20": and hour < "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the
        month = "December"

    print("today is last day")
    rows_g2 = app.getGridRowCount("g2")
    print(rows_g2)
    for i in range(rows_g2):
        row=app.getGridRow("g2",i)#for each row in the table store values in the variable data
```

```
print("today is last day")
rows_g2 = app.getGridRowCount("g2")
print(rows_g2)
for i in range(rows_g2):
    row=app.getGridRow("g2",i)#for each row in the table store values in the variable data
    if row[0] == "":
        continue  #skip this row if the consultant ID is empty
    consultants = []#makes an empty list for consultants
    reasons = []
    if row and (row[8]==month):#if row[8] is equal to the corresponding month
        for x in range(rows_g2):
            data = app.getGridRow("g2",x)#iterate through the grid and get each row
            consultant_id = data[0]#assigns the first row index to the consultant ID
            reason_for_transfer = data[6]
            consultants.append([consultant_id])#adds data to the new list
            reasons.append([reason_for_transfer])#adds reasons to new list
print("consultants are", consultants)
print("reasons are", reasons)
new_consultantList = [x[0] for x in consultants if x != ['']]#remove any empty values from the created list
print(new_consultantList)

new_reasonsList = [x[0] for x in reasons if x != ['']]#remove any empty values from the created list
print(new_reasonsList)
```

This piece of code works similarly to the TransferDailyTable(). Although here the current month is calculated based on if the time is between 8pm and 9pm using the import time library as well as if it is the last day of the month using the import calendar library. I have also made two global variables to store the results of the calculations. The subroutine creates two empty lists consultants and reasons. If index 8 is equal to the current month the grid iterates through each row and adds anything that is in the consultant ID to the consultants list. In addition, anything in the reason for transfer column is added to the reasons list. As these lists can contain **blank values** like this

`[[''], [''], [''], [''], ['JJ'], ['AM'], ['HH'], ['L'], ['IGD']]`

so I created a statement which checks the 2D list and forms a 1D list with only the data, removing any empty indexes.

new_consultantList = `['JJ', 'AM', 'HH', 'L', 'IGD']`

new_reasonsList = `['3', '3', '3', '6', '6']`

As the variables are global I used them to plot the consultant graph.

```
def PlotConsultantGraph():
    current_time = time.localtime()#gets the struct_time function in format(year, month, day, hour, minute, second, weekday, day of the year, daylight saving)
    hour = time.strftime("%H:%M:%S",current_time)#puts the current time in H/M/S format
    str_hour = (str(hour))#makes hour variable a string
    str_hour=str_hour.strip()
    hour,minute,second=str_hour.split(":")#splits the time into single integers of hour,minute and second
    today = datetime.date.today()#gets todays date
    last_day_of_month = calendar.monthrange(today.year,today.month)[1]#gets the exact number of days in the current month
    date = datetime.date(2023, 3, 31)
    if date.day == last_day_of_month and hour >= "20" and hour < "21":#if the day is the last of the month and it is 8pm or later count the totals

        #for x in range(new_consultantList):
        x = np.array(new_consultantList)#x-axis
        y = np.array(new_reasonsList)#y-axis

        plt.figure(figsize=(10,6))#size of bar graph
        plt.bar(x,y)#plot bar graph
        plt.show()#show bar graph
```

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

This is the function for plotting the data in a consultant graph. Here again the import time and calendar library are used to get the current time and last day of the month. By using the split function, I obtained the hour value of the time. If hour is between 8pm and 9pm store the global variables as **x** and **y** then plot the values of these variables on a x/y bar graph. Going back to the AuditGraph() subroutine the common consultant is calculated using the max function in python on the **new_reasonsList** and then it is displayed in the Graph Summary page.

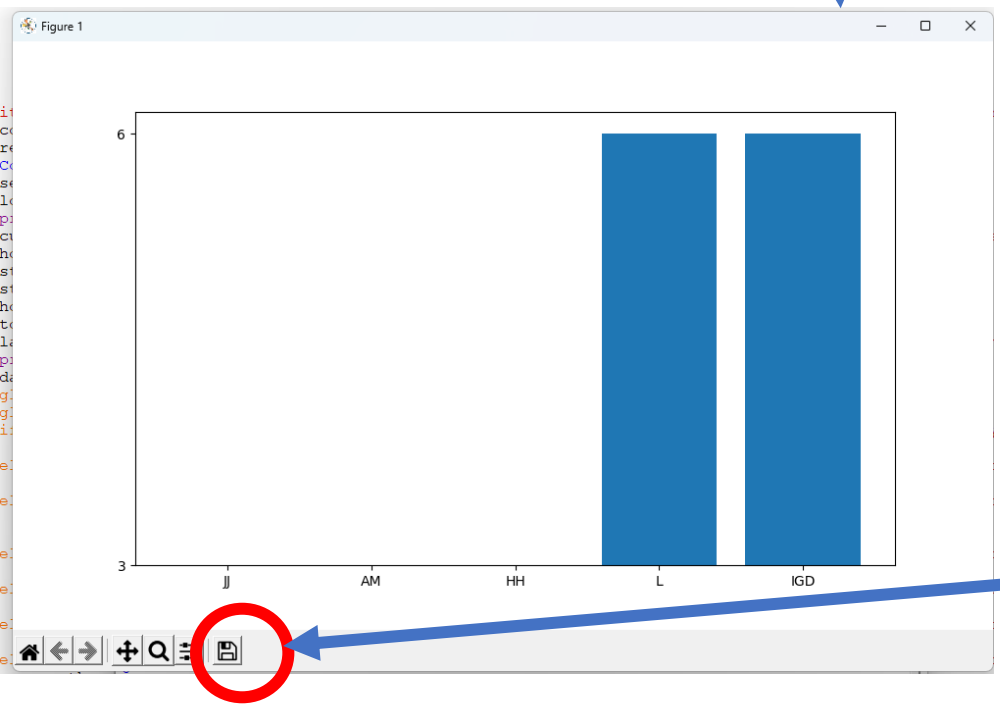
```
####Audit Graphs####
    app.startSubWindow("AuditGraphs",0,1)
    app.setSticky("n")
    app.setPadding([40,40])
    app.addLabel("title14","DSW Transfer Audit",0,0,2,1)
    app.setLabelBg("title14", "white")
    app.setLabelFg("title14", "blue")
    app.setSize("fullscreen")
    app.setFont(18)
    app.setSticky("e")
    app.setPadding([40,40])
    app.addNamedButton("View Graph Summary","VGS",press,1,1)#adds the View Graph Summary button for any month
    app.addNamedButton("Consultant Graph","CG",press,1,2)#adds the consultant graph button for any month
    app.setButtonBg("CG","lightblue")
    app.setButtonBg("VGS","lightblue")
    app.setBg("white")
    app.setSticky("sw")
    app.setPadding([40,40])
    app.addNamedButton("Back","BackAGP",press)
    app.setButtonBg("BackAGP","lightblue")
```

x = ['JJ', 'AM', 'HH', 'L', 'IGD']

y = ['3', '3', '3', '6', '6']

Consultant graph displays march's data

Adds the "consultant graph" button to all the months of the Audit Graph subwindow. Therefore, when it is clicked it creates a graph for the corresponding month of the calendar year.



The user has an option of saving a copy of the bar graph to their files

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

```
def MonthlyGraph():
    current_time = time.localtime()#gets the struct_time function in format(year, month, day, hour, minute, second, weekday, day of the year, daylight saving)
    hour = time.strftime("%H:%M:%S",current_time)#puts the current time in H/M/S format
    str_hour = (str(hour))#makes hour variable a string
    str_hour=str_hour.strip()
    hour,minute,second=str_hour.split(":")#splits the time into single integers of hour,minute and second
    today = datetime.date.today()#gets todays date
    last_day_of_month = calendar.monthrange(today.year,today.month)[1]#gets the exact number of days in the current month
    date = datetime.date(2023, 3, 31)
    if date.day == last_day_of_month and (hour >= "20" and hour < "21"):#if the day is the last of the month and it is 8pm or later count the totals

        x = np.array(["Medical Reason", "Patient request", "Late Discharge","Consultant Request","Social Reason","Other"])#x-axis
        y = np.array([MedicalReason_Total, PatientRequest_Total, LateDischarge_Total,ConsultantRequest_Total,SocialReason_Total,Other_Total])#y-axis

        plt.figure(figsize=(10,6))#size of bar graph
        plt.bar(x,y)#plot bar graph
        plt.show()#show bar graph
```

This is the main graph in my software that takes the totals of the different reasons of transfer for the current month and plots them into a bar chart for the nurses to view. When the time is used from the import time library and .monthrange is used from the import calendar library, again the graph is only plotted if time is between 8pm and 9pm and it is the last day of the month. (x-axis = different reasons of transfer, y axis = number of transfers)

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Graph Summary

highest number of transfers = 3

highest catgory of transfers = LateDischarge

highest number of transfers for one consultant = 6

Please see the Consultant Graph for more information on the most common consultant

Back

Prototype 2 finished

To add/improve for prototype 3:

1) For prototype 3, the user should be able to make changes in the current month database and that should be able to be added to the current month table in my database. I will also delete any consultants that have now left or no longer practice at The London clinic.

2) I am going to add validation some of the key grid columns so when a record is made it is in one format that can be recognised consistently throughout the whole and this also prevents mis-inputted/incorrect information being saved under a patient transfer

3) Finally, following my clients request I am going make it easier to login for the users by allowing the user to press the enter button on the keyboard to go into the program saving time when trying to make a patient transfer.

Number	Success Criteria Description	Has it been met yet?
1	My program should allow a user to login with the correct credentials.	Yes
2	The day case to overnight patient data present in the database should be represented on a bar chart.	Yes
3	Information must be logically displayed on each page	Yes
4	My program must be able to display pages in a sensible time.	Yes
5	Data entered into the data base should be valid and stored in the correct format.	Partially – Data can be entered into tables but this is not checked for validation.
6	The colour scheme should be representable of The London Clinic: dark blue and white.	Yes

7	My program should be able to run on a standard desktop/laptop in the clinic.	Yes
8	Daily database for each day should refresh every day to allow for new data.	Yes
9	The program colour scheme should be same as the OS theme used.	No – Still optional
10	Data can be seen/altered at any time	Partially – Data can be altered in the daily grid at any time but the current grid cannot be edited yet
11	Make sure I have an updated database/tables	Yes – tables updates when new data is saved.

Client Feedback from Prototype 2:

Question	Client Feedback	How I will respond to this
Does the prototype provide a clear and easy way for nurses to make audits of patient transfers?	Yes, now I have seen it properly it seems to provide the information we want to gather at a good speed making it more user-friendly	I will carry on making sure the audit is able to always do this even if I change lots of things within my program.
Are you happy with the transferring of data across grids?	Yes, I am pleased with the execution of data being transferred across grids. However, I would you to add a function so data could be added to the current month grid as it cannot be edited as of now.	To resolve this, I will make a similar function to my daily grid where data can be added and save to the database.
Are you happy with the security of the prototype?	I like how the security of the program has improved, as now the username and password is checked for validity and checked with the password in the database. Although, in the future if this is used on a bigger scale, where users can create accounts, I would like to see more safety measures put in place if someone gain unauthorised access to the saved passwords.	Taking this into account, I will continue monitoring the login process, making sure there are no hiccups. In the future, I may add encryption if required.
Are you happy with the speed of the program?	Yes, the speed is fast when switching between windows but it could be quicker when loading functions such as at the end of the month when the current month grid button is clicked.	To increase the speed of navigating the program, I will make sure to add a timer to wait for the function to finish then load the subwindow.
Are you happy with the presentation of the data in the bar graphs?	The presentation of the data in the bar graphs is very clear directly indicating which consultants are responsible for patient	Considering this, I will make sure the graphs are accessible while I am making changes to the

	transfers. I can also save the bar graph to my files if I ever need a copy of it so I can send the data to managers on different wards in the hospital.	program and maintain data integrity.
Is there anything you don't like about this prototype?	There are only some things I do not like which should be implemented into the program such as the outdated consultant list. I would like you to add Professor Lim and Professor Jackson to the consultant's key and when logging in I cannot press "enter" on my keyboard to access the program. Therefore, make it so when the user presses the enter button, they go straight into the software instead of pressing the login button, which saves our time.	I will make the following changes to the current version of my software, improving my client satisfaction towards my program.
Is this prototype, so far, in line with the specification for the system you have requested?	Yes, the program does the job it intends to do and it gives a clear graph format and identifies key trends accurately which can help provide the feedback and data required for the senior management.	From this feedback, I will carry on producing high level code making sure the program meets its purpose every time a change is made.

Prototype 3:

Editing the current Month Grid

At the moment, the users can only edit the daily grid, but if the user wants to make a change to that day but it is past that date they can't. Therefore, I am implementing another function to make changes to the current month grid and the changes should be saved to the database.

While trying to add this function I came across this error where my **tbl_CurrentMonth** had an extra empty character in the SQL so my "comments" were not adding to the incorrect "comments " column in the table.

```
#This creates the CurrentMonth table in the DB
cur.execute("""CREATE TABLE "tbl_CurrentMonth" (
    "ConsultantID"  TEXT NOT NULL,
    "wardID"        TEXT NOT NULL,
    "RoomNo"        INTEGER NOT NULL,
    "TimeOfAdmission"       time NOT NULL,
    "TransferDate"  date NOT NULL,
    "TransferTime"  time NOT NULL,
    "ReasonsforTransfer"    INTEGER NOT NULL,
    "Comments "     TEXT,
    "Month" TEXT NOT NULL
```

```
Audit.py", line 1000, in TransferDailyTable
    cur.execute("INSERT INTO tbl_CurrentMonth (ConsultantID, WardID, RoomNo, TimeOfAdmissio
, TransferDate, TransferTime, ReasonsForTransfer, Comments , Month) VALUES (?, ?, ?, ?, ?,
, ?, ?,?)", (consultant_id, ward_id, Room_No, Time_Of_Admission, transfer_date, transfer_ti
e, reason_for_transfer,comments,Month))
sqlite3.OperationalError: table tbl_CurrentMonth has no column named Comments
```

To solve this problem, I created these database tables again and changed the "comments " to the word without the extra empty space. I also made the field null so it can store empty values as the user does not always have to fill in the comments box(optional).

Now to not lose any data I have to also save the data from the daily grid when it is being transferred to the current month table in my database. To do this I edited my TransferDailyTable() function to insert values into the current month table.

Patient Transfer Audit.py - C:\Users\216298\OneDrive - Kingsmead\Alevel CS coursework 2022-2023\Patient Transfer Audit.py (3.9.2)

File Edit Format Run Options Window Help

```
def TransferDailyTable(button):
    seconds = time.time()#gets the current time in seconds
    local_time = time.ctime(seconds)#converts the time in seconds into a readable format
    print("Local time:", local_time)#prints the readable time
    current_time = time.localtime()#gets the struct_time function in format(year, month, day, hour, minute, second, weekday, day of the year, daylight saving)
    today = datetime.date.today() #gets today's date
    print(today)
    hour = time.strftime("%H:%M:%S",current_time)#puts the current time in H/M/S format
    str_hour = (str(hour))#makes hour variable a string
    str_hour=str_hour.strip()
    hour,minute,second=str_hour.split(":")#splits the time into single integers of hour,minute and second
    if hour >="00":#and hour <"00":#if hour is between 7pm and 8pm run the funtion
        print("hey")
        rows = app.getGridRowCount("g1")#gets number of rows with data
        for i in range(rows):
            row = app.getGridRow("g1", i)
            print(row)
            print(rows)
            rowData=[]#empty list to transfer data
            for eachRow in row:
             if any(row):#for any row in the table
                        consultant_id = row[0]#assigns each row index to the corresponding field name
                        ward_id = row[1]
                        Room_No = row[2]
                        Time_Of_Admission = row[3]
                        transfer_date = row[4]
                        transfer_time = row[5]
                        reason_for_transfer = row[6]
                        comments = row[7]
                        Month = row[8]

        conn = sqlite3.connect('PatientTransferAudit.db')#connect db
        cur = conn.cursor()
        cur.execute("INSERT INTO tbl_CurrentMonth (ConsultantID, WardID, RoomNo, TimeOfAdmission, TransferDate, TransferTime, ReasonsForTransfer, Comments , Month) VALUES (?, ?, ?, ?, ?,
        conn.commit()#commits all changes into the tbl_Daily in my database
        print("complete")



        for x in range(rows):
                data=app.getGridRow("g1",x)#for each row in the table store values in the variable data
                rowData.append([data])#2D list
        print("row data",rowData)


        if row[8] == "January" or row[8] == "Febraury" or row[8] == "March" or row[8] == "April" or row[8] == "May" or row[8] == "June" or row[8] == "July" or row[8] == "August" or row[8]
            for aList in rowData:#for each list in rowData
```

This is edited version of the TransferDailyTable() which is similar to the retrive_tbl_Daily() where each row is assigned an index and the data in that row Is stored to the respected field. Each row is inserted into the current month table using the SQL statement.

(The hours here is set “00” as I was running the code outside the specified hours of 7pm-8pm)

Daily

ConsultantID	WardID	RoomID
RH	3	303

	ConsultantID	wardID	RoomNo	TimeOfAdmission	TransferDate	TransferTime	ReasonsforTransfer	Comments	Month
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	RH	3	303	10:20	12/03/23	11:00	3	Ward closed early	March

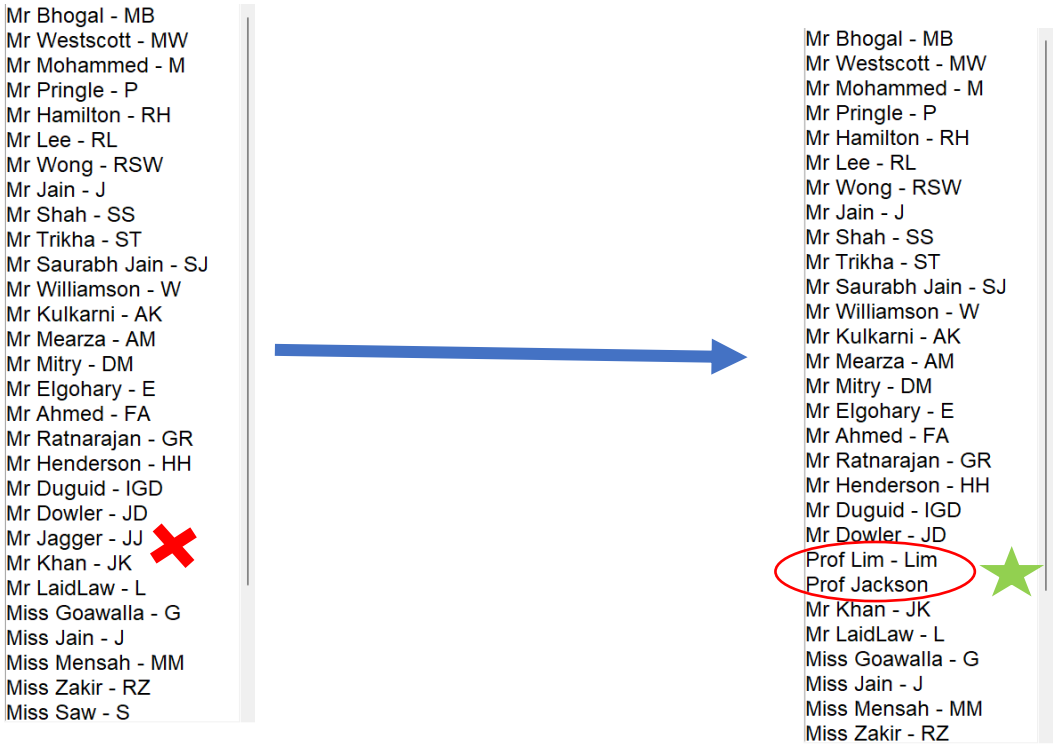
Now this is done I can implement a function to edit the actual current month grid and save it to the current month table in the database. I made a SaveTableg2() function which is similar to the SaveTableg2() function.

```
def SaveTableg2(btn):
    print("hellooooooo")
    save = app.yesNoBox("Save transfer 2", "Are you sure you want to add this patient transfer to this month? ALL CHANGES ARE FINAL!!",)

    if save==True:
        data = app.getGridRow("g2", btn)#gets inputs from the row and stores it in the variable data
        print(data)
        if data is None:
            print("No data")#prints no data if row is empty
        else:
            for row in data:
                print(row)
                if any(row):#for any row in the table
                        consultant_id = data[0]#assigns each row index to the corresponding field name
                        ward_id = data[1]
                        Room_No = data[2]
                        Time_Of_Admission = data[3]
                        transfer_date = data[4]
                        transfer_time = data[5]
                        reason_for_transfer = data[6]
                        comments = data[7]
                        Month = data[8]
        conn = sqlite3.connect('PatientTransferAudit.db')#connect db
        cur = conn.cursor()
        cur.execute("INSERT INTO tbl_CurrentMonth (ConsultantID, WardID, RoomNo, TimeOfAdmission, TransferDate, TransferTime, ReasonsforTransfer, Comments, Month) VALUES (?, ?, ?, ?, ?, ?,
        conn.commit()#commits all changes into the tbl_CurrentMonth in my database
        print("complete")

    else:

        app.showSubWindow("CurrentMonthDatabase")
```

When the save button was pressed in g2 the current whole role is placed into the corresponding field indexes stored as a variable with the field names. These variables are then inserted into the current month table in my database.

Listening to my client, I deleted the consultants who are no longer in practice at the clinic from my program and added 2 more consultants.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**



Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

To ensure my systems certainty and to main data consistency in my program, I have added validity to some fields within my grids so each cell in the row must be in a specific format to be saved to be saved to the database.

```
check_Row_Inputs_g1
    seconds = time.time()#gets the current time in seconds
UnboundLocalError: local variable 'time' referenced before assignment
```

When I was creating the function to check the validity of each cell in the daily grid, I came across this error. I realised this was occurring because I have already defined **time** earlier in my code. To fix this error I renamed the **time** module when I imported it to `import time as mytime` so the **time** module functions could be used without conflicting any other **time** variables in my code.

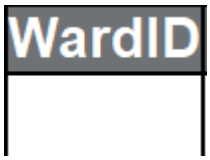
```
def check_Row_Inputs_g1(btn):

    data = app.getGridRow("g1", btn)#gets inputs from the row and stores it in the variable data
    print(data)
    if data is None:
            print("No data")#prints no data if row is empty
    else:
        for row in data:
            print(row)
            if any(row):#for any row in the table
                    consultant_id = data[0]#assigns each row index to the corresponding field name
                    ward_id = data[1]
                    Room_No = data[2]
                    Time_Of_Admission = data[3]
                    transfer_date = data[4]
                    transfer_time = data[5]
                    reason_for_transfer = data[6]
                    comments = data[7]
                    Month = data[8]

    seconds = mytime.time()#gets the current time in seconds
    local_time = mytime.ctime(seconds)#converts the time in seconds into a readable format
    current_time = mytime.localtime()# Get the current date
    date = mytime.strftime("%d/%m/%y", current_time)#specifies that the date should be in DD/MM/YY format
    time = mytime.strftime("%H:%M", current_time)#specifies that the time should be in HH/MM format

    if ward_id == "" or ward_id!=int:#print error message if wardID cell is empty or not an integer
        app.errorBox("Cannot save","WardID is not a number")
        return False#sets subroutine to false

    elif Room_No == "" or Room_No!=int:#print error message if RoomNo cell is empty or not an integer
        app.errorBox("Cannot save","Room ID is not a number")
        return False#sets subroutine to false

    elif transfer_date == "" or transfer_date!= date:#print error message if TransferDate is empty or date is not in DD/MM/YY format
        app.errorBox("Cannot save","Date is not in DD/MM/YY format")
        return False#sets subroutine to false

    elif transfer_time == "" or transfer_time!= time:#print error message if TransferTime is empty or time is not in HH/MM format
        app.errorBox("Cannot save","Time is not in HH/MM format")
        return False#sets subroutine to false

    elif reason_for_transfer == "" or reason_for_transfer!=int:
        app.errorBox("Cannot save","Reason for transfer is not a number")#print error message if ReasonForTransfer cell is empty or not an integer
        return False#sets subroutine to false
```

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

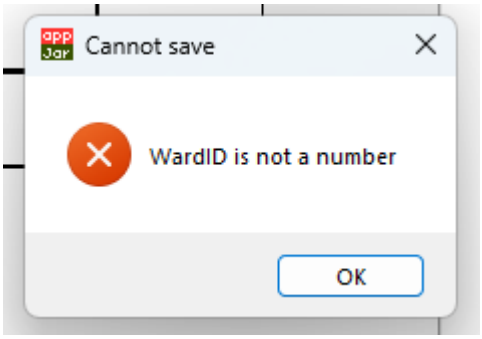
Here is final function to check the inputs of the 'daily"/"g1" grid. The function takes each row and stores it in the variable **data**. The grid is iterated over and each value in the row is placed into the corresponding field index. I used the import **time** module to get the current date and time and set them both to the correct format (DD/MM/YY and HH/MM). I have then created if statements to check most of the cells in each row. If the cell is not in its expected format the appropriate error message will displayed, so the user would know what to change in order to successfully add rows to the database.

When I was running the function the "Are you sure you want to save?" popup kept on coming up even if the data in the row was invalid. Therefore, to fix this, I made sure that when the check_Row_Inputs_g1() was set to False if the data in the row was not valid. I then also altered the SaveTableg1() function, so if the previous subroutine is set to false the rest of the code in the SaveTableg1() function would not take place. The SaveTableg1() function would only take place if check_Row_Inputs_g1() was to return true.

Example:



=



empty wardID

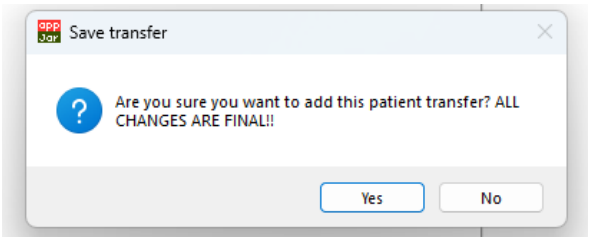
However, when I tried adding a number into the **wardID** the same error popup was coming up which was incorrect. I realised the rows are taken from the grid by **lists** that contain **strings** so I had to check whether there were **integers in the string**. To do this I used the isdigit() function where I found at https://www.w3schools.com/python/ref_string_isdigit.asp#:~:text=The%20isdigit()%20method%20returns,considered%20to%20be%20a%20digit.

I edited **wardID, RoomNo** and **ReasonForTransfer** if statements as those were the main fields which contained numbers. I then repeated this for the current month grid too.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

```
if  ward_id == "" or (not ward_id.isdigit()):#print error message if wardID cell is empty or
    app.errorBox("Cannot save","WardID is not a number")
    return False#sets subroutine to false


elif Room_No == "" or (not Room_No.isdigit()):#print error message if RoomNo cell is
    app.errorBox("Cannot save","Room No is not a number")
    return False#sets subroutine to false



elif reason_for_transfer == "" or (not reason_for_transfer.isdigit()):
    app.errorBox("Cannot save","Reason for transfer is not a number")#print error
    return False#sets subroutine to false
```

4 =



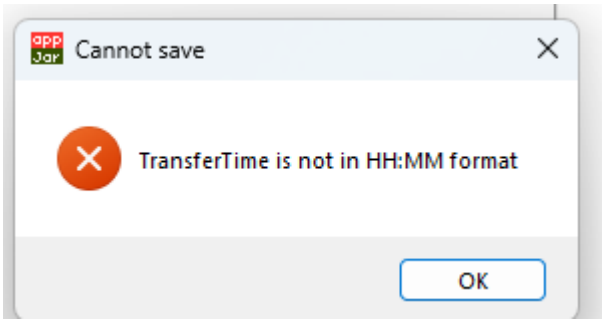
A similar thing was happening with the **date** and **time** fields as it was checking if the cell matched the **current** date and time as well as the **format**, while I only wanted to check the format of the **date** and **time.** This was because datetime return a **datetime object** which cannot be compared to an **entered string.** Therefore, I had to use try an except commands to parse the time and catch the "ValueError" (https://docs.python.org/3/library/exceptions.html)if the time or date is not in the correct format as this was the most suitable way of validating my grid with the limited AppJar software I was using. I also did this for the daily grid and current month grid.

```
if  ward_id == "" or (not ward_id.isdigit()):#print error message if wardID cell is empty or not an integer
    app.errorBox("Cannot save","WardID is not a number")
    return False#sets subroutine to false

elif Room_No == "" or (not Room_No.isdigit()):#print error message if RoomNo cell is empty or not an integer
    app.errorBox("Cannot save","Room No is not a number")
    return False#sets subroutine to false

elif reason_for_transfer == "" or (not reason_for_transfer.isdigit()):#print error message if ReasonForTransfer cell is empty or not an integer
    app.errorBox("Cannot save","Reason for transfer is not a number")
    return False#sets subroutine to false
try:
    datetime.datetime.strptime(Time_Of_Admission, '%H:%M')#print error message if Time_Of_Admission is empty or the time is not in HH/MM format
except ValueError:#if right type but inappropiate value used run except clause
    app.errorBox("Cannot save","TimeOfAdmission is not in HH:MM format")
    return False#sets subroutine to false

try:
    datetime.datetime.strptime(transfer_date, '%d/%m/%y')#print error message if TransferDate is empty or the date is not in DD/MM/YY format
except ValueError:#if right type but inappropiate value used run except clause
    app.errorBox("Cannot save","TransferDate is not in DD/MM/YY format")
    return False#sets subroutine to false

try:
    datetime.datetime.strptime(transfer_time, '%H:%M')#print error message if TransferTime is empty or the time is not in HH/MM format
except ValueError:#if right type but inappropiate value used run except clause
    app.errorBox("Cannot save","TransferTime is not in HH:MM format")#sets subroutine to false
    return False


return True
```

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

10

=



In regards to my client's feedback, I am going create a function so when the enter button is pressed on the keyboard when the user is on the login page, it will run the function to check the validity of entries, then determine if the user can gain access to the program or not.

After researching in the Appjar website (**http://appjar.info/pythonEvents/#binding-keys**), I noticed there is a function called. enableEnter(function) which binds the "Enter/Return" button on the keyboard to whatever function is passed as the parameter. This means that when the "Enter" key is pressed that specific function will run.

Using this information this is this what I did:

```
def createInterface():
####Login page#########
    app.startSubWindow("Login")
    app.addLabel("title","DSW Transfer Audit" ,0,1)
    app.getLabelWidget("title").config(font=("","20", "bold","underline"))
    app.setLabelBg("title", "white")
    app.setLabelFg("title", "blue")
    app.addLabelEntry("Username :",1,1)
    app.addSecretLabelEntry("hiddenPassword",2,1)
    app.setLabel("hiddenPassword","Password :")
    app.addLabelEntry("visiblePassword",2,1)
    app.setLabel("visiblePassword","password:")
    app.addImageButton("eye",showPass,"eye.png",2,2)
    app.addImageButton("eyehide",hidePass,"eyehide.png",2,2)
    app.enableEnter(validLogin)
    app.setEntryBg("Username :", "lightblue")
    app.setEntryBg("hiddenPassword", "lightblue")
    app.setEntryBg("visiblePassword", "lightblue")
    app.setFocus("Username :")
    app.setBg("white")
    app.startLabelFrame("", 0, 2)
    app.addImage("tlc", "tlc.png", 0,2)#Adds the company logo
    app.setImageSize("tlc", 200,200)
    app.setGuiPadding(10,10)
    app.stopLabelFrame()
    app.setFont(18)
    app.addButton("Login", press, 3, 0)
    app.addButton("Forgot Login",press, 3, 2)
    app.setButtonBg("Login", "lightblue")
    app.setButtonBg("Forgot Login", "lightblue")
    app.showSubWindow("Login")
    app.stopSubWindow()
```

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

I have passed in my ValidLogin() function so that when the user is within the login page if they pressed the enter button the ValidLogin() would run. This will either deny or allow them access to the program.

However, when I ran my code again and pressed “enter” while in any of the other SubWindows , ValidLogin() would run when it shouldn’t. To fix this I used the disableEnter() function ,also found at the AppJar website, to disable the set enter button if the user had past the login sub window.

```
#if credentials are valid user can login successfully
    else:
        if result[0]==hidden_password  or result[0]==visible_password:
            app.hideSubWindow("Login")#User may login as details are correct
            app.clearEntry("visiblePassword")#clears password entry when user logs in
            app.clearEntry("hiddenPassword")
            app.showSubWindow("MainMenu")#Shows the MainMenu
            app.unbindKey("Return")
```

This is within the ValidLogin() function, so when the entered details are correct the main menu would be shown and the “enter” button on the keyboard will be **disabled** from running this function again.

```
def GraphSummary():
    current_time = time.localtime()#gets the struct_time function in format(year, month, day, hour, minute, second, weekday, day of the year, daylight saving)
    hour = time.strftime("%H:%M:%S",current_time)#puts the current time in H/M/S format
    str_hour = (str(hour))#makes hour variable a string
    str_hour=str_hour.strip()
    hour,minute,second=str_hour.split(":")#splits the time into single integers of hour,minute and second
    today = datetime.date.today()#gets todays date
    last_day_of_month = calendar.monthrange(today.year,today.month)[1]#gets the exact number of days in the current month
    date = datetime.date(2023, 4, 30)
    if date.day == last_day_of_month and (hour >= "20" and hour < "21"):#if the day is the last of the month and it is between 8pm and 9pm count the totals for the current month
```

I also quickly added a **timing factor** for this GraphSummary() function to run as I realised, I forgot to add it. This will work just like the time in any of the other functions. If the date is the last day of the month and the time is between 8pm and 9pm, the graph’s summary can be calculated.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Since the grids I created were slightly squished, I altered the size of the grids for the users to clearly see the different categories, making it straightforward to view the information they are looking for. I did this by adjusting the **rowspan** and **colspan** of the grid causing it to increase in width in the specified row and column[x,y]. To enable the grid to stretch horizontally across its container, I utilized the **app.setSticky("ew")** function in my code. This function sets the sticky property to "east and" west," which allows the grid to expand and fill the available horizontal space within its container. This change is shown below.

```
app.setSticky("ew")
app.addGrid("g1",

[["ConsultantID", "WardID","RoomNo", "TimeOfAdmission","TransferDate","TransferTime","ReasonForTransfer","Comments","Month"],

["","","","","","","","","",""],

["","","","","","","","","",""],

["","","","","","","","","",""],

["","","","","","","","","",""]],
llows for the features of the grid to exist and when user right-clicks on an entry box a sub-menu pops up which allows for changes to the grid
  action=SaveTableg1,showMenu=True,actionButton ="save",addRow=addGridRowg1,row=1,column=1, rowspan=9, colspan=9)
```

I done this for both of my grids. ("g1","g2")

Transfer reasons key

1 - Medical Reason

2 - Patient Request

3 - Late discharge

4 - Consultant Request

5 - Social Reason

6 - Other

Daily

ConsultantID	WardID	RoomID	TimeOfAdmission	Trans

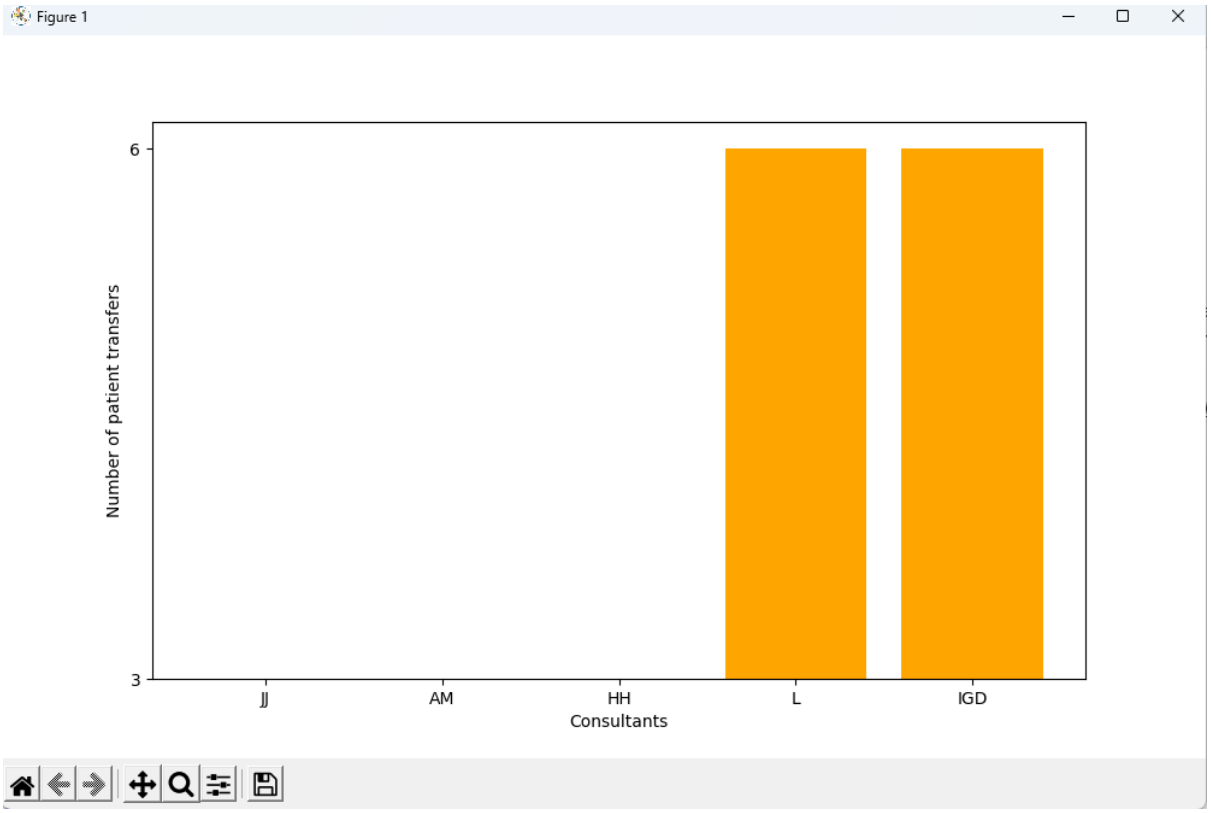
Consultants Key

Mr Bhogal - MB
Mr Westscott - MW
Mr Mohammed - M
Mr Pringle - P
Mr Hamilton - RH
Mr Lee - RL
Mr Wong - RSW
Mr Jain - J
Mr Shah - SS
Mr Trikha - ST
Mr Saurabh Jain -
Mr Williamson - W
Mr Kulkarni - AK
Mr Mearza - AM
Mr Mitry - DM
Mr Elgohary - E
Mr Ahmed - FA
Mr Ratnarajan - G
Mr Henderson - HI
Mr Duguid - IGD
Mr Dowler - JD
Prof Lim - Lim
Prof Jackson
Mr Khan - JK
Mr LaidLaw - L
Miss Goawalla - G
Miss Jain - J
Miss Mensah - MM
Miss Zakir - RZ

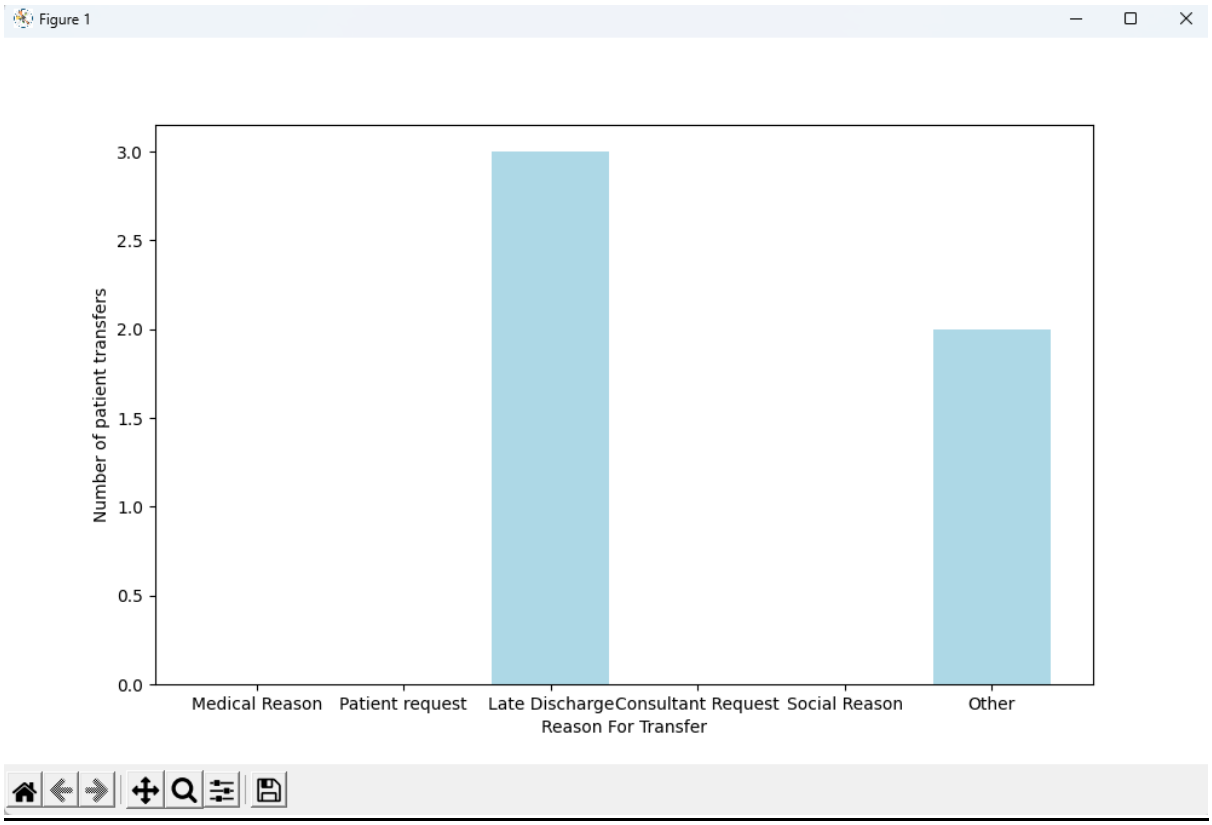
To improve visual impressions and user friendliness I also decided to edit my graphs and add labels to the **x** and **y** axis. I also made the bar colours a different colour so it is clearer and more presentable for the nurses to identify which graph is which.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Consultant Graph



Monthly Graph



Prototype 3 finished

Number	Success Criteria Description	Has it been met yet?
1	My program should allow a user to login with the correct credentials.	Yes
2	The day case to overnight patient data present in the database should be represented on a bar chart.	Yes
3	Information must be logically displayed on each page	Yes
4	My program must be able to display pages in a sensible time.	Yes
5	Data entered into the data base should be valid and stored in the correct format.	Yes
6	The colour scheme should be representable of The London Clinic: dark blue and white.	Yes – My client wanted this colour scheme only.
7	My program should be able to run on a standard desktop/laptop in the clinic.	Yes
8	Daily database for each day should refresh every day to allow for new data.	Yes
9	The program colour scheme should be same as the OS theme used.	No –The client decided to stick with the colours of the hospital so this success criteria point can be ignored
10	Data can be seen/altered at any time	Yes – Data can be altered in the daily grid and the current grid at anytime
11	Make sure I have an updated database/tables	Yes – tables updates when new data is saved.

Client Feedback from Prototype 3:

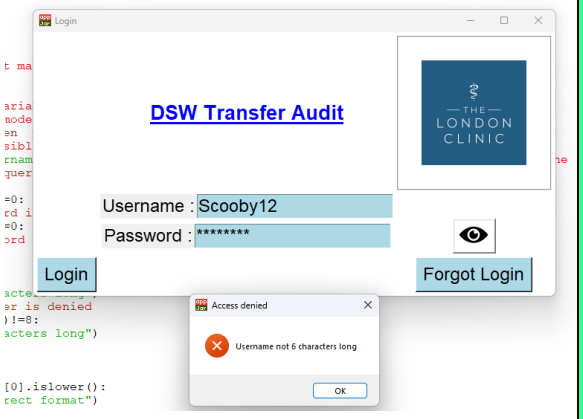
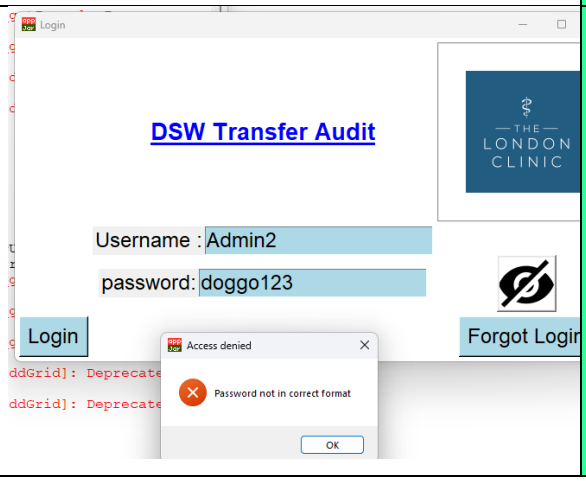
Is there anything you do not like about this prototype?	There is only one thing I do not like about this prototype is that you must wait slightly in the patient transfer database sub window when accessing the current month database at the end of the month because the program takes a couple seconds to load the data in the table.	This is true because due to the GUI software I am using it sometimes takes time to load large amounts of data. Therefore, I am limited on what I could do here, so this slight delay will have to be kept until a future update is available to AppJar, that will allow me to improve the speed.
Is this prototype in line with the specification for the system you have requested?	Yes, the program does the necessary job and it gives a clear graph format and identifies key trends. The colours are accurate to The London Clinic and this is pretty much what I expected the software to be like.	From this information I will not change the layout of the program or the colour scheme but concentrating on evaluating the flaws and strengths of my system in testing.
Do you have any suggestions for additional features or improvements that could be made to the prototype beyond the original specifications?	The software could be able to provide more detail on each reason for transfer within the graphs and a way users can add new consultants to the system so the people who work for the clinic can be kept up to date.	I could add more detail to the graphs but due to my limited skill, it would be tricky so it will have to be added in the future. In addition, the integration of new consultants to the consultant's key was not an original requirement for the program, therefore I will delay the implementation of this feature to a later date.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

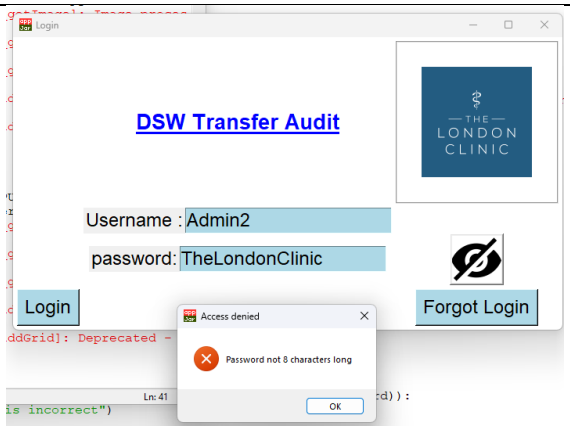
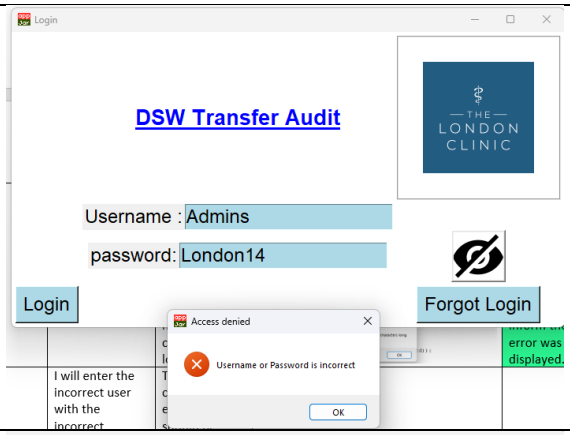
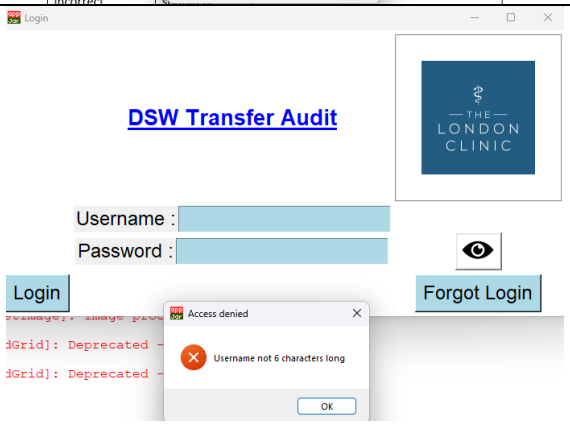
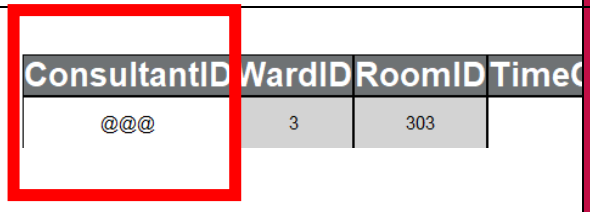
Post-Development Testing: Alpha Testing

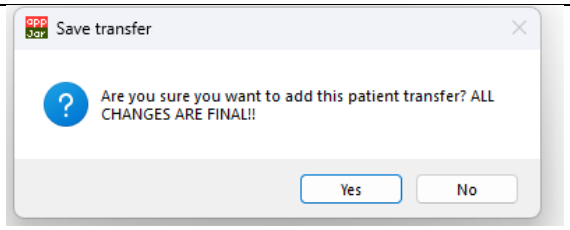
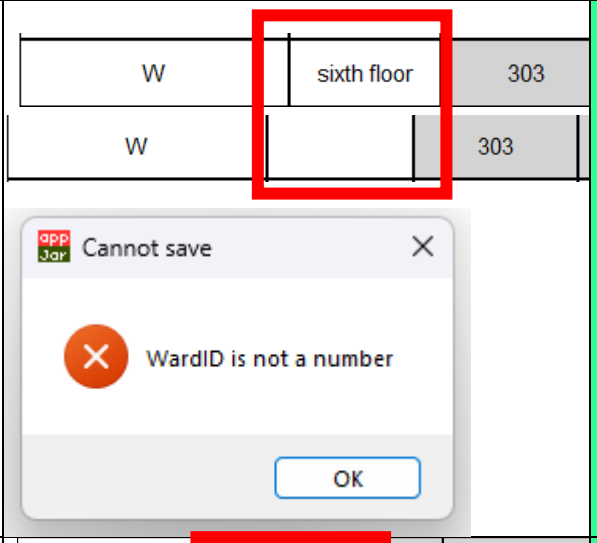
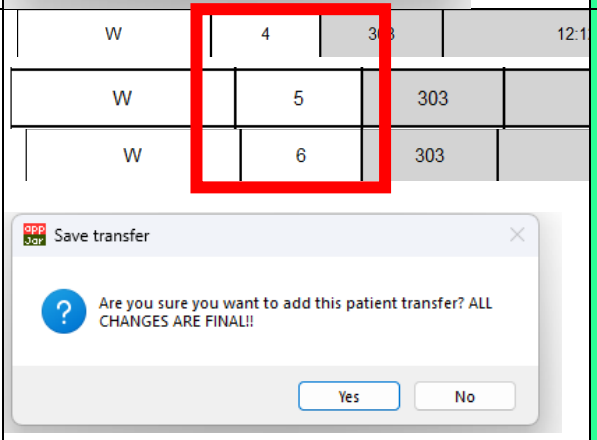
White Box testing – Testing all the text inputs

All the white box testing was carried out by me the developer and for tests with multiple inputs on a page, assume all other entries are filled and are valid, apart from the input being tested.

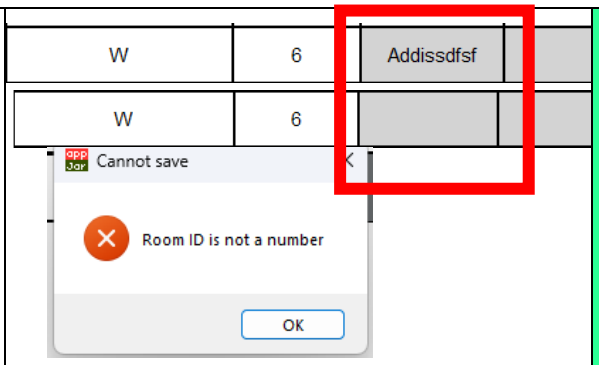
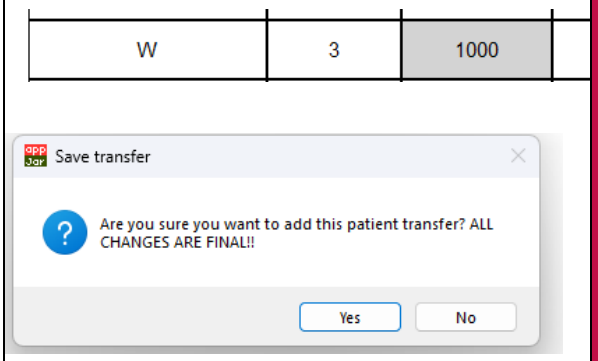
Input Name	Test	Outcome	Screenshot	Was it successful?
Username and Password input boxes (login screen)	I will enter an existing username and its appropriate password: **Admin1 London12**	The main menu in the program should be displayed.		Yes, the test was successful and the user was taken to the main menu page.
	I will enter an incorrect username with the correct password: **Scooby12 London12**	The login credentials error label should be displayed, informing the user that the username is not 6 characters long.		Yes, the test was successful, entry to the main menu was denied and the message to inform the error was displayed.
	I will enter the correct username with incorrect password: **Admin2 doggo123**	The login credentials error label should be displayed, informing the user that the password is not in the correct format (without capital letter)		Yes, the test was successful, entry to the main menu was denied and the message to inform the error was displayed.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

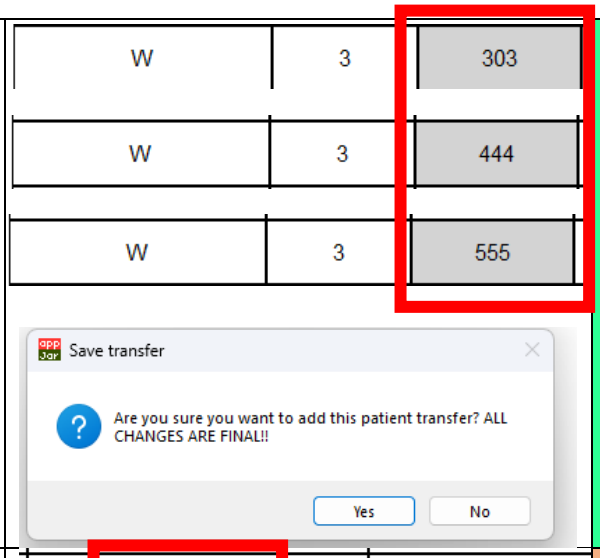
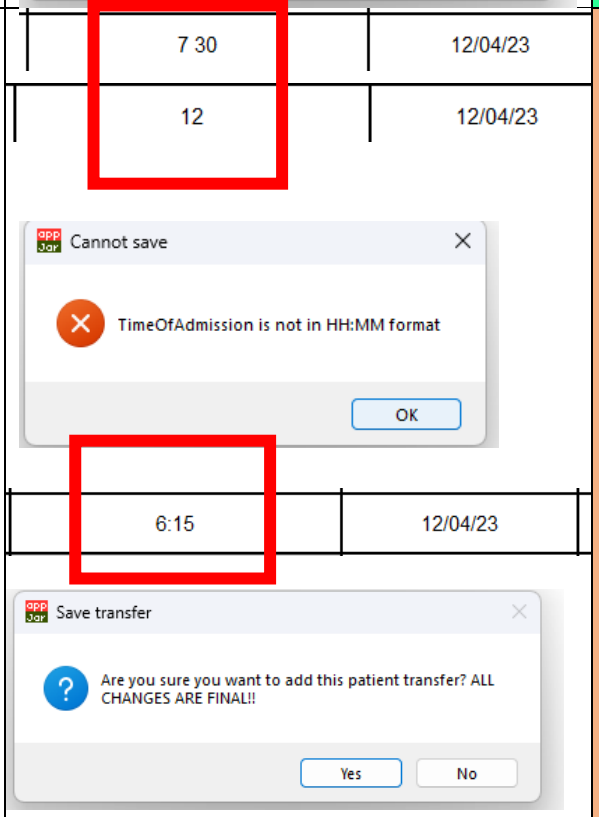
	I will enter the correct username with the incorrect password character length: **Admin2 TheLondonClinic**	The login credentials error label should be displayed, informing the user that the password is not 8 characters long.		Yes, the test was successful, entry to the main menu was denied and the message to inform the error was displayed.
	I will enter the incorrect username with the incorrect password: **Adminsss hospital 23**	The login credentials error label should be displayed, informing the user that the username and password is incorrect.		Yes, the test was successful, entry to the main menu was denied and the message to inform the error was displayed.
	I will enter a **null** input to both input boxes	The login credentials error label should be displayed, informing the user that the username and password is incorrect.		No, the test was unsuccessful, the incorrect error label was displayed as there is no function to a display a label when nothing is entered into the input boxes. Therefore, I would need to add a validation textbox for null inputs.
Daily/Current Month grid input boxes (Daily/Current Month	I will enter an incorrect data format for the consultant ID: **@@@**	The corresponding error label should be displayed, informing the		No, the test was unsuccessful as the save textbox was being

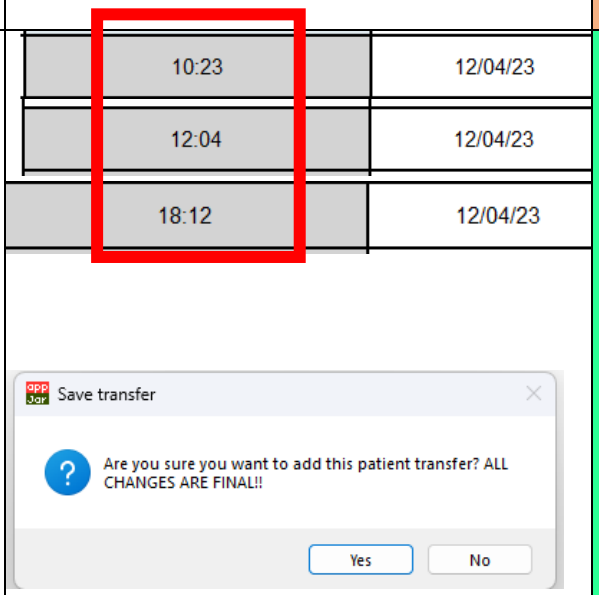
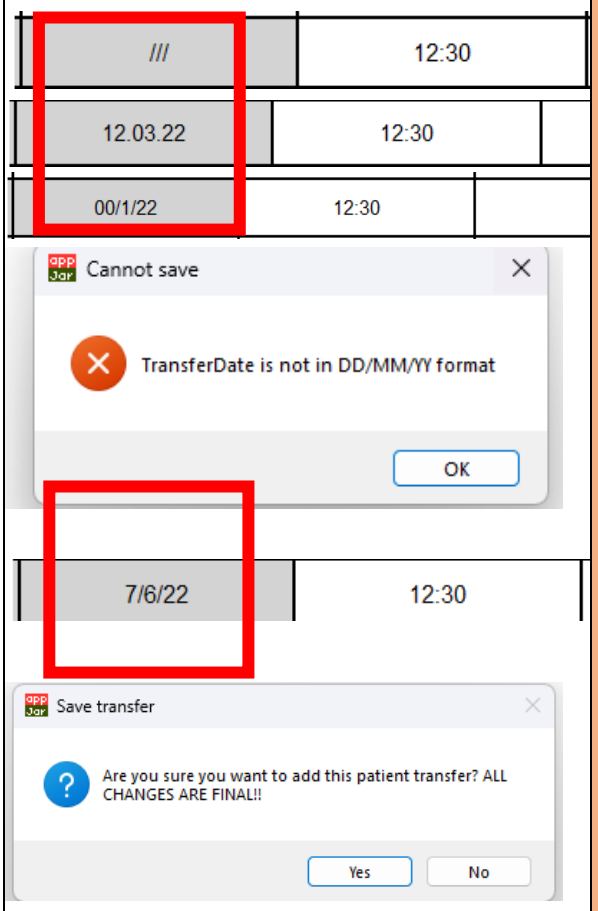
Database screen)		user that the data entered is not in the correct format		displayed using my previous test data as I forgot to add the validation for erroneous data in the consultant ID. Therefore, after user testing, I will fix this minor issue
	I will enter an incorrect data format for the WardID: **sixth floor** **“ ”**	There should be a label error displayed, informing the user that the WardID should only be numbers		Yes, the test was successful, as saving the row to the database table was denied and the corresponding error message was displayed.
	I will enter the correct data format for the WardID: **4** **5** **6**	There should be a save textbox displayed with no error label		Yes, the test was successful, as the user could save the row data to the database table because the save textbox was displayed.

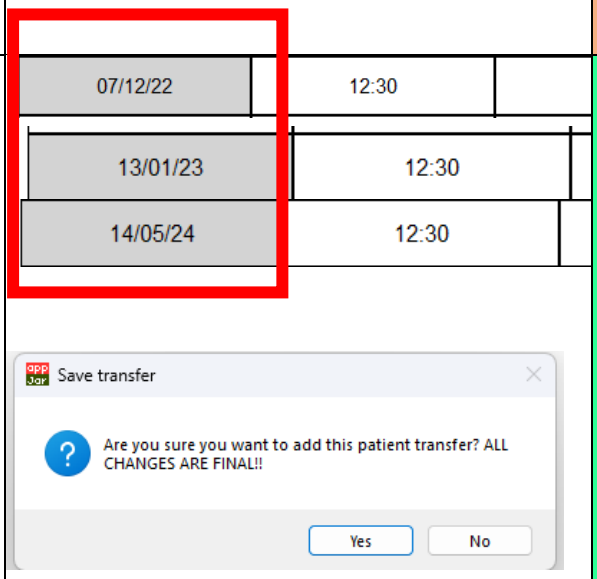
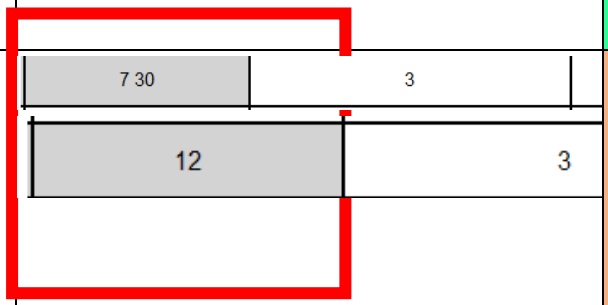
Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

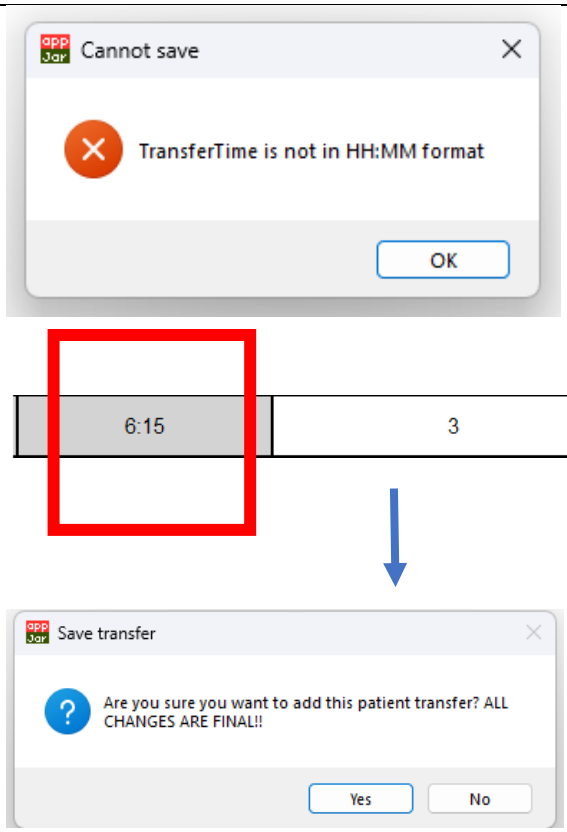
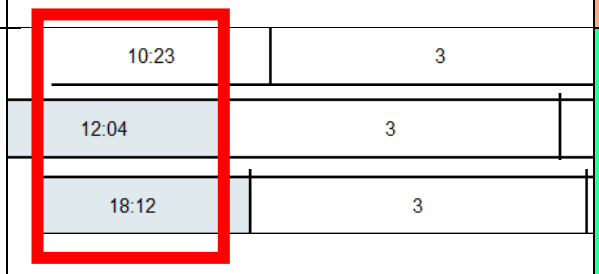
	I will enter an incorrect data format for the RoomID: **Addissdfsf** **“ ”**	There should be an error label displayed, informing the user that RoomID is not a number.		Yes, the test was successful, as saving the row to the database table was denied and the corresponding error message was displayed.
	I will enter an incorrect boundary data format for the RoomID: **1000**	There should be an error label displayed, informing the user that RoomID is not 3 digits long.		No, the test was unsuccessful as the save textbox was being displayed using my previous test data as I forgot to add the validation to check for the number of digits in the RoomID. Therefore, after user testing, I will alter the validation so the users cannot enter more than 3 digits.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

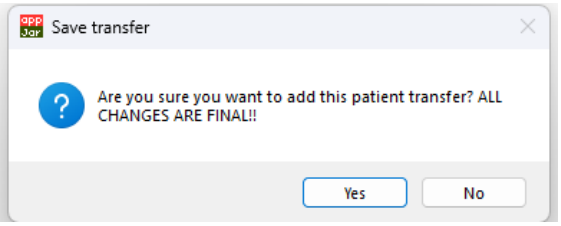
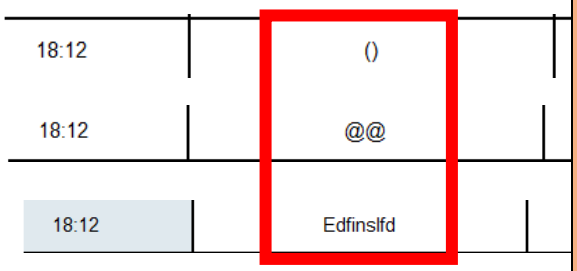
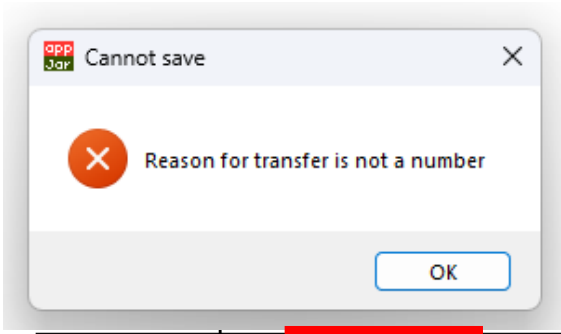
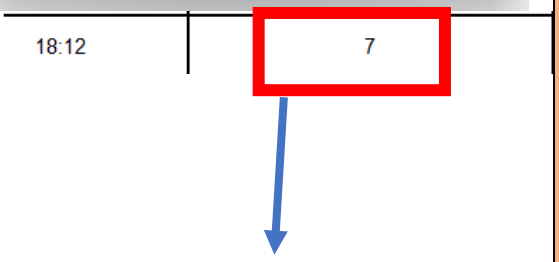
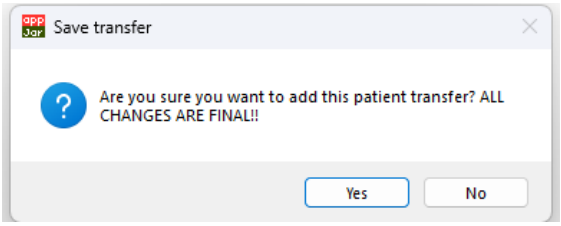
	I will enter the correct data format for the RoomID: **303** **444** **555**	There should be no error label and the save transfer textbox should be displayed.		Yes, the test was successful, as the user could save the row data to the database table because the save textbox was displayed.
	I will enter the incorrect data format for the TimeOfAdmission: **7 30** **6:15** **12**	There should be error label displayed, informing the user that the time is not in HH:MM format and shows the user it can only be the 24-clock period		Partially yes because the correct error message was displayed to the user when some of the incorrect data was inputted. However, when **6:15** was inputted, it was still recognised as valid even when it is clearly not in the 24-hour format. Despite this, this is not a major problem as any nurse viewing this data will know this specific time is the morning as most of the other times in the grid will be in 24- hour format. Moreover, to

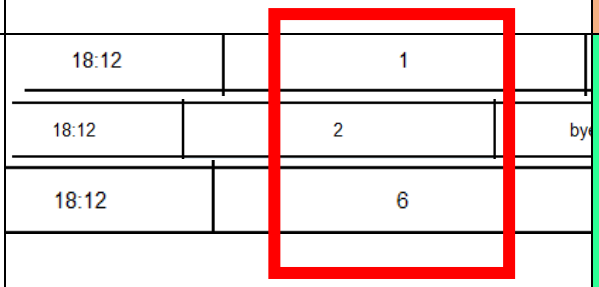
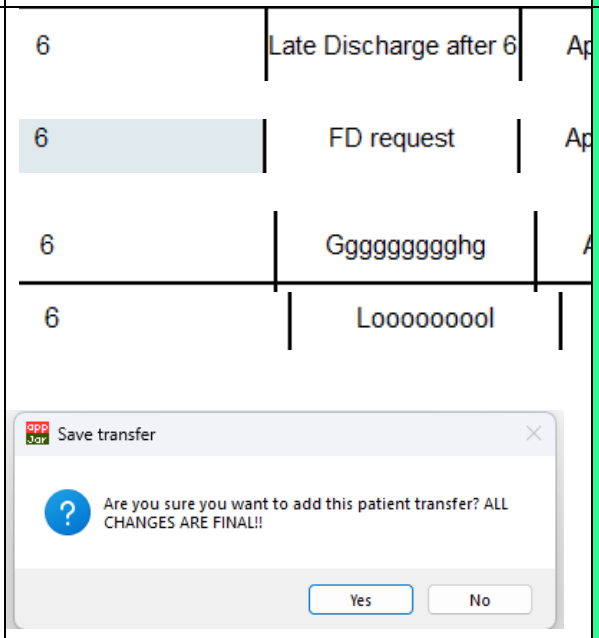
				make it clearer to all users I could fix this slight problem in the future.
	I will enter the correct data format for the TimeOfAdmission: **10:23** **12:04** **18:12**	There should be no error label and the save transfer textbox should be displayed.		Yes, the test was successful, as the user could save the row data to the database table because the save textbox was displayed and the time was in valid 24 - hour format.
	I will enter an incorrect data format for the TransferDate: **///** **12.03.22** **00/1/22** **7/6/22**	There should be error label displayed, informing the user that the date is not in DD/MM/YY format.		Partially yes because the correct error message was displayed to the user when some of the incorrect data was inputted. However, when **7/6/22** was inputted into the TransferDate textbox the save transfer pop up was being displayed even though the inputted data wasn't in **DD/MM/YY** format. Although, this is not a real concern

				because most nurses who read the date would recognise it as what it should be so it will be highly unlikely there would miscommunication of dates through these grids. Besides this, I could implement this little change in future improvements of this program. (Expanded on with non -uk nurse's perspectives in evaluation)
	I will enter the correct data format for the TransferDate: **07/12/22** **13/01/23** **14/05/24**	There should be no error label and the save transfer textbox should be displayed.		Yes, the test was successful, as the user could save the row data to the database table because the save textbox was displayed and the date was in DD/MM/YY format.
	I will enter an incorrect formatted data for the TransferTime: **7 30** **6:15**	There should be error label displayed, informing the user that the time is not in HH:MM		Partially yes because the correct error message was displayed to the user when some of the

	12	format and shows the user it can only be the 24-clock period		incorrect data was inputted. However, when **6:15** was inputted, it was still recognised as valid even when it is clearly not in the 24-hour format. Despite this, this is not a major problem as any nurse viewing this data will know this specific time is the morning as most of the other times in the grid will be in 24- hour format. Moreover, to make it clearer to all users I could fix this slight problem in the future.
	I will enter the correct data format for the TransferTime: **10:23** **12:04** **18:12**	There should be no error label and the save transfer textbox should be displayed.		Yes, the test was successful, as the user could save the row data to the database table because the save textbox was displayed and the time was in valid 24 - hour format.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

				
	I will enter the incorrect data format for the ReasonForTransfer: “” **()** **Edfinslfd** **@@** **7**	There should be error label displayed, informing the user that the ReasonForTransfer is not an integer or the integer inputted>options available	   	Partially yes because the correct error message was displayed to the user when some of the incorrect data was inputted. However, when **7** was inputted into the textbox the error was not being displayed as it should have done because **7** is not an option in the transfer reasons key. I realised I had forgotten to add the validation for any integer that was not involved in the transfers reasons key. Therefore, after user testing, I will alter the validation so the users cannot enter any integer

				greater than 6, meeting the boundaries I set in my design.
	I will enter the correct formatted data for the ReasonForTransfer textbox: **1** **2** **6**	There should be no error label and the save transfer textbox should be displayed.		Yes, the test was successful, as the user could save the row data to the database table because the save textbox was displayed and ReasonsForTransfer inputs are all integers that are a maximum of 6.
	I will enter any formatted data in the Comments field: **Late Discharge after 6** **FD request** **Ggggggggghg** **Looooooool**	There should be no error label and the save transfer textbox should be displayed as there is no validation for the comments box because the user could write anything as extra notes.		Yes, the test was successful, as the user could save the row data to the database table because the save textbox was displayed and any letters or numbers could be inputted.

Points of improvement:

1. Add validation for the null input of username and password in the login screen
2. Add validation to the consultant ID in the grids to deny saving thee patient transfer from the daily or current month grid if erroneous data is present
3. Add validation to only allow up to 3 digits for the RoomID
4. Set input for ReasonForTransfer to only allow integers 6 or less

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Black Box Testing

Running through every screen

Patient Transfer Audit Walkthrough

Successful

At **0:00** you can see my testing the Login page, show/hide button, forgot login button and the login button.

At **00:17** you can see me testing the "patient data" button that takes the user from the main menu onto the year page.

At **00:21** you can see me testing the year button, "2023". I did not click the other years as they all take the user to the same month PD page.

At **00:23** you can see me testing the month button, "April." I did not click the other month buttons as they all take the user to the same Patient Transfer Database page

At **00:25** you can see me testing the "Daily" button

At **00:31** you can see me testing the "back" button on the daily database page

At **00:33** you can see me testing the "Current Month Database" button

At **00:37** you can see me testing the "back" button on the current month database page

At **00:39** you can see me testing the "back" button on the Patient Transfer Database

At **00:42** you can see me testing the "back" button on the Month PD page

At **00:44** you can see me testing the "back" button on the Year PD page

At **00:46** you can see me testing the "logout" button on the main menu

At **01:19** you can see me testing the "Audit Graphs" button on the main menu

At **01:21** you can see me testing the year button, "2023". I did not click the other years as they all take the user to the same month AG page.

At **01:23** you can see me testing the month button, "April." I did not click the other month buttons as they all take the user to the same Audit Graphs page

At **01:25** you can see me testing the "month" graph button, "April's graph."

At **01:30** you can see me testing the "View Graph Summary" button.

At **01:33** you can see me testing the “back” button on the view graph summary page

At **01:34** you can see me testing the “Consultant Graph” button on the Audit Graphs page

At **01:41** you can see me testing the “back” button on the Audit Graphs page

At **01:42** you can see me testing the “back” button on the month AG page

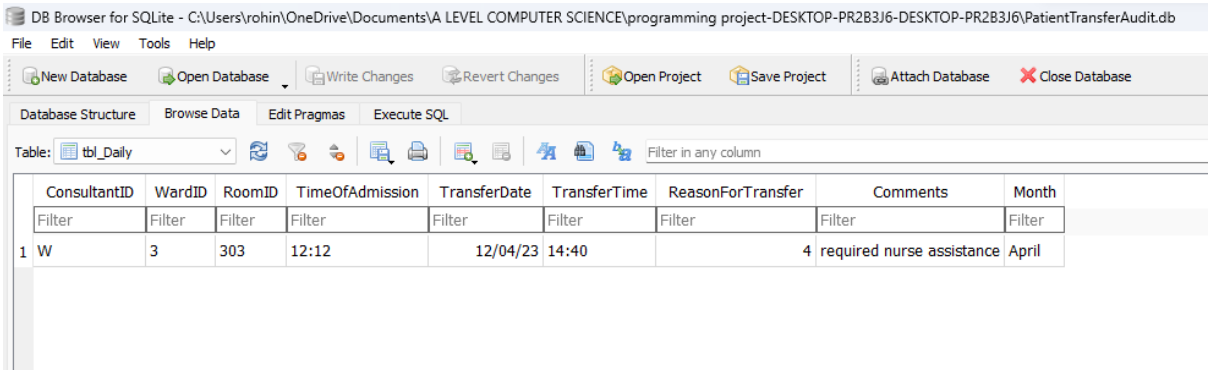
At **01:45** you can see me testing the “back” button on the year AG page

The following tasks have been carried out by (me) the developer:

Adding a patient transfer using the Daily grid to the database

01:51

The following added data from the grid can be shown in the database here:



	ConsultantID	WardID	RoomID	TimeOfAdmission	TransferDate	TransferTime	ReasonForTransfer	Comments	Month
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	W	3	303	12:12	12/04/23	14:40	4	required nurse assistance	April

Retrieving a patient transfer from the database

03:05

Transferring data from the Daily Grid to the Current Month Grid and the refreshing of the daily grid

03:28

Viewing the months data in the bar chart, viewing the consultant’s data in the bar graph and, viewing the graphs key trends

03:57

Adding grid rows to the Daily and Current Month grid

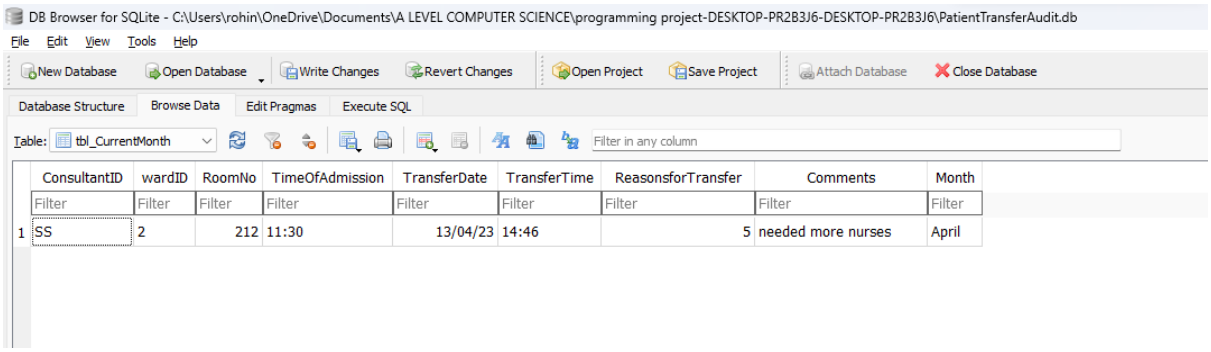
04:44

Adding a patient transfer using the Current Month grid to the corresponding database table

05:16

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

The following added data from the grid can be shown in the database here:



User testing

To note for the user testing, the **specific timings** for these different functions to take place were all set to **one time** so all the functions in my program could be thoroughly tested by multiple users.

Test	Who carried it out	Result	Was the test successful/user comments?
Check if the user can log into the software on their own	My client	The user was able to log into the software in 15 seconds with no help – given the developer(me) has provided the user with the fixed login credentials	The test was successful as my client has no issues gaining access to the software because of the clear and direct labels on the page, indicating what each button did.
Check if the user can add data to the daily table and save it to the database	A random nurse on my client's ward	They were able to add a patient transfer to the daily table in 45 seconds which proves that my program is quick and easy to use	It was successful as the staff member loved how quick they were able to get to the grid screen and how a warning was given to tell the user that all changes are final. However, they also suggested that when selecting the consultants ID/date/time, there should be a list to scroll and select from as it will be even quicker to add data. I could keep this in mind for the future of my program but I may not incorporate this update in as Appjar do not support list boxes within grids.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Check if the user can add patient transfers to the current month grid	A random nurse on my client's ward	He could add multiple patient transfers to the current month grid in 60 seconds showing my design is simple and clear to the user	The test had gone well. The nurse could add the patient transfers that were not for the current day to the month grid without a problem but he had some slight suggestions on how I could improve the certainty of the saved data. He suggested that as the save of the data to the database was so sudden, there should be another pop up which confirms to the user that it has been successfully saved to the database, making the save more explicit towards the user.
Check if the user can view the months graphs data on their own	My client	My client was able to view the months graph data in 50 seconds as they had a little trouble	It was successful and my client could view the months data in the bar graph the program created, although it took a little longer than I would have hoped because my client did not realise, they had to load the current month grid first so the data could be transferred into the graph. As the month button needs to be run first to load the data into the grid, my client was not fussed on fixing this slight issue, but I could look into improving it in the near future.
Check if the user can navigate through the program without any problems	A random nurse on my client's ward	The chosen nurse could move through the screens with ease	They said they liked how quick it was to move to different windows which is important when the nurses are in a rush and need to make the audit of a patient transfer smoothly.
Check if the user can add as many rows as they want to the grids	A random nurse on my client's ward	The nurse was able to add as many rows as possible in 5 seconds without crashing the program.	It was successful as the nurse could add as many rows as they want, without the program buffering or ruining the process of saving the patient transfer.
See if the user can save the bar graphs data to their files	My client	My client could comfortably save a screenshot of the bar graph date to their files	This was a success as user could click one button in the graph page and the files application would open in their computer, providing a direct and efficient way of sharing this data to other nurses on the ward.
Try clicking multiple buttons as fast as possible	The developer(me)	In the login, main menu, audit graph, patient data page and the patient transfer database pages when clicking multiple buttons, the expected output was carried out but for the **grids** when saving the data, the save info boxes were overlapping each	This test was mostly successful as many of the buttons worked as they should but in the daily and current month grid when the patient transfer was saved, the save info boxes were appearing on top of each other, allowing the program to look unprofessional and a potential cause of a program crash. Therefore, for future improvements, I should make sure only one info box pops up at any one given time.

		other if pressed too quickly.	
Check if user adds data to an empty daily table after end of month	My client	The client was able to add data to an empty daily table because it had refreshed as expected at the end of the month.	My client was happy that the daily table was able to refresh at the appropriate time of the month giving my program a systematic structure which is key within a busy hospital where schedules dominate the daily routine.
Try viewing other months and the present month data	The developer(me)	I was able to view the other months that were not currently in use and those tables were empty as expected but when I checked the present month data again the grid was empty too.	This was partly successful as the other months could be checked and their grids were empty, however when the present months data was checked, the data had disappeared. This is a small limitation as the user can only see data within the present month because all the data is stored under one grid(daily) and another grid (current month). Subsequently, the nurses can take a screenshot at the end of the month of both grids so that data is not lost. In addition, the data backup would be stored in the database anyway so if needed the information can be viewed from there directly.

General feedback

Beta Testing – User experience

Name	**Role**	**Problems Discovered**	**Additional Comments**	**Developer Response**
Charlotte Hamon	Day surgery nurse	When using the software, a couple times, I realised when I added data to the grids, if the integer added was not one of the options available in the reasons for	The reason for transfer and consultant key gives unique feeling to the program.	I previously discovered the same thing during white box testing. As you have spotted this error too, I could alter the validation function for these grids to set a range of

		transfer key, the data save would still occur.		integers that can be added to the field.
Ash Sooks	Day surgery nurse	I tried logging in for the first time and clicked the enter button by accident when nothing was present in either username or password textbox. When I did this, the username error came up but for accuracy I would like to see a different error box, identifying that the fields are empty.	I liked the colour scheme of the login page; it was very inviting.	I will add an extra validation message to the ValidLogin() function inform the user that the username or password field is empty.
David Cooper	Charge nurse	The buttons were clear and the layout was simple but I think there should be brighter colours used as personally, the system looks plain and dull.	I love how easy it is to add a patient transfer.	This is a valid point, but I will not be changing the colour scheme of the program as my client wants the user interface to match the hospital colours to keep its professional look.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Issues fixed as a result of Testing

1. Added validation for the null input of username or password in the login screen

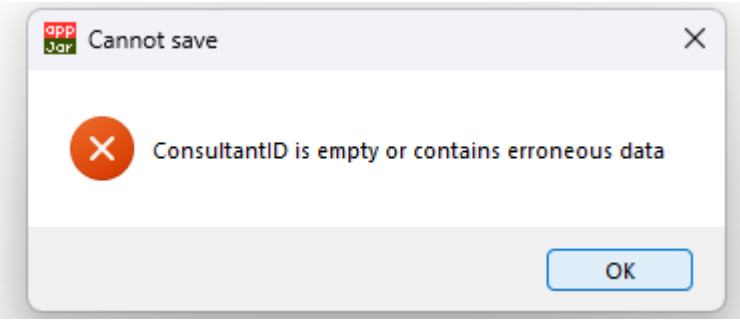
The code below, is the validation I implemented to satisfy the users who tested my program, to display more accurate errors to the users.

```
#Checks username or password field is empty
    if len(username)== 0 or (len(hidden_password)== 0 or len(visible_password)== 0):
        app.errorBox("Access denied","Username or Password not present")
```

2. Add validation to the consultant ID in the grids to deny saving patient transfer if erroneous data is present such as "@" or "#"

I made it so the consultant ID field does not allow any symbols which are entered into the grids.

```
#isalnum = alphabet and number

    if  consultant_id == ""  or (not consultant_id.isalnum()):
        app.errorBox("Cannot save","ConsultantID is empty or contains erroneous data")#print error message if ConsultantID cell is empty or contains a symbol
        return False#sets subroutine to false
```



3. From the results of white box testing, I am going to add validation to the RoomID where only up to 3 digits is allowed

I added a limit to thenumber of digits the user can add to the RoomID field because as of now there are no rooms in the hospital which have 4 digits.

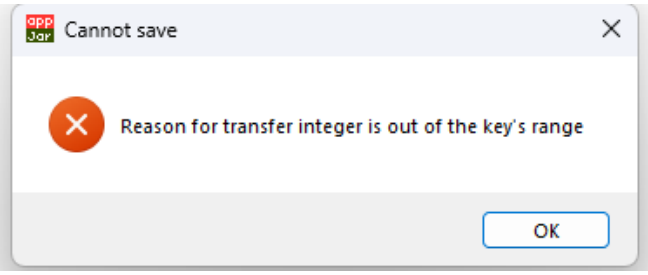
```
elif Room_ID == "" or (not Room_ID.isdigit()) or len(Room_ID)>3:#print error message if RoomID cell is empty or not an integer or there is more than 3 integers
    app.errorBox("Cannot save","Room ID is not a number")
    return False#sets subroutine to false
```

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

4. Set user input for the ReasonForTransfer field to only allow the integers 1-6

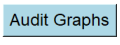
After feedback from the users who tested my program, I set limits to the user's input to a range of integers that corresponds with the available reasons for transfer.

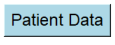
```
elif int(reason_for_transfer)<1 or int(reason_for_transfer)>6 :#print error message if ReasonForTransfer cell value is not between 1 and 6
    app.errorBox("Cannot save","Reason for transfer integer is out of the key's range")
    return False#sets subroutine to false
```



I also quickly added a piece of code to each title on each page so the style remains consistent throughout the program.

```
app.getLabelWidget("title2").config(font=("","20", "bold","underline"))#sets font size and makes title bold and underlined
```





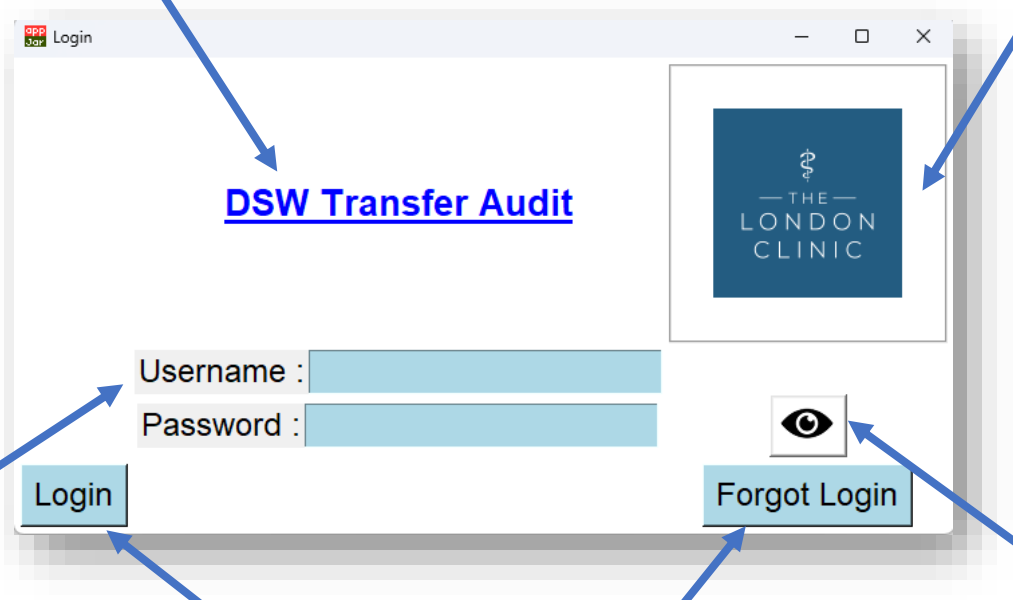
Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Screen Evaluation

Login Screen

Here is my login screen title, which identifies what this program is about. I have made it bold and a slightly darker blue so this is noticed first by the user. It is underlined and at the top of the page to give the screen a more professional look, while fitting in with the colour scheme of the hospital.

This is the hospital logo which I have place in the top right of the screen. It makes the screen more colourful and representative of The London Clinic. It also gives the user more certainty that they are using an official program.



I have created these entry boxes for the user to input their username and password. I made the entry box background colour light-blue to match the colour scheme and to contrast the bright white background. This makes the black text clearer when the user is typing something. It also meets the requirement I had set in my success criteria to present the clinic's colours.

These are two buttons that are very important in the login screen. The "Login" button allows the user to login and the details are checked by validation to deny any unauthorised access. This fulfils the requirement for a valid login as outlined in my success criteria. The "Forgot Login" buttons provides an info box with more information to user of how they can contact the developer for help regarding their login details. Both buttons have a light-blue background matching the colours of the hospital, which is maintained throughout the different screens.

This is the reveal hidden password button converts the hidden password, inputted by the user, into plaintext (vice versa) so the user can see what they are typing, which can help identify any mistake they may have made. I would have liked this image button to be next to the password entry box (I could not add the image button directly next to the input box in AppJar) as it would have made this screen look tidier. Therefore, I am somewhat happy about this button as the eye image gives a clear and precise indication to its function but it is not positioned in the way I wanted.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Why was this screen Necessary?

This screen was needed so the users can enter their username and password and only authorised users can access the program, preventing others viewing personal patient data. This is very important when working in a busy environment and there are many patients that could access a staff computer while the nurses are distracted.

Have I ensured robustness and security of this page?

Yes, the entry boxes each have validation which checks for the length of the inputs and accuracy to the details stored in the database so not just anyone can gain access to the software. This is very important when working in a busy environment and there are many patients that could access a staff computer while the nurses are distracted.

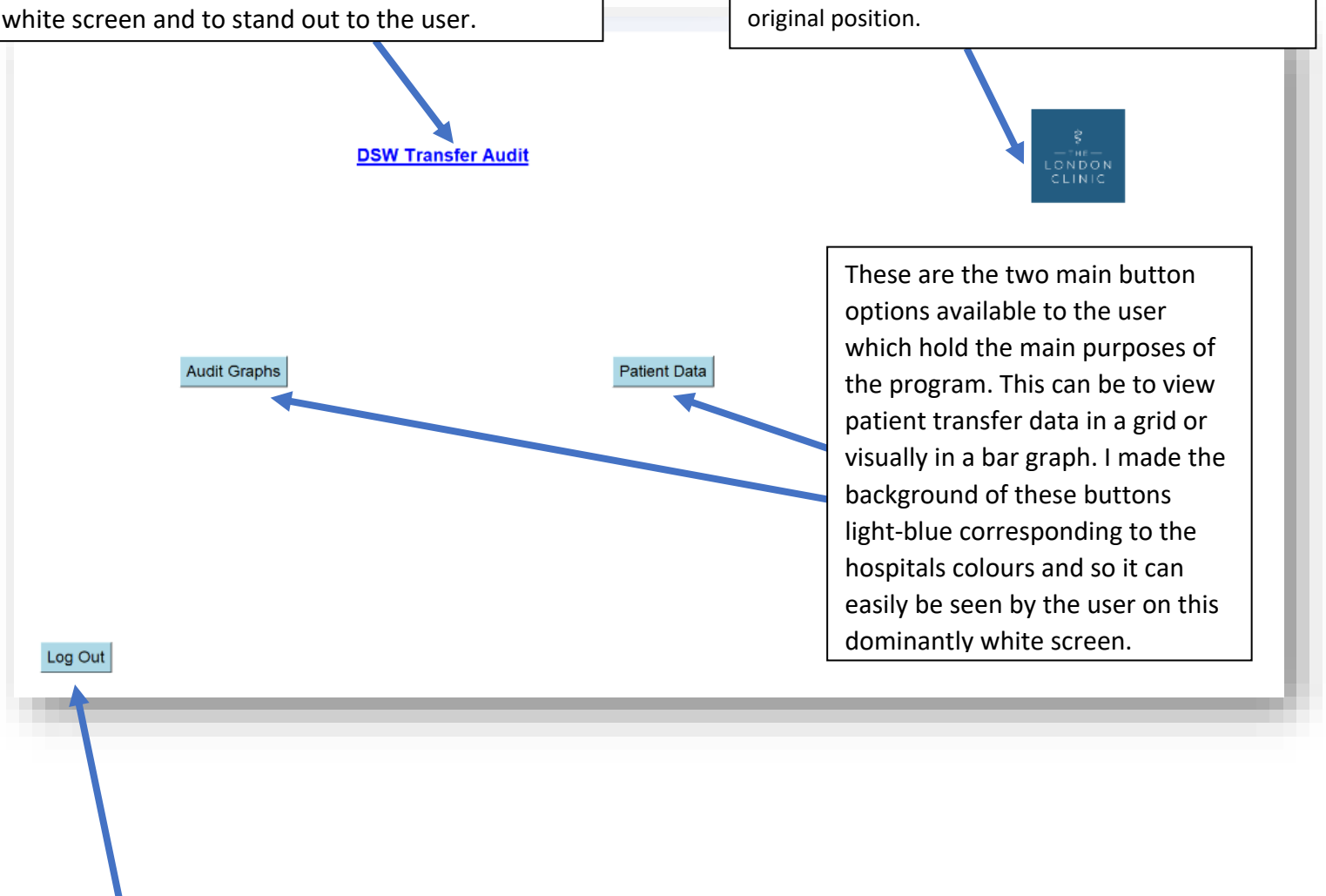
Final evaluation

Overall, I am pleased with this screen as it meets its purpose and it is quite secure from unauthorised access. The layout is clear and the same as the screen I made in my design which is something I hoped for. Therefore, I think this screen has been a success.

Main Menu

This is the title of the program I have placed on the page as it is the Main Menu where the two main functions are displayed. I have again made the title a darker blue, bold, and underlined to contrast the white screen and to stand out to the user.

Here, maintaining its position, is the hospital logo I have added to the top right of the page. This further represents the program as The London Clinic. I was going to place the logo right in the corner of the screen but it looked too distant from the other information on the screen. Therefore, I left it in its original position.



These are the two main button options available to the user which hold the main purposes of the program. This can be to view patient transfer data in a grid or visually in a bar graph. I made the background of these buttons light-blue corresponding to the hospitals colours and so it can easily be seen by the user on this dominantly white screen.

This is the "Log Out" buttons which allows the user to leave the program if they would like to. I also made the colour of this button match the colour scheme of the rest of the program. This button is also out of the way in the bottom corner, as it is not as important as the other buttons being displayed.

Why was this screen Necessary?

It was required to display to the user the main options of this program so they could find what they were looking for effortlessly.

Have I ensured robustness and security of this page?

As all the only features of this screen is buttons, I did not need any validation. These buttons will direct the user to the proceeding screens and robustness has been ensured by hiding the current and showing the next sub window when any of the two main buttons are pressed.

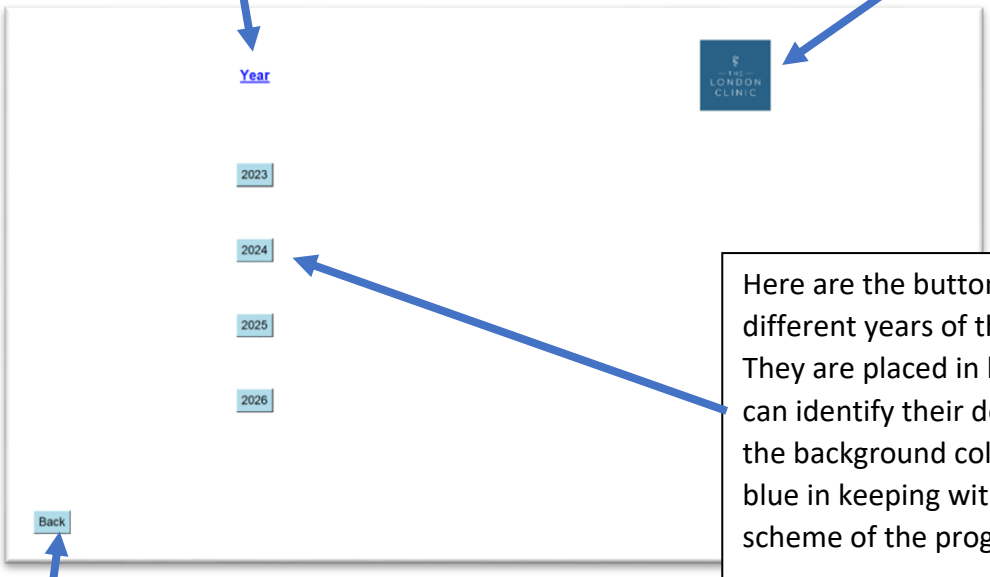
Final evaluation

In conclusion, this page has met my expectations by presenting a simplistic approach towards its objective without making things too complicated for the user, which could improve user experience.

This is the title of the page informing the user what this page is about and I have made it a darker blue to again stand out to the user.

Year Page

This is the hospital logo seen again as it is maintained throughout the program. It shows the user that they are using a professional program, as well as making the screen look more exciting.



Here are the buttons that take the user to the different years of the patient transfer grids. They are placed in logical order so the user can identify their desired year quickly. I made the background colour of these buttons light-blue in keeping with the overall colour scheme of the program.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

This is the "Back" button which takes the user back to the main menu. It is also light-blue to match the colours of the other buttons and to stand out against the white background.

Why was this screen Necessary?

This screen was required to take the user to view the different years of grids with patient data and this is called **YearPD**. There is another page identical to this screen which does the same thing but for the Audit graphs called **YearAG**. I needed two separate screens because the following functions that are carried out are different depending on which option the user picks in the main menu.

Have I ensured robustness and security of this page?

This page only contains buttons that direct the user to different sub-windows so no validation was necessary. However, robustness is shown, by showing and hiding the different sub-windows when a button is clicked.

Final evaluation

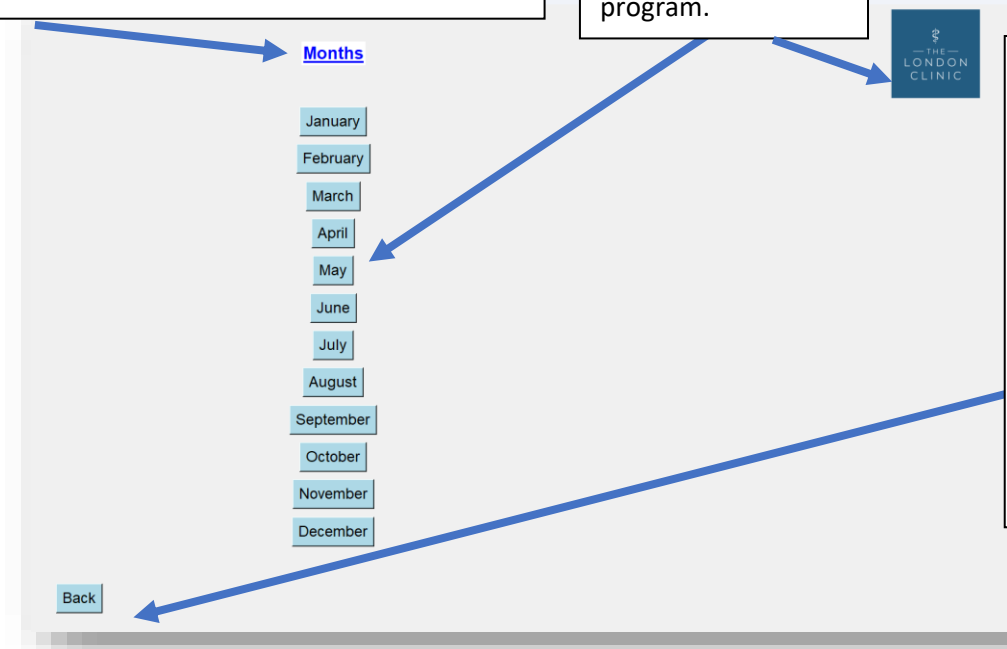
Overall, I am satisfied with the outcome of this page as it meets my success criteria as the buttons are "logically displayed on the screen." It carries out its intended purpose with efficiency as it is simple and easy to read.

Month Page

This is the title of the screen which informs the user it is the page containing all the month options. It has the same style has the other titles - dark, underlined, and bold. This stands out to the user and makes the screen look more presentable.

This is the hospital logo which is seen is most of the other screens matching the theme of the program.

These are the different month buttons that direct the user to the Patient Transfer Database page. I made all the buttons background colours light-blue which again matches the colours of the hospital.



Here lies the "Back" button which directs the user back to the year selection page. The background colour is also light-blue fitting in with the colour scheme. I also placed this button in the bottom left of the screen, which is maintained throughout the rest of the screens, as it less purposeful than the main month buttons.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Why was this screen Necessary?

This screen was necessary to direct the user to the Patient Transfer Database page that held the corresponding month grids. The page in the screenshot is called **MonthPD** and there is another identical page in the program called **MonthAG** that carries out the same job but for the Audit Graphs. Again, these two separate month selection pages are needed because the following functions that are carried out are different depending on which option the user picks in the main menu. For example, if the user picks “patient data” and then “January” the data needs to be retrieved for that month but not display the bar graph.

Have I ensured robustness and security of this page?

In this screen, there is only buttons so no validation was needed. I ensured robustness by making sure sub-windows would hide and show when the button is clicked. This allows the user to move through the screens as fast as possible as many times as they want.

Final evaluation

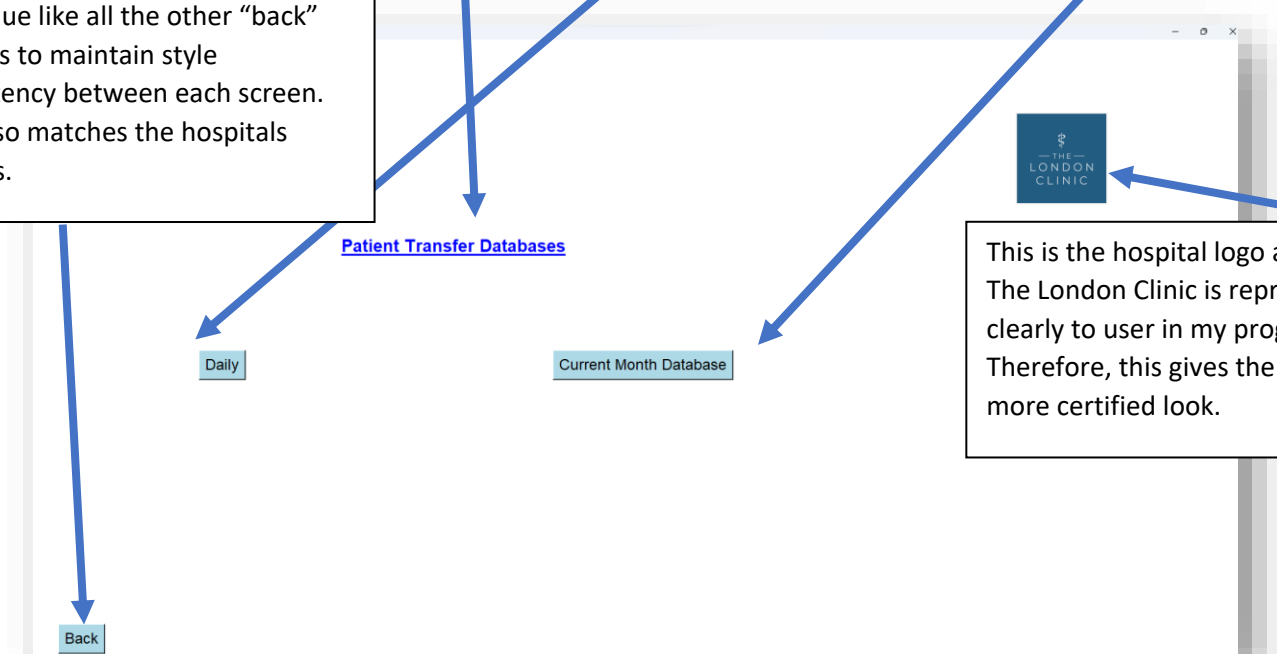
Overall, I am happy with this page as the buttons are clear and displayed logically. This screen also meets the requirement where pages in the program need to be displayed in a sensible time, improving the efficiency of this software.

This is the title of this page. It informs the user that they can display either the Daily grid or the current month grid that contains all the patient transfers for the month. I made the title dark blue, underlined, and bold to match the other titles, showing consistency within my program.

Patient Transfer Database Page

These are the two main grids within my program. The “Daily” button takes the user to a grid containing patient transfers from the current day. The “Current Month Database” button takes the user to the grid containing patient transfers from the whole month. I made these two buttons’ backgrounds light-blue fitting in with the colour scheme of the hospital style.

Lastly, this is the “back” button which returns the user to the month selection page. This button is also light-blue like all the other “back” buttons to maintain style consistency between each screen. This also matches the hospitals colours.



This is the hospital logo again where The London Clinic is represented clearly to user in my program. Therefore, this gives the screens a more certified look.

Why was this screen Necessary?

This screen was required for the user to gain access to either grids so they can view or add patient transfers. These options on this screen determines what important functions are executed and displayed in the program.

Have I ensured robustness and security of this page?

There is no validation here as there are only buttons on the page. However, when the user switches between screens I had made sure the corresponding sub-windows show and hide, providing effortless scrolling between screens.

Final evaluation

As a whole, I am pleased with how this page turned out as it the buttons are far apart and black text on the buttons stands out against the white background making it clear for the user to see the options displayed.

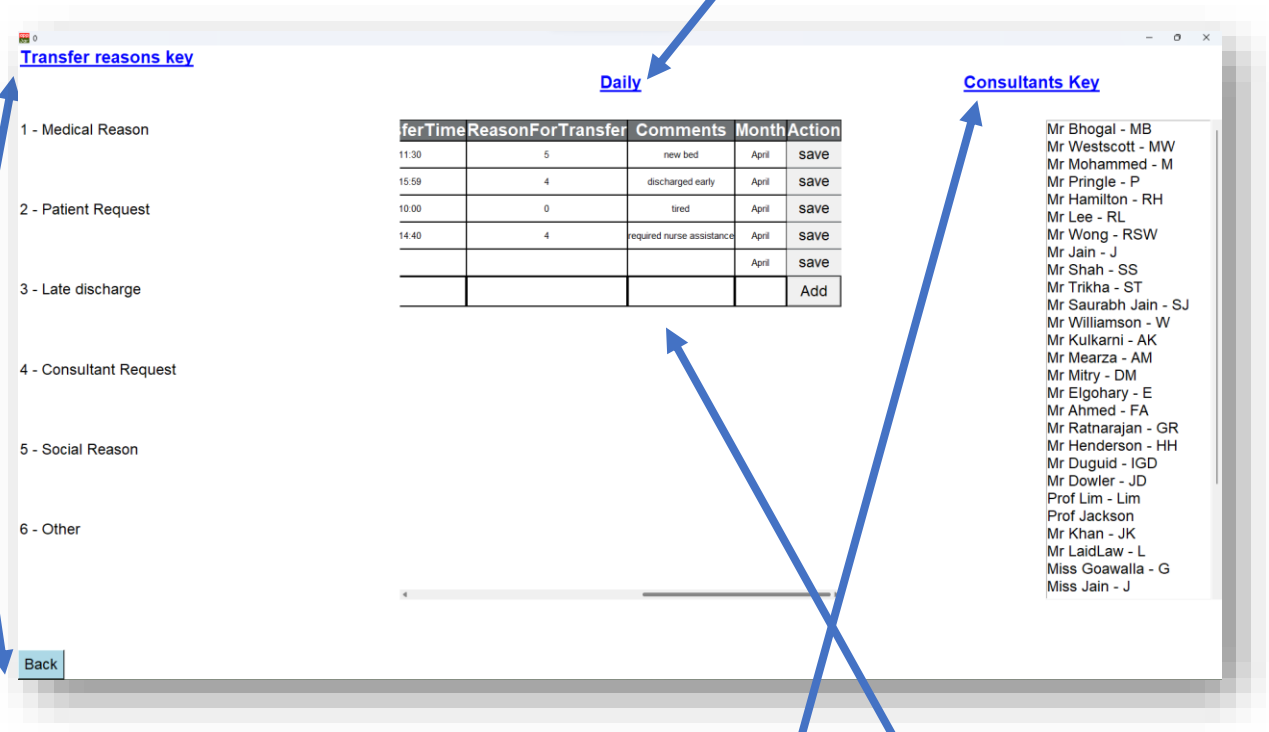
Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

The following screen is the “Daily” grid. I will only annotate this screen as the “Current Month Database” button brings up an **identical** screen with the title “Current Month” and containing data for the whole month.

This is the “back” button which redirects the user to the Patient Transfer Database page. I made the button background light-blue matching with all the other buttons throughout my program, showing off the colours of the hospital.

Here is the title of the screen “Daily” which suggests to the user this grid is for the current day. I made the title fit the style of the whole program.

Daily Grid Page



This the Transfers reason key which a fixed key matching up 6 options with 6 reasons of patient transfer. It follows a logical order making it easy for the user to identify the different reasons. I made the title of the key dark blue, underlined, and bold to match the rest of the titles in my program. This also makes they key more noticeable to the user as it is important when entering data into the grid.

At this side on the screen, is the Consultants Key where each consultant currently attending the clinic have their own initials as the inputs for the grid. As there are lots of consultants, I made a list box so the nurses can scroll down the list to look for their desired consultant. I made the title of this list box dark blue, bold, and underlined as it stands out prominently to the user because this information is vital when filling the grid.

In the centre is the grid that allows patient transfers to be added to the database. I originally wanted the grid to fill the whole page so every field could be visible. However, the grid was too big so I had to place the grid in the middle decreasing the width so the other **keys** could be displayed either side of the screen. The grid contains a “save” button which saves the current row to the database and an “Add” button which adds a new row to the table.

Why was this screen Necessary?

This screen is highly necessary as it carries out one of the important functions within the whole program. It provides the essential foundations for saving patient transfers to the database. It also allows user to edit/change patient transfers with easy, simplifying the audit process.

Have I ensured robustness and security of this page?

Yes, this screen included validation on the ConsultantID, WardID,RoomID,TimeOfAdmission,TransferDate,TransferTime and ReasonForTransfer fields in the grid. The ConsutlantID must identify erroneous data as invalid. The wardID and RoomID both must contain integer inputs. TimeOfAdmission and TransferTime should both contain data in readable HH:MM format. The TransferDate field should be in DD/MM/YY form and the ReasonsForTransfer column should only allow integers from 1-6, otherwise an error should occur. All this entered data would remain if the user clicks the “back” button so the user can switch between screens without having to re-enter data. If the row inputs were not saved and the program was closed, the row would be cleared preventing any confusion between nurses due to unfinished patient transfers. These validations are crucial for nurses to be able to view readable patient transfer rows that can easily be distinguished.

Final evaluation

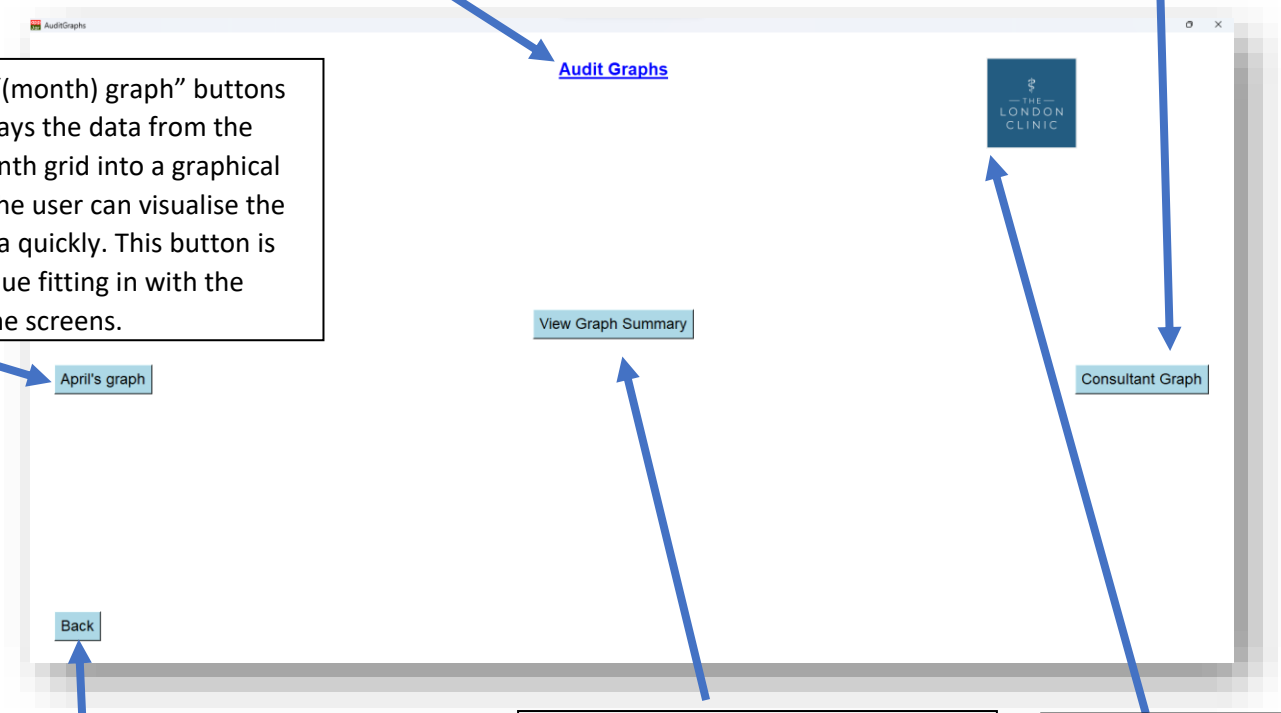
Overall, I am satisfied with the outcome of the screen as it carries out its role of allowing the addition of patient transfers. It clearly shows the user how to use the different keys in the table and presents data in a professional way. Despite moving the grid to the centre of the screen it is still appropriate for any nurse to use at any given time.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

This is the other main page (Audit Graphs) that shows the user 3 options they could pick. I made the title dark blue, bold, and underlined just like the other titles in my program.

Here is the "Consultant Graph" button. This contains the number of transfers for each consultant during the month as presents the data in a bar graph for the nurses to analyse. I made this button have a light-blue background representing the colours of the hospital and theme of the program.

Audit Graphs



This is the "(month) graph" buttons which displays the data from the current month grid into a graphical format so the user can visualise the written data quickly. This button is also light-blue fitting in with the theme of the screens.

The "View Graph Summary" button allows the user to view the key data/trends in each other bar graphs in words format. I made the button background light-blue too matching the colours of The London Clinic.

Here is the logo for the hospital, giving the program a professional and unique look, which maintained throughout the whole program.

Here lies the "back" button in the bottom corner, to direct the user back to the month selection page. The "back" button background is also light-blue contrasting the white background making the button text easier to see.

Why was this screen Necessary?

This screen is acts as a pathway to guide the user to observe patient transfers in graph format or view the key trends/anomalies.

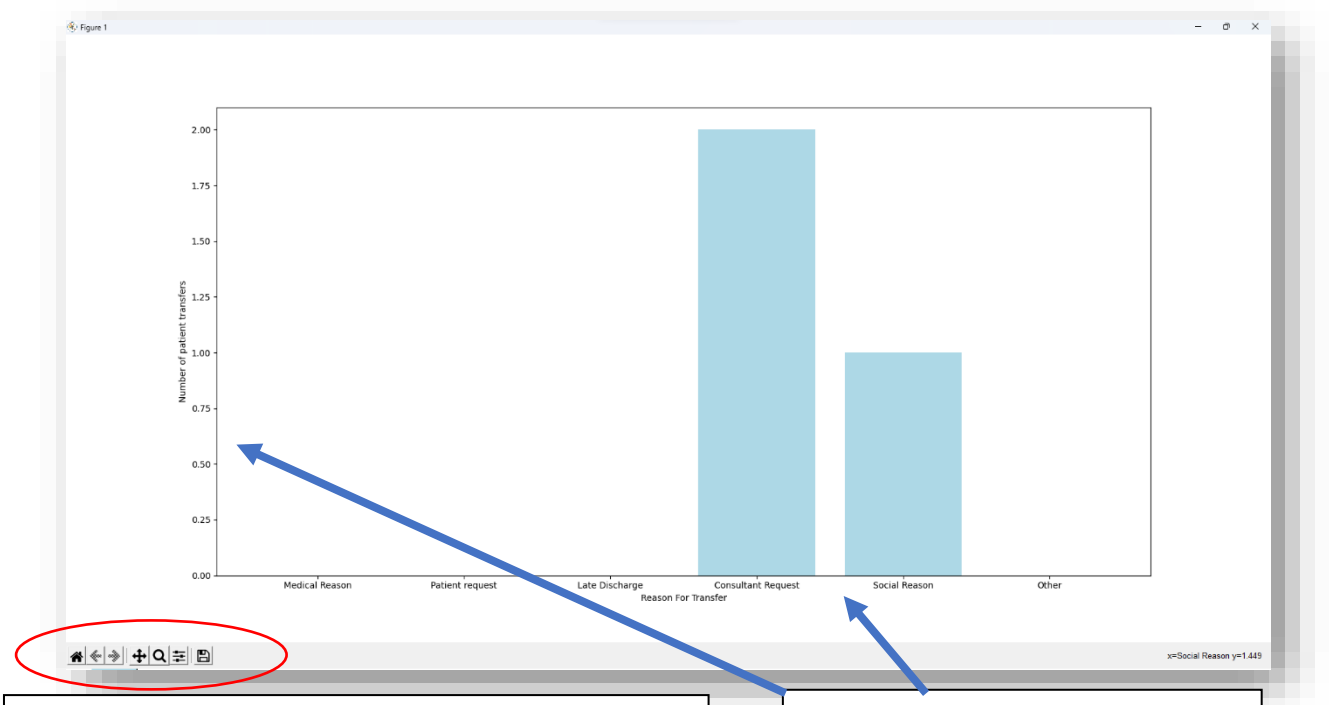
Have I ensured robustness and security of this page?

No validation was essential on this screen as all the features are buttons that direct the user to different screens. However, I made sure the software is robust by showing and hiding the corresponding screens so the user can focus on viewing one screen at a time. The user can also move back and forth through the pages as much as they want without affecting the software's performance.

Final evaluation

Overall, I am somewhat happy about this page because at first, I wanted these buttons to be vertically placed in my designs. As now I have an extra button, "Consultant graph" I had to place the buttons horizontally. I would have liked the 3 buttons in a straight line but because I used the **stickiness** function in AppJar, the middle button is slight further up the page than the others. As this is a minor concern, I do not need to worry because the rest of the page is performing and displaying as anticipated.

This is the result of the "(month) graph" button. This displays the bar graph data for the specific month using the library **matplotlib**. Therefore, there is not title to this page.



All these small buttons here are part of the **matplotlib** library. They can carry out many jobs such as saving a copy of the graph to the user's files, increasing, or decreasing the bar graph in size and change the width and height of the bar graph to allow for users to view the graph in many ways to see different perspectives.

The bar chart has an x-axis of the different reasons of transfer and a y-axis for the number of patient transfers. My actual bars are light-blue matching the colours of the hospital while also making the graph look less boring.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Why was this screen Necessary?

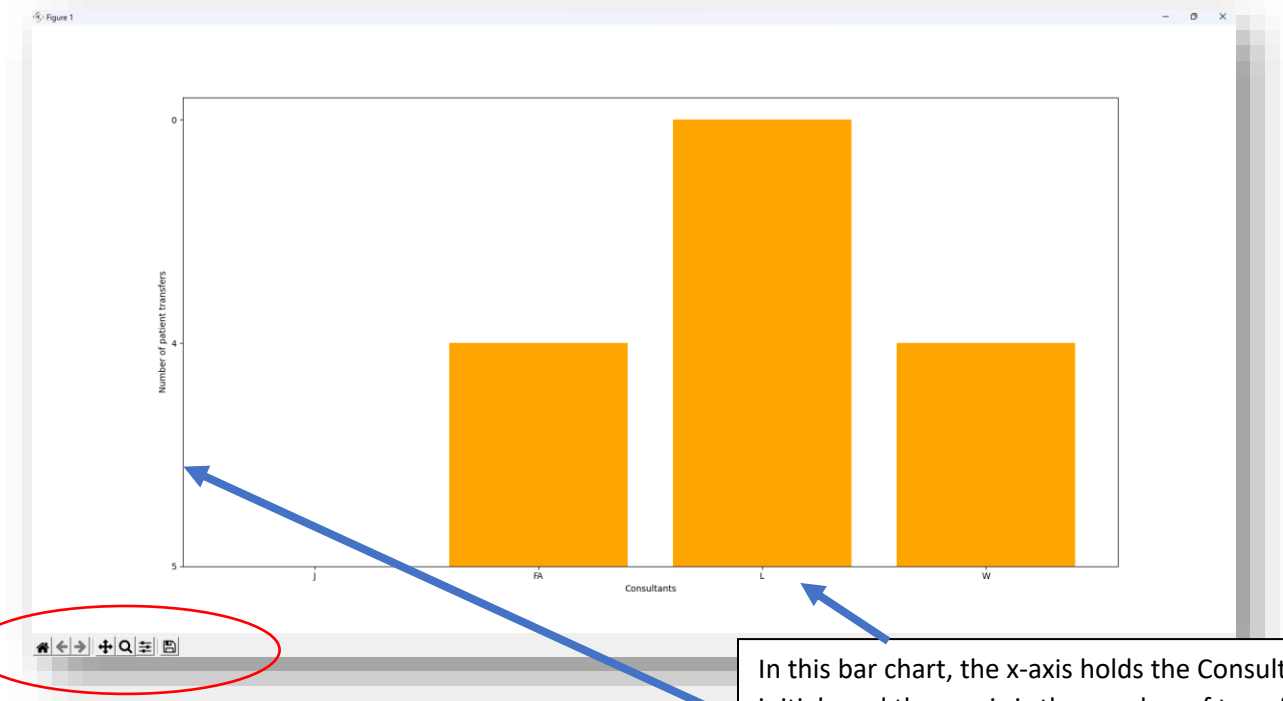
This screen was highly significant in displaying the written patient transfers visually on a graphical scale, giving user a different way of reading the data for the month.

Have I ensured robustness and security of this page?

Validation of this page was not required as it was only displaying pre-determined patient transfers from the current month grid and there were no inputs for the user to enter.

Final evaluation

Overall, I think this page has turned out fantastically as it accurately represents the inputted grid data and the graph size can be changed to be seen clearer by any user. However, during my designs I wanted the consultant's information on the same (month) graph so data will all be in one place. I soon realised this could not be done as I could not add the "consultant key" to the (month) **matplotlib** bar graph. With this knowledge, I created a separate bar graph called "Consultants graph." This allowed me to still compare the number of transfers with each consultant and make the data easier to view because there were 2 graphs instead of 1 complicated graph with all the data in. This simplifies my program and allows the users to view either graph finding specific information that can distinctively be seen.



All these small buttons here are part of the **matplotlib** library. They can carry out many jobs as previously mentioned.

In this bar chart, the x-axis holds the Consultant's initials and the y-axis is the number of transfers. The bars are orange for the user to differentiate from the (month) graph and it also stands out to the user making the data clearer to see.

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Why was this screen Necessary?

This screen provides the user with visual information of the consultants and how many patients they have transferred under their name within the month. This can supply user with evidence to speak to consultants regarding their patients and why they transferred that many, helping identify consultants going against typical procedures.

Have I ensured robustness and security of this page?

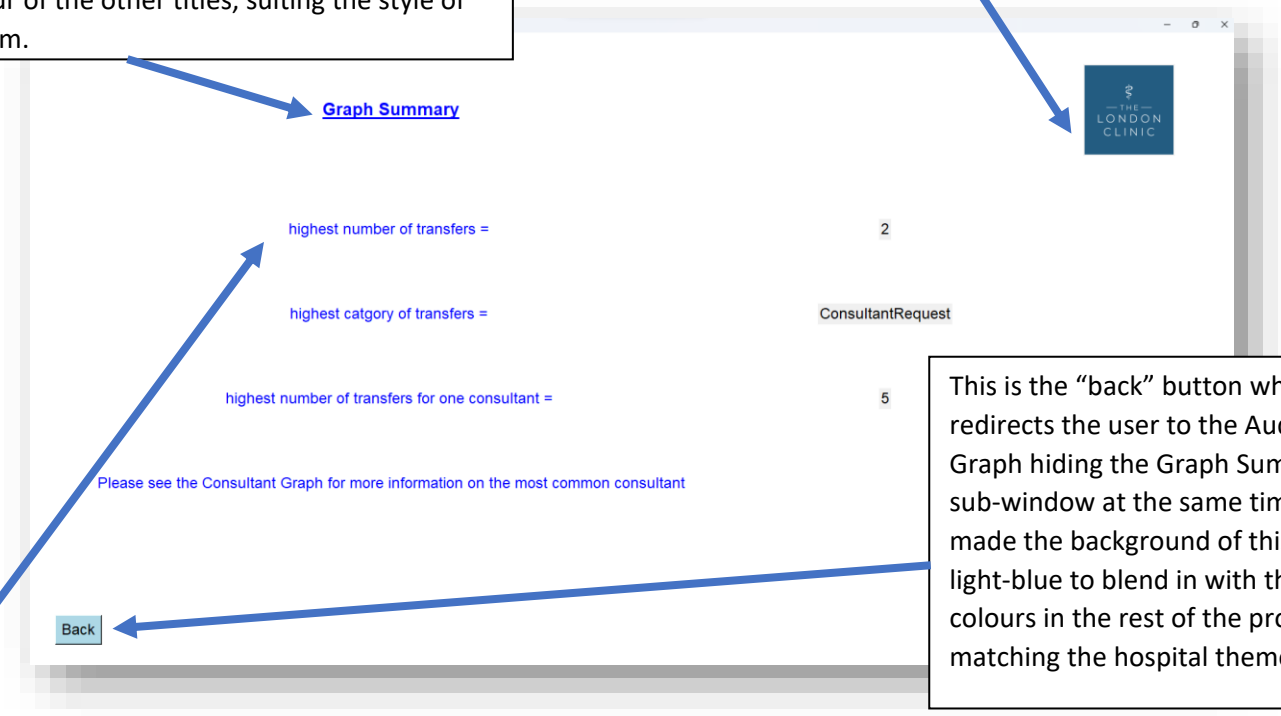
No, validation was not required on this page as it is only displaying the grid data to the user so they can observe the information they need.

Final evaluation

Overall, I think the bar graph displays the consultant's data truthfully as there is no missing data and it is very visible to all users with the bright orange bars being used. Therefore, I think it fulfils its intended use of displaying patient transfers graphically, for a more of an insight of what the data looks like.

Here is the title of the page, that informs the user the screen is a summary of the bar graphs. I have made the title dark blue, underlined, and bold so it draws attention to the user. It also matches all the colour of the other titles, suiting the style of my system.

The hospital logo is shown again in the top right like most of the other screens informing the user they are utilizing a professional and trusted piece of software.



This is the "back" button which redirects the user to the Audit Graph hiding the Graph Summary sub-window at the same time. I made the background of this button light-blue to blend in with the colours in the rest of the program, matching the hospital theme.

This is the main trends and key data that the user would like to know according to my client. Each of these "bar graph highlights" are a dark blue colour to inform the user that this is important information and to match the colour theme of the program. The last point advises the user to look at the consultant graph to check the consultant's name of the most common consultant. Furthermore, the user could use this information as evidence when questioning that specific consultant.

Why was this screen Necessary?

This part of the program is essential in the purpose of my program. It shows the user the trends of the different bar graphs and identifies key data that the user may need. This can be used to prevent day-case patients blocking impatient beds and to talk with the consultants who are not following the correct policies.

Have I ensured robustness and security of this page?

There is no validation being used on this page as there are only buttons and labels being displayed. Robustness has been ensured by showing and hiding the correct sub-windows when required, speeding up movability between different screens in my program.

Final evaluation

Overall, I think this page is very useful as it provides the user with the desired information for the hospital ward and helps deal with consultants who are not following key rules in the clinic. I am also really pleased on how this screen turned out as it matches exactly to what I had envisioned in my design.

Evaluation of my finished code against the success criteria:

Number	Specification point	Has the criteria been met?	Client Opinion
1	My program should allow a user to login with the correct credentials.	Yes – there is a functional login page that works for all users.	"I am happy with security of the program as it doesn't allow unauthorised access."
2	The daycase to overnight patient data present in the database should be represented on a bar chart.	Yes – there are two bar graphs in my program, one presenting the consultant data and one for the patient transfer data. These two graphs are created at the end of the month, every month, accurately exhibiting the audit in real time.	"It provides me the data I require in a bar chart format. However, in the future data could be represented in other means such as a pie charts, line graphs etc."
3	Information must be logically displayed on each page	Yes – Data on each page like the month and year selection	"The displayed widgets are in an

		page is displayed in logical order	order which are easy to follow."
4	My program must be able to display pages in a sensible time.	Partly Yes – Most of the pages bring up the desired sub-window in a reasonable time. However, when it is the end of the day at the time of transfer, it takes slightly longer to load the grids as all the daily grid data is being transferred to the current month grid at once.	"The system allows me to swich between pages as quick as I would like but it takes slightly longer when lots of data needed to be loaded at the same time. In the future, I think you should add a loading screen so if the data is going to take some time to be displayed, the program will flow better."
5	Data entered into the database should be valid and stored in the correct format.	Yes – When the user enters data into the grids, the necessary fields are checked for validation so the auditing of patient transfers are appropriate and can be read by any user that has access to the system.	"It is programmed well because when data is not entered under a field, the program prompts the user to inform them there is missing or incorrect data. This will make the user re-think before saving. Although, in the future I think you should try adding list boxes for some of the fields in the grids as it will save time for us when entering data."
6	The colour scheme should be representable of The London Clinic: light/dark blue and white.	Yes – My program accurately represents the hospitals colour scheme. Every screen in my software makes somewhat use of these colours. All	"It accurately corresponds to the branding of The London Clinic."

		the screens have a white backgrounds with a dark blue underlined title. Each button is light-blue keeping within the style desired by my client.	
7	My program should be able to run on a standard desktop/laptop in the clinic.	Yes – the program can run very quickly using my laptop which has 100GB SSD as main memory and a 2.2GHz CPU. Therefore, the system will run seamlessly as desktops at my client's hospital are i7s with windows 10 that have a 1TB HDD.	"It is able to run efficiently on the desktops at The London Clinic with no problem."
8	Daily database for each day should refresh every day to allow for new data.	Yes – Any time between 7pm-8pm the daily grid will transfer all its data to the current month grid, refreshing itself at the same time.	"The grids refresh as required which allows the user to enter data for the coming day."
9	The program colour scheme should be same as the OS theme used.	No – This has not been met, as my client decided to go against incorporating the colour scheme of the operating system as she suggested the hospital theme colours are more suitable for her clinic.	"It looks as though as you have listened to what I have requested, which was the cooperate branding colours for The London Clinic. Therefore, I am not concerned about this success criteria not being met ."
10	Data can be seen/altered at any time	Yes – The data can be altered at any time after the grid	"I am happy because the grid data can be

		has been created and if the user wanted to add a patient transfer for the previous day, they could just add the transfer to the current month grid. This is because all the data for the current month will end up in that grid.	easily corrected when required."
11	Make sure I have an updated database	Partly Yes - The grid's up to date data is saved to the database each time the user clicks the "save" button in the corresponding row. Although, I previously mentioned that there was going to be a read only mode for the grids but I did not implement this feature. I went against this feature because I was later informed by my client that the program will only be accessed on one main computer, one user at a time, making this feature pointless.	"The software clearly shows any amendments made to the grids so any user can view the data most recently added. As I have mentioned before there is going to be only one desktop being used to run this program. Therefore, I agree that the read only mode is not needed as the data will be updated by one user at a time."

Name: Rohin Heer **Candidate Number: 5616** **Centre Number:12520**

Successes

Screen	What was successful?	Why was it successful?	How could I improve this in future developments?
Login Screen	The screen provided the user with a way to enter the software with correct authorisation from the program. This prevented unwanted users accessing the program.	There were 2 textboxes on the screen, clearly labelled. I ensured only users whose inputs matched to the credentials stored in the database were allowed access to the system. Validation of these entries was put in place to make sure data was in its correct format.	In the future I could change the function of the "forgot login" button to allow the user to reset their password. This will permit users to create their own new password so they can quickly access the system instead of contacting the developer to assist them if they forgot their details.
Main Menu	Users can move swiftly to the other main functions of the program.	I made 2 buttons in the centre of the page making it clear and easy to use for any user.	Moving forward, I could make this screen look livelier by, adding more colours or images that help support the branding of the hospital.
Year Selection Page	The screen logically lists the years in order allowing the user to view the month selection page and the data the specific year chosen.	I listed all the years from 2023-2026 vertically down the screen making it simple for user to select the button for their desired year.	In upcoming updates, I could add a list box so the user can select any future year and bring up the data for that year which save me (the developer) time when adding new buttons for the coming years after 2026.
Month Selection Page	The screen also logically lists the months of the year in order allowing user to view data only for the specific month chosen.	I arranged the months of the year vertically down the screen facilitating the user's selection of the corresponding button, to access the grids displaying data for that month.	In the future, I could again add a list box for the user to select their desired month as it will be quicker and more efficient to view data.
Patient Transfer Databases	This screen provides the options for the user to view the different types of grids holding the data. The user can either view the data from the current day or for the whole current month	The buttons on the page clearly stated what each one did. These buttons were place in the centre of the page just like the main menu to maintain some style in the program.	In the future, I could add a lading screen feature to either button so when it is clicked, the data is only displayed once fully loaded into the grid. This will help solve the issue of the program responding to the user in a sensible time, improving staff satisfaction.
Daily Grid/ Current Month Grid screen	These screens carry out the same job but one for daily and one for the whole months data. It was successful as it allows the user to	The grids make it easy to add rows and save data for each row. There is validation used where necessary to make sure the user is adding appropriate data that everyone can read. I	In due course, I could adjust the grids by finding a way to add list boxes as my client suggested. Therefore, selecting ConsultantIDs,WardIDs,RoomIDs and ReasonsForTraansfer would

	add patient transfers to the database which is one of the main purposes of this software.	used keys for the ConsultantIDs and the different ReasonForTransfer options. These keys help guide the user in how to fill the grid to match the expected format, showing my program is very informative and useful to first time users.	be much easier saving important time for the user and so they can tend to the need of patients which is a higher priority.
Audit Graphs	This screen allows users to select any of the 3 options, "(month name) graph", "view graph summary" and" Consultant's graph". This can view the patient transfers graphically so the user can see the data from another perspective and they can identify can key trends they pick up on.	The screen clearly presents all the options horizontally for user to see clearly and select the option they need.	Going forward, I may add some preview information that informs the user what the next sub-windows are going to show such as the what each colour represents on the graph and what the **matplotlib(library)**buttons on the graph pages do to the bar charts.
Month/Consultant Graph	Here, the patient transfers and the consultant information are shown on a bar chart as my client requested. This helps the users to view any key comparisons they missed in the patent transfer grids. This also aids data similarities and differences between different months.	Data represented clearly on a bar chart helped Identify which consultants were not following procedures. The graphs could also be modified by the **matplotlib(library)** buttons which could save the graph as a picture to the user's files or increase/decrease the size of the graph, again supporting in the findings of key resemblances between graphs.	In the future, I could implement these two separate bar charts onto one graph as the user wouldn't have to switch between the two-saving time. I could also add a feature where a backup picture of the graph is sent to the user's email, instead of the user having to save a backup directly to their personal files, improving the professionalism within my program.
Graph Summary	In this screen, an overview of both of the bar chart's are displayed, identifying the key trends and data to look out for. This provides the user with the most important information especially when there is a lot of	The highest number of transfers, common consultants etc, are calculated and presented in a report format down the page. Each point is distinctly displayed with the corresponding data. This gives the program a certified feel as a summary report is given at the end of the month. In	Later, I could add more trends that the graph summary page displays so the users get a bit more of a detailed overview. Moreover, I could add an option so the user can print the summary report off, if they would like to keep a physical copy for future references. This will help when documenting a report

	data present in the graphs.	addition, this could impress the head of nursing in the hospital, leading the users to have more pride with their work.	against a particular consultant who is not following hospital policies.

Failures

Screen	**What went wrong?**	**Why did this happen and if I overcome it, how?**	**How can I improve it in further development?**
Login Screen	When the user's details are correct the second time after an error message has popped up the first time due to the wrong details, the error message is displayed again.	This had occurred because I used two entry boxes for the password textbox. One was for the visiblePassword and one was for the hiddenPassword. In this scenario, when the user logs in with the correct password on the second time, their first input, which was wrong, is still being checked by the validation function so the error message appears. In addition, the visiblePassword textbox has the title "password" instead of "Password" because I could not use two textboxes with the same name. I could not overcome these problems using the current version of AppJar.	In the future I could remove the show/hide password all together if the users are not finding it a useful feature. It can be changes to a normal login will should run perfectly with no problems.
Daily Grid/Current Month Grid	If the user clicks on a different month which is not the current month, the current months data in each grid becomes empty.	This happened because I wanted the grids to be empty if any other month but the current month was clicked on. In doing this, I realised the user would not be able to view the other months when they want to. Despite this, I decided to still clear the data when the user wants to view another months data. As this problem, could not be overcome the user can only view the data from the current month. For example if it is April now they can only view Aprils data, otherwise the grids will be cleared for everything. (There are only two grids so if something happens to	In the future, I could create lots of **separate grids** so each change I make to a grid happens for that **grid only** and does not affect the other months of the year.

		one month it happens to all of them)	
Daily / Current Month Grid	When data was being saved from the Daily or Current Month Grid it was not saved under the correct month to the database.	There was no month column in the grid that specified the month responsible for the data in the grid. I **overcame** this by adding the month column in row [8] when the grids were created, so when any patient transfers were saved it would be added with the month its associated with. eg, January will have row[8] = "January".	I the future I could again create **separate grids** where I would not need the month column as data can be retrieved directly to the **grids name.**
Current Month Grid	Data cannot be retrieved to the Current Month Grid	As I was constantly**, testing** as I went along, I was transferring data from the "daily" grid to the "current month" grid quite a lot. Therefore, no data was actually being retrieved from the database so it slipped my mind to add this feature when I finished the program.	Soon, I will quickly add a retrieve function that gets the data from the current month table in my database so it can be shown in the current month grid.
(Month name) graph/Consultant's graph	The bar graphs cannot be shown without loading the grids first	As my program works iteratively with **time** quite a lot. Each function must work in order of the process. As a result, the data that needs to be inserted into the bar charts are within the grid. Therefore, for this to occur the data needs to be first loaded directly into the grids. I did not know how to overcome this due to my lack of knowledge and skill with concurrent processing of different functions.	In the future, I could set up the functions to run in accordance to **time** instead of relying on manual **button** pressing. This approach may accelerate the process of executing functions as they can be carried out seemingly at the same time rather than depending on the user to perform them sequentially.

Limitations

The biggest limitation I had in my program was using the AppJar interface. The Appjar UI limited the format and functionality of my grid. For simple tasks such as transferring data into another grid, so the data appears from the start row, I had to create an unnecessary overcomplicated function for this to occur which, took longer to think about and carry out. Another problem was that the grids could not work accordingly with the month buttons in Appjar. This is because the grids did not work like a database where data in the grids could be shown using SQL code.

Another limitation was the matplotlib python library as the page which displayed the bar charts could not be edited. I wanted to be able to edit the pages so I could add my own buttons and extra features on the page. Due to time, I decided to not implement these features as it would have taken a lot longer to figure out how to do it.

Another limitation was level of authority at the hospital. Originally, I had wanted to use MySQL to run the system on a server so any of the staff's computers could connect to the server and use the program each having their own account. Since, my client explained to me that she does not have the authority implement such a change in the clinic and due to the lack of experience I had with MySQL, I decided to run the database on only one main computer where everyone on the ward would go to access to my program (using SQLite for the database). To some extent I had overcome this limitation by finding an alternative solution to a sever-based LAN. Although, in the future I may update the whole program to run a local area network in the hospital so any user can access the program from any computer on the ward.

In addition, when programming I realised I had to introduce many more functions and variables I did not expect which were not in my design. Consequently, I had spent much more time creating these functions with the new variables which I had not prepared for. Furthermore, later in development, I discovered that some of the functions that I had initially created were unnecessary as I used more efficient and time-saving alternatives such as replacing the **check_month_g1** and **check_month_g2** function with a smaller piece of code in my **AddGridRow** function. The unnecessary functions I used first were a waste of valuable time and effort however, as I was able to overcome this limitation it increased my confidence and altered my approach to the problems I anticipated facing.

In conclusion, despite all these limitations, I had pleased my client and presented them a working software that had met all the necessary success criterions to an extent. Therefore, I think my project has been successful as it carries out its intended purpose and every function in my program operates efficiently.

Future Developments

Given more time, experience, or skill I would have implemented the following features to my system.

1) **Add reset password function:**
   As previously mentioned, I would have added a reset password function if I had more time but because I did not, the users are going to have access to the database with the developer's permission(me).

2) **Add Separate Grids for each Month:**
   As the user can only view the data for the current month, I could add separate grids for each month so the user can display any month's data at any given time.

3) **Add a print function for the graph summary:**
   There is currently only one way, which is screenshotting to save the graph summary for future references. I would like to add a print function to the summary page so nurses can gain a copy of the data which can be helpful to send to other nurses who have requested some information about a specific month. This could also be used to compare future findings with previous data to see if the efficiency of auditing patient transfers has improved or declined over time.

4) **Add an email function for the bar charts:**
   Since the bar charts can only be saved through the matplotlib(library) button, I could add an alternative way of retaining this data which is through emails. The user would be able to select a "save" button which then brings a pop up, prompting the user to enter their email. A copy of the graph would then be sent to the users email, which they could use when necessary. This may be to compare or present data to senior management if an issue has been detected.

5) **Grid Timeout feature:**
   As the grids may sometimes contain classified information in the "comment" section and because the program is withing a public hospital, if the user is distracted and remains on this page, anyone could view or change the confidential data. Therefore, I want to add a timing function that if the user is unactive for around 5 minutes the system automatically logs them out preventing unauthorised access to the program. The time may differ depending on my clients and the rest of the ward staff's opinion.

6) **Loading Screens:**

   When there is a lot of data that needs to be loaded into a grid it can take a little longer than usual to display it. With this knowledge I would like to create a loading screen feature which keeps the user in a waiting screen(this will have lots of images and colours so the user doesn't get bored) until the desired output is fully displayed in the grid.

7) Max tries password function:

To increase the security of my software, I would like to implement a max number of password tries function that only allows the user to enter a password 3 times. If the user reaches this maximum, they will be locked out of the system for 5 minutes and if this happens repeatedly the system timeout will only get longer.

Project Maintenance Issues

Users

- Nurses may forget their login details as the I (the developer) have provided these to them. Therefore, I will need to add a another function to my "forgot login" button that allows the user to reset their password

- Currently, if the user forgets their password, they have to check the database table containing that information to find out what it is or change their password

- If the company at the hospital changes and they decide to change the staff's login credentials they would have to be changed directly from the CSV file as there is no create new account function withing my program.

- I could add a hint function in my login page to decrease the chance of the user forgetting their password

Years

- After 2026, there is no more years displayed in my program so as of now my client would have to add the coming years themselves in the main code but this will not be long-term solution

- I could add a function so after every year a new button is created in the year page making this an autonomic process.

- I could instead create a list box that the user can just select the year they want the data for and this will be a long-term solution

- As the years go on, there will be multiple months such as "January" for 2023 and "January" for 2024. Therefore, I would also need to implement a **year column** in the grids, so the correct month and years' data can be viewed/added to.

Conclusion

Overall, this project has been enjoyable and a good experience. I think my project was successful because I met all the client agreed success criterions to a degree of certainty and my program can accomplish its intended purpose. Throughout development, I picked up on many programming skills which quickened my ability to solve problems (thinking computationally) and I could put these to use in future projects of my own. Being a software developer is difficult, as I came across many issues that I had to overcome for my program to be functional. However, when I was able to solve these issues, I had a sense of relief and desire to carry on. I enjoyed creating the functionality of my grids such as the transfer of data as it was a challenging task to execute as well as thinking of different approaches to fulfil the success criteria's I had made in my analysis. Now that I have completed this project, these skills I have learnt could be transferred to future projects where I can develop and design more sophisticated software.

Full Finished code:

```
import matplotlib   #The information for matplotlib can be found here - https://www.w3schools.com/python/matplotlib_pyplot.asp
import matplotlib.pyplot as plt
import numpy as np
import sqlite3 # The information for sqlite3 can be found here - https://docs.python.org/3/library/sqlite3.html
from appJar import gui #The information for appJar can be found here - http://appjar.info/
import time #The information for the time library  can be found here - https://docs.python.org/3/library/time.html
import time as mytime
import calendar  #The information for the calendar library  can be found here - https://docs.python.org/3/library/calendar.html#module-calendar
import datetime #The information for the datetime library - https://docs.python.org/3/library/datetime.html#module-datetime
#from datetime import date
con=sqlite3.connect('PatientTransferAudit.db') # creates a new file
cur=con.cursor()


app=gui("PatientTransferAudit")


def setupDB():

    #This creates the nurse table in the DB
    cur.execute("""CREATE TABLE IF NOT EXISTS "tbl_Nurse" (
                "NurseID"     INTEGER NOT NULL UNIQUE,
                "username"    TEXT NOT NULL UNIQUE,
                "password"    TEXT NOT NULL,
                PRIMARY KEY("NurseID" AUTOINCREMENT)
    )""")

    con.commit()
    #This creates the Daily audit table in the DB
    cur.execute("""CREATE TABLE "tbl_Daily" (
      "ConsultantID"    TEXT NOT NULL,
      "WardID"    TEXT NOT NULL,
      "RoomID"    TEXT NOT NULL,
      "TimeOfAdmission" time NOT NULL,
      "TransferDate"    date NOT NULL,
      "TransferTime"    time NOT NULL,
      "ReasonForTransfer"     INTEGER NOT NULL,
      "Comments"  TEXT NOT NULL,
      "Month"     TEXT NOT NULL


    )""")

    con.commit()

    #This creates the CurrentMonth table in the DB
    cur.execute("""CREATE TABLE "tbl_CurrentMonth" (
      "ConsultantID"    TEXT NOT NULL,
      "wardID"    TEXT NOT NULL,
      "RoomNo"    INTEGER NOT NULL,
      "TimeOfAdmission" time NOT NULL,
```

```
      "TransferDate"    date NOT NULL,
      "TransferTime"    time NOT NULL,
      "ReasonsforTransfer"    INTEGER NOT NULL,
      "Comments"  TEXT,
      "Month"     TEXT NOT NULL




    )""")
    con.commit()



    #This creates the RoomNo table in the DB
    cur.execute("""CREATE TABLE IF NOT EXISTS  "tbl_RoomNo" (
                    "RoomNo"  INTEGER NOT NULL UNIQUE,
                    "ConsultantID"  TEXT NOT NULL,
                    "NurseID" INTEGER NOT NULL UNIQUE,
                    PRIMARY KEY("RoomNo"),
                    FOREIGN KEY("ConsultantID") REFERENCES
"tbl_Consultant"("ConsultantID"),
                    FOREIGN KEY("NurseID") REFERENCES
"tbl_Nurse"("NurseID")
    )""")
    con.commit()



    #This creates the Ward table in the DB
    cur.execute("""CREATE TABLE IF NOT EXISTS  "tbl_Ward" (
                    "WardID"  TEXT NOT NULL,
                    "ConsultantID"  TEXT NOT NULL,
                    "wardName"      TEXT NOT NULL UNIQUE,
                    PRIMARY KEY("WardID"),
                    FOREIGN KEY("ConsultantID") REFERENCES
"tbl_RoomNo"("ConsultantID")
    )""")
    con.commit()



    #This creates the Admission table in the DB
    cur.execute("""CREATE TABLE IF NOT EXISTS  "tbl_Admission" (
                "WardID"      TEXT NOT NULL,
                "RoomNo"      INTEGER NOT NULL UNIQUE,
                "TransferDate"      date NOT NULL,
                "TransferTime"      time NOT NULL,
                "ReasonForTransfer" INTEGER NOT NULL,
                FOREIGN KEY("RoomNo") REFERENCES "tbl_RoomNo"("RoomNo"),
                FOREIGN KEY("WardID") REFERENCES "tbl_Ward"("WardID")
    )""")
    con.commit()

    #This creates the Consultant table in the DB
    cur.execute("""CREATE TABLE IF NOT EXISTS  "tbl_Consultant" (
                "ConsultantID"      TEXT NOT NULL,
                "Speciality " TEXT NOT NULL,
                "RoomNo"      INTEGER NOT NULL,
```

```
                PRIMARY KEY("ConsultantID")
                )""")
    con.commit()




    #populates Nurse table with logins
    NurseCredentialsFile=open("NurseCredentials.csv","r")#Information added
from NurseCrendentials csv file
    for line in NurseCredentialsFile:
        line=line.strip()#removes \n
        userName,passWord=line.split(",")
        cur.execute("INSERT INTO tbl_Nurse (username,password) VALUES
(?,?)",[userName,passWord])


        con.commit()#all changes are commited
        print("Database Setup Complete")



    #populates Consultant table with their initials and their speciality
    ConsultantDetailsFile=open("ConsultantDetails.csv","r")#Information
added from ConsultantDetails csv file
    for line in ConsultantDetailsFile:
        line=line.strip()#removes\n
        cID,Speciality,roomNo=line.split(",")
        cur.execute("INSERT INTO tbl_Consultant VALUES
(?,?,?)",[cID,Speciality,roomNo])

        con.commit()#all changes are commited
        print("Database Setup Complete")



#setupDB()






#creates functions for each of my buttons on every page of my program
def press(button):

    if button=="Login":
        validLogin()#runs subroutine to check if username and password is
valid



    elif button=="Forgot Login":
        app.infoBox("Help","Contact 216298@kingsmead.org for more
information or any quieries reqarding your credentials")

    elif button=="ButtonPD":
        app.hideSubWindow("MainMenu")#hides main menu
```

```
        app.showSubWindow("YearDisplayPD")#shows YearPD page
        retrive_tbl_Daily()




    elif button=="ButtonAG":
        app.hideSubWindow("MainMenu")#hides main menu
        app.showSubWindow("YearDisplayAG")


    elif button=="backYPD":
        app.hideSubWindow("YearDisplayPD")#hides YearPD page
        app.showSubWindow("MainMenu")#shows main menu



    elif button=="backYAG":
        app.hideSubWindow("YearDisplayAG")#hides YearAG page
        app.showSubWindow("MainMenu")#shows main menu

    elif button=="backMAG":
        app.hideSubWindow("MonthDisplayAG")#hides MonthAG page
        app.showSubWindow("YearDisplayAG")#shows YearAG page


#Allows each year selected to direct to the same month page of Patient Data
    elif button=="PD_2023" or button=="PD_2024" or button=="PD_2025" or
button=="PD_2026":
        app.hideSubWindow("YearDisplayPD")#hides YearPD page
        app.showSubWindow("MonthDisplayPD")#shows MonthPD page



#Allows each year selected to direct to the same month page of audit graphs
    elif button=="AG_2023" or button=="AG_2024" or button=="AG_2025" or
button=="AG_2026":
        app.hideSubWindow("YearDisplayAG")#hides YearAG page
        app.showSubWindow("MonthDisplayAG")#shows MonthPD page


    elif button=="backMPD":
        app.hideSubWindow("MonthDisplayPD")#hides MonthPD page
        app.showSubWindow("YearDisplayPD")#shows YearPD page


    elif button=="backMAG":
        app.hideSubWindow("MonthDisplayAG")#hides MonthAG page
        app.showSubWindow("YearDisplayAG")#shows YearAG page



    elif button=="PD_January" or button=="PD_February" or
button=="PD_March" or button=="PD_April" or button=="PD_May" or
button=="PD_June" or button=="PD_July" or button=="PD_August" or
button=="PD_September" or button=="PD_October" or button=="PD_November" or
button=="PD_December":
        app.hideSubWindow("MonthDisplayPD")
        app.showSubWindow("PatientTransferDatabases")
```

```
        set_month_to_add_row(button)#sets row[8] of grids ro the month of
the button clicked
        month_check(button)#checks if row[8] in grid already has the month
name
        AuditGraph()#calculates graph summaries
        ConsultantGraph()#Calculates consultant graph data




    elif button=="Daily":
        app.hideSubWindow("PatientTransferDatabases")
        app.showSubWindow("DailyDatabase")




    elif button=="Current Month Database":
        app.hideSubWindow("PatientTransferDatabases")
        app.showSubWindow("CurrentMonthDatabase")


    elif button=="BackPTD":
        app.hideSubWindow("PatientTransferDatabases")
        app.showSubWindow("MonthDisplayPD")# switch back to previous
subwindow

    elif button=="BackDD":
        app.hideSubWindow("DailyDatabase")
        app.showSubWindow("PatientTransferDatabases")# switch back to
previous subwindow

    elif button=="BackCD":
        app.hideSubWindow("CurrentMonthDatabase")
        app.showSubWindow("PatientTransferDatabases")# switch back to
previous subwindow

    elif button=="BackAGP":
        app.hideSubWindow("AuditGraphs")
        app.showSubWindow("MonthDisplayAG")# switch back to previous
subwindow

    elif button=="VGS":
        GraphSummary()#shows graph summaries
        app.hideSubWindow("AuditGraphs")
        app.showSubWindow("GraphSummary")

    elif button=="CG":
        app.hideSubWindow("AuditGraphs")
        PlotConsultantGraph()#plots consultant graph
        app.showSubWindow("AuditGraphs")# switch back to previous subwindow


    elif button=="BackGS":
        app.hideSubWindow("GraphSummary")
        app.showSubWindow("AuditGraphs")# switch back to previous subwindow

    elif button=="January":
        app.hideSubWindow("AuditGraphs")
        MonthlyGraph()
```

```
        app.showSubWindow("AuditGraphs")# switch back to previous subwindow

    elif button=="February":
        app.hideSubWindow("AuditGraphs")
        MonthlyGraph()
        app.showSubWindow("AuditGraphs")# switch back to previous subwindow

    elif button=="March":
        app.hideSubWindow("AuditGraphs")
        MonthlyGraph()
        app.showSubWindow("AuditGraphs")# switch back to previous subwindow

    elif button=="April":
        MonthlyGraph()
        app.showSubWindow("AuditGraphs")# switch back to previous subwindow

    elif button=="May":
        MonthlyGraph()
        app.showSubWindow("AuditGraphs")# switch back to previous subwindow

    elif button=="June":
        MonthlyGraph()
        app.showSubWindow("AuditGraphs")# switch back to previous subwindow

    elif button=="July":
        MonthlyGraph()
        app.showSubWindow("AuditGraphs")# switch back to previous subwindow

    elif button=="August":
        MonthlyGraph()
        app.showSubWindow("AuditGraphs")# switch back to previous subwindow

    elif button=="September":
        MonthlyGraph()
        app.showSubWindow("AuditGraphs")# switch back to previous subwindow

    elif button=="October":
        MonthlyGraph()
        app.showSubWindow("AuditGraphs")# switch back to previous subwindow

    elif button=="November":
        MonthlyGraph()
        app.showSubWindow("AuditGraphs")# switch back to previous subwindow

    elif button=="December":
        MonthlyGraph()
        app.showSubWindow("AuditGraphs")# switch back to previous subwindow


#Allows the month clicked on the selection page to display the button
required to present that graph
def monthGraph(button):

    if  button=="AG_January":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.setSticky("w")#sets the months button to the left side of the
screen
        app.setPadding([40,40])
        app.addNamedButton("January's graph","January",press,1,0)
        app.setButtonBg("January", "lightblue")
```

```
    elif button=="AG_February":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.setSticky("w")#sets the months button to the left side of the
screen
        app.addNamedButton("February's graph","February",press,1,0)
        app.setButtonBg("February", "lightblue")

    elif button=="AG_March":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.setSticky("w")#sets the months button to the left side of the
screen
        app.addNamedButton("March's graph","March",press,1,0)
        app.setButtonBg("March", "lightblue")


    elif button=="AG_April":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.setSticky("w")#sets the months button to the left side of the
screen
        app.addNamedButton("April's graph","April",press,1,0)
        app.setButtonBg("April", "lightblue")

    elif button=="AG_May":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.setSticky("w")#sets the months button to the left side of the
screen
        app.addNamedButton("May's graph","May",press,1,0)
        app.setButtonBg("May", "lightblue")


    elif button=="AG_June":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.setSticky("w")#sets the months button to the left side of the
screen
        app.addNamedButton("June's graph","June",press,1,0)
        app.setButtonBg("June", "lightblue")


    elif button=="AG_July":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.setSticky("w")#sets the months button to the left side of the
screen
        app.addNamedButton("July's graph","July",press,1,0)
        app.setButtonBg("July", "lightblue")

    elif button=="AG_August":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.setSticky("w")#sets the months button to the left side of the
screen
        app.addNamedButton("August's graph","August",press,1,0)
        app.setButtonBg("August", "lightblue")
```

```
    elif button=="AG_September":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.setSticky("w")#sets the months button to the left side of the
screen
        app.addNamedButton("September's graph","September",press,1,0)
        app.setButtonBg("September", "lightblue")


    elif button=="AG_October":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.setSticky("w")#sets the months button to the left side of the
screen
        app.addNamedButton("October's graph","October",press,1,0)
        app.setButtonBg("October", "lightblue")


    elif button=="AG_Novemeber":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.setSticky("w")#sets the months button to the left side of the
screen
        app.addNamedButton("November's graph","November",press,1,0)
        app.setButtonBg("November", "lightblue")


    elif button=="AG_December":
        app.hideSubWindow("MonthDisplayAG")
        app.showSubWindow("AuditGraphs")
        app.setSticky("w")#sets the months button to the left side of the
screen
        app.addNamedButton("December's graph","December",press,1,0)
        app.setButtonBg("December", "lightblue")


        app.stopSubWindow()#stops subwindow




#creates a function so the user can logout
def logout(leave):

    leave = app.questionBox("Log Out","Are you sure you want to log out?")
    if leave==True:#checks if conditon is true and then open the MainMenu
subwindow
        app.hideSubWindow("MainMenu")
        app.showSubWindow("Login")


    else:
        app.showSubWindow("MainMenu")


#isalnum = alphabet and number
#is alpha = checks all characters is alphabet
#is digit = checks digit only

#mypassword[0].isupper()
```

```
#Allows user to only login if they enter valid inputs that match the value
stored in the database
def validLogin():
    global hidden_password
    username = app.getEntry("Username :")#sets username variable to the
username entered by the user
    #password can be written in either hidden or visible mode
    hidden_password = app.getEntry("hiddenPassword")#hidden  password
    visible_password = app.getEntry("visiblePassword")#visible password
    cur.execute("SELECT password FROM tbl_Nurse WHERE username=?",
[username])#selects password from the database associated with the  entered
username
    result=cur.fetchone()#fetches the passwords from the queried username
in the database and stores it in the variable result
    print(result,hidden_password,visible_password)
    if len(hidden_password)==0 and len(visible_password)!=0:
        hidden_password=visible_password#if hidden password is empty
visible password can be stored as the hidden password
    if len(visible_password)==0 and len(hidden_password)!=0:
        visible_password=hidden_password#if visible password is empty
hidden password can be stored as the visible password

#Checks username or password field is empty
    if len(username)== 0 or (len(hidden_password)== 0 or
len(visible_password)== 0):
        app.errorBox("Access denied","Username or Password not present")
#Checks username is 6 characters long
    elif len(username)!=6:
        app.errorBox("Access denied","Username not 6 characters long")
#If either password entry box is not 8 characters long user is denied
    elif len(hidden_password)!=8 and len(visible_password)!=8:
        app.errorBox("Access denied","Password not 8 characters long")

#Checks either password entry boxes have a capital letter

    elif hidden_password[0].islower() or visible_password[0].islower():
        app.errorBox("Access denied","Password not in correct format")

#Makes sure the only way to login is if the username and password matches
exactly
    elif result==None or (result[0]==username or
(result[0]!=hidden_password  or result[0]!=visible_password)):
         app.errorBox("Access denied","Username or Password is incorrect")

#if credentials are valid user can login successfully
    else:
        if result[0]==hidden_password  or result[0]==visible_password:
            app.hideSubWindow("Login")#User may login as details are
correct
            app.clearEntry("visiblePassword")#clears password entry when
user logs in
            app.clearEntry("hiddenPassword")
            app.showSubWindow("MainMenu")#Shows the MainMenu
            app.unbindKey("Return")



def set_month_to_add_row(button):
    global month_in_new_row#makes global variable for each month from
press(button)
    if button == "PD_January":
```

```
        month_in_new_row = [""] * 9#create a list with empty strings for
all columns
        month_in_new_row[8] = "January"#sets index[8] column to "January"
        btn = True#set btn = true so addGridRow() can now take place
        addGridRowg1(btn)

    if button == "PD_February":
        month_in_new_row = [""] * 9#create a list with empty strings for
all columns
        month_in_new_row[8] = "February"#sets index[8] column to "February"
        btn = True#set btn = true so addGridRow() can now take place
        addGridRowg1(btn)

    if button == "PD_March":
        month_in_new_row = [""] * 9#create a list with empty strings for
all columns
        month_in_new_row[8] = "March"#sets index[8] column to "March"
        btn = True#set btn = true so addGridRow() can now take place
        addGridRowg1(btn)

    if button == "PD_April":
        month_in_new_row = [""] * 9#create a list with empty strings for
all columns
        month_in_new_row[8] = "April"#sets index[8] column to "April"
        btn = True#set btn = true so addGridRow() can now take place
        addGridRowg1(btn)

    if button == "PD_May":
        month_in_new_row = [""] * 9#create a list with empty strings for
all columns
        month_in_new_row[8] = "May"#sets index[8] column to "May"
        btn = True#set btn = true so addGridRow() can now take place
        addGridRowg1(btn)

    if button == "PD_June":
        month_in_new_row = [""] * 9#create a list with empty strings for
all columns
        month_in_new_row[8] = "June"#sets index[8] column to "June"
        btn = True#set btn = true so addGridRow() can now take place
        addGridRowg1(btn)

    if button == "PD_July":
        month_in_new_row = [""] * 9#create a list with empty strings for
all columns
        month_in_new_row[8] = "July"#sets index[8] column to "July"
        btn = True#set btn = true so addGridRow() can now take place
        addGridRowg1(btn)

    if button == "PD_August":
        month_in_new_row = [""] * 9#create a list with empty strings for
all columns
        month_in_new_row[8] = "August"#sets index[8] column to "August"
        btn = True#set btn = true so addGridRow() can now take place
        addGridRowg1(btn)


    if button == "PD_September":
        month_in_new_row = [""] * 9#create a list with empty strings for
all columns
        month_in_new_row[8] = "September"#sets index[8] column to
"September"
```

```
        btn = True#set btn = true so addGridRow() can now take place
        addGridRowg1(btn)


    if button == "PD_October":
        month_in_new_row = [""] * 9#create a list with empty strings for
all columns
        month_in_new_row[8] = "October"#sets index[8] column to "October"
        btn = True#set btn = true so addGridRow() can now take place
        addGridRowg1(btn)

    if button == "PD_November":
        month_in_new_row = [""] * 9#create a list with empty strings for
all columns
        month_in_new_row[8] = "November"#sets index[8] column to "November"
        btn = True#set btn = true so addGridRow() can now take place
        addGridRowg1(btn)

    if button ==  "PD_December":
        month_in_new_row = [""] * 9#create a list with empty strings for
all columns
        month_in_new_row[8] = "December"#sets index[8] column to "December"
        btn = True#set btn = true so addGridRow() can now take place
        addGridRowg1(btn)



def check_Row_Inputs_g1(btn):

    data = app.getGridRow("g1", btn)#gets inputs from the row and stores it
in the variable data
    print(data)
    if data is None:
            print("No data")#prints no data if row is empty
    else:
        for row in data:
            print(row)
            if any(row):#for any row in the table
                    consultant_id = data[0]#assigns each row index to the
corresponding field name
                    ward_id = data[1]
                    Room_ID = data[2]
                    Time_Of_Admission = data[3]
                    transfer_date = data[4]
                    transfer_time = data[5]
                    reason_for_transfer = data[6]
                    comments = data[7]
                    Month = data[8]

#isalnum = alphabet and number

    if  consultant_id == ""  or (not consultant_id.isalnum()):#print error
message if ConsultantID cell is empty or contains a symbol
         app.errorBox("Cannot save","ConsultantID is empty or contains
erroneous data")
         return False#sets subroutine to false

    elif ward_id == "" or (not ward_id.isdigit()):#print error message if
wardID cell is empty or not an integer
         app.errorBox("Cannot save","WardID is not a number")
         return False#sets subroutine to false
```

```
    elif Room_ID == "" or (not Room_ID.isdigit()) or len(Room_ID)>3:#print
error message if RoomID cell is empty or not an integer or there is more
than 3 integers
         app.errorBox("Cannot save","Room ID is not a number")
         return False#sets subroutine to false

    elif reason_for_transfer == "" or (not
reason_for_transfer.isdigit()):#print error message if ReasonForTransfer
cell is empty or not an integer
        app.errorBox("Cannot save","Reason for transfer is not a number")
        return False#sets subroutine to false

    elif int(reason_for_transfer)<1 or int(reason_for_transfer)>6 :#print
error message if ReasonForTransfer cell value is not between 1 and 6
        app.errorBox("Cannot save","Reason for transfer integer is out of
the key's range")
        return False#sets subroutine to false

    try:
        datetime.datetime.strptime(Time_Of_Admission, '%H:%M')#print error
message if Time_Of_Admission is empty or the time is not in HH/MM format
    except ValueError:#if right type but inappropiate value used run except
clause
        app.errorBox("Cannot save","TimeOfAdmission is not in HH:MM
format")
        return False#sets subroutine to false

    try:
        datetime.datetime.strptime(transfer_date, '%d/%m/%y')#print error
message if TransferDate is empty or the date is not in DD/MM/YY format
    except ValueError:#if right type but inappropiate value used run except
clause
        app.errorBox("Cannot save","TransferDate is not in DD/MM/YY
format")
        return False#sets subroutine to false

    try:
        datetime.datetime.strptime(transfer_time, '%H:%M')#print error
message if TransferTime is empty or the time is not in HH/MM format
    except ValueError:#if right type but inappropiate value used run except
clause
        app.errorBox("Cannot save","TransferTime is not in HH:MM
format")#sets subroutine to false
        return False


    return True



def check_Row_Inputs_g2(btn):

    data = app.getGridRow("g2", btn)#gets inputs from the row and stores it
in the variable data
    print(data)
    if data is None:
            print("No data")#prints no data if row is empty
    else:
        for row in data:
            print(row)
```

```
            if any(row):#for any row in the table
                    consultant_id = data[0]#assigns each row index to the
corresponding field name
                    ward_id = data[1]
                    Room_ID = data[2]
                    Time_Of_Admission = data[3]
                    transfer_date = data[4]
                    transfer_time = data[5]
                    reason_for_transfer = data[6]
                    comments = data[7]
                    Month = data[8]



#isalnum = alphabet and number

    if  consultant_id == ""  or (not consultant_id.isalnum()):
         app.errorBox("Cannot save","ConsultantID is empty or contains
erroneous data")#print error message if ConsultantID cell is empty or
contains a symbol
         return False#sets subroutine to false

    elif ward_id == "" or (not ward_id.isdigit()):#print error message if
wardID cell is empty or not an integer
         app.errorBox("Cannot save","WardID is not a number")
         return False#sets subroutine to false

    elif Room_ID == "" or (not Room_ID.isdigit()) or len(Room_ID)>3:#print
error message if RoomID cell is empty or not an integer or there is more
than 3 integers
         app.errorBox("Cannot save","Room ID is not a number")
         return False#sets subroutine to false

    elif reason_for_transfer == "" or (not
reason_for_transfer.isdigit()):#print error message if ReasonForTransfer
cell is empty or not an integer
        app.errorBox("Cannot save","Reason for transfer is not a number")
        return False#sets subroutine to false


    elif int(reason_for_transfer)<1 or int(reason_for_transfer)>6 :#print
error message if ReasonForTransfer cell value is not between 1 and 6
        app.errorBox("Cannot save","Reason for transfer integer is out of
the key's range")
        return False#sets subroutine to false

    try:
        datetime.datetime.strptime(Time_Of_Admission, '%H:%M')#print error
message if Time_Of_Admission is empty or the time is not in HH/MM format
    except ValueError:#if right type but inappropiate value used run except
clause
        app.errorBox("Cannot save","TimeOfAdmission is not in HH:MM
format")
        return False#sets subroutine to false

    try:
        datetime.datetime.strptime(transfer_date, '%d/%m/%y')#print error
message if TransferDate is empty or the date is not in DD/MM/YY format
    except ValueError:#if right type but inappropiate value used run except
clause
```

```
        app.errorBox("Cannot save","TransferDate is not in DD/MM/YY
format")
        return False#sets subroutine to false

    try:
        datetime.datetime.strptime(transfer_time, '%H:%M')#print error
message if TransferTime is empty or the time is not in HH/MM format
    except ValueError:#if right type but inappropiate value used run except
clause
        app.errorBox("Cannot save","TransferTime is not in HH:MM
format")#sets subroutine to false
        return False


    return True

def addGridRowg1(btn):

    rows = app.getGridRowCount("g1")
    for i in range(rows):
        row = app.getGridRow("g1", i)
        row[8] = month_in_new_row[8] # set the 9th column (index 8) to the
value of month_in_new_row
        app.replaceGridRow("g1", i, row)#replace original row with updated
version




    rows = app.getGridRowCount("g1")#count number of rows in g1
    if rows == 0:
        app.addGridRow("g1", month_in_new_row)#add current month to row[8]
if theres no rows at all
    else:
        for i in range(rows):
            row = app.getGridRow("g1", i)#gets each individual row
            if not any(row):#checks if each field row is empty
                continue#if the row is not empty then continue
            if row[8] != month_in_new_row[8]:#checks if row[8] already
contains the current month
                row[8] = month_in_new_row[8]#if not assign row[8] to the
current month
                app.replaceGridRow("g1", i, row)#Replaces the row with the
updated changes


        else:
            app.addGridRow("g1", month_in_new_row)#if no rows have an empty
row[8] add new row to grid with the corresponding month


def addGridRowg2(btn):

    rowsg2 = app.getGridRowCount("g2")#count number of rows in g2
    row = app.getGridRow("g2", 0)#gets the first row in g2
    if row[8] != month_in_new_row[8] or row[8] ==
month_in_new_row[8]:#checks if row[8] already contains the current month or
if it does not
                row[8] = month_in_new_row[8]#if not assign row[8] to value
in month_in_new_row
```

```
                app.addGridRow("g2", month_in_new_row)#add grid row
containing displaying the value in month_in_new_row in row[8]




def SaveTableg1(btn):

    if not check_Row_Inputs_g1(btn):#if the check_Row_Inputs_g1(btn)
subroutine is false this function will not execute the following code
        return
    save = app.yesNoBox("Save transfer", "Are you sure you want to add this
patient transfer? ALL CHANGES ARE FINAL!!",)

    if save==True:
        data = app.getGridRow("g1", btn)#gets inputs from the row and
stores it in the variable data
        print(data)
        if data is None:
            print("No data")#prints no data if row is empty
        else:
            for row in data:
                print(row)
                if any(row):#for any row in the table
                        consultant_id = data[0]#assigns each row index to
the corresponding field name
                        ward_id = data[1]
                        Room_No = data[2]
                        Time_Of_Admission = data[3]
                        transfer_date = data[4]
                        transfer_time = data[5]
                        reason_for_transfer = data[6]
                        comments = data[7]
                        Month = data[8]




        conn = sqlite3.connect('PatientTransferAudit.db')#connect db
        cur = conn.cursor()
        cur.execute("INSERT INTO tbl_Daily (ConsultantID, WardID, RoomID,
TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments,
Month) VALUES (?, ?, ?, ?, ?, ?, ?, ?,?)", (consultant_id, ward_id,
Room_No, Time_Of_Admission, transfer_date, transfer_time,
reason_for_transfer, comments,Month))
        conn.commit()#commits all changes into the tbl_Daily in my database
        print("complete")
    else:

        app.showSubWindow("DailyDatabase")



def SaveTableg2(btn):

    if not check_Row_Inputs_g2(btn):#if the check_Row_Inputs_g2(btn)
subroutine is false this function will not execute the following code
```

```
        return
    save = app.yesNoBox("Save transfer 2", "Are you sure you want to add
this patient transfer to this month? ALL CHANGES ARE FINAL!!",)

    if save==True:
        data = app.getGridRow("g2", btn)#gets inputs from the row and
stores it in the variable data
        print(data)
        if data is None:
            print("No data")#prints no data if row is empty
        else:
            for row in data:
                print(row)
                if any(row):#for any row in the table
                        consultant_id = data[0]#assigns each row index to
the corresponding field name
                        ward_id = data[1]
                        Room_No = data[2]
                        Time_Of_Admission = data[3]
                        transfer_date = data[4]
                        transfer_time = data[5]
                        reason_for_transfer = data[6]
                        comments = data[7]
                        Month = data[8]
        conn = sqlite3.connect('PatientTransferAudit.db')#connect db
        cur = conn.cursor()
        cur.execute("INSERT INTO tbl_CurrentMonth (ConsultantID, WardID,
RoomNo, TimeOfAdmission, TransferDate, TransferTime, ReasonsforTransfer,
Comments, Month) VALUES (?, ?, ?, ?, ?, ?, ?, ?,?)", (consultant_id,
ward_id, Room_No, Time_Of_Admission, transfer_date, transfer_time,
reason_for_transfer, comments,Month))
        conn.commit()#commits all changes into the tbl_CurrentMonth in my
database
        print("complete")

    else:

        app.showSubWindow("CurrentMonthDatabase")
```

```
def retrive_tbl_Daily():
    conn = sqlite3.connect('PatientTransferAudit.db')#connect db
    cur = conn.cursor()
    cur.execute("SELECT COUNT(*) FROM tbl_daily")#counts all the rows in
the table
    row_count = cur.fetchone()[0]#stores number of rows in the variable
row_count
    if row_count != 0:
        today = datetime.date.today()#gets todays date
        print(today)
        if today.month == 1:
            cur.execute("SELECT ConsultantID, WardID, RoomID,
TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments,
Month FROM tbl_Daily WHERE Month='January'")#selects all rows with data
from the table with the month January
        elif today.month == 2:
```

```
            cur.execute("SELECT ConsultantID, WardID, RoomID,
TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments,
Month FROM tbl_Daily WHERE Month='February'")#selects all rows with data
from the table with the month February
        elif today.month == 3:
            cur.execute("SELECT ConsultantID, WardID, RoomID,
TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments,
Month FROM tbl_Daily WHERE Month='March'")#selects all rows with data from
the table with the month March
        elif today.month == 4:
            cur.execute("SELECT ConsultantID, WardID, RoomID,
TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments,
Month FROM tbl_Daily WHERE Month='April'")#selects all rows with data from
the table with the month April
        elif today.month == 5:
            cur.execute("SELECT ConsultantID, WardID, RoomID,
TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments,
Month FROM tbl_Daily WHERE Month='May'")#selects all rows with data from
the table with the month May
        elif today.month == 6:
            cur.execute("SELECT ConsultantID, WardID, RoomID,
TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments,
Month FROM tbl_Daily WHERE Month='June'")#selects all rows with data from
the table with the month June
        elif today.month == 7:
            cur.execute("SELECT ConsultantID, WardID, RoomID,
TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments,
Month FROM tbl_Daily WHERE Month='July'")#selects all rows with data from
the table with the month July
        elif today.month == 8:
            cur.execute("SELECT ConsultantID, WardID, RoomID,
TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments,
Month FROM tbl_Daily WHERE Month='August'")#selects all rows with data from
the table with the month August
        elif today.month == 9:
            cur.execute("SELECT ConsultantID, WardID, RoomID,
TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments,
Month FROM tbl_Daily WHERE Month='September'")#selects all rows with data
from the table with the month September
        elif today.month == 10:
            cur.execute("SELECT ConsultantID, WardID, RoomID,
TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments,
Month FROM tbl_Daily WHERE Month='October'")#selects all rows with data
from the table with the month October
        elif today.month == 11:
            cur.execute("SELECT ConsultantID, WardID, RoomID,
TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments,
Month FROM tbl_Daily WHERE Month='November'")#selects all rows with data
from the table with the month November
        elif today.month == 12:
            cur.execute("SELECT ConsultantID, WardID, RoomID,
TimeOfAdmission, TransferDate, TransferTime, ReasonForTransfer, Comments,
Month FROM tbl_Daily WHERE Month='December'")#selects all rows with data
from the table with the month Decem



    else:
            print("table is empty")
```

```
    rows = cur.fetchall()
    headers = ["ConsultantID", "WardID","RoomID",
"TimeOfAdmission","TransferDate","TransferTime","ReasonForTransfer","Commen
ts","Month"]#assigns all the titles for each column in the grid to the
variable headers
    app.replaceAllGridRows("g1",rows)#replaces all the rows in my grid with
the retrived data
    first_row = app.getGridRow("g1",-1)#stores the first row that is
currently in the header in the variable first_row
    app.setGridHeaders("g1",headers)#set the grid headers to the headers of
the Daily grid
    app.addGridRow("g1",first_row)#adds the first row to the end of the
existing grid
    print(rows)


def month_check(button):
   rowsg1 = app.getGridRowCount("g1")
   today = datetime.date.today() #gets today's date
   print(today)
   if (button == "PD_January" or button == "PD_February" or button ==
"PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June"
or button=="PD_July" or button=="PD_August" or button=="PD_September" or
button=="PD_October" or button=="PD_November" or button=="PD_December") and
(today.month ==1):#checks if the current month is january
            for i in range(rowsg1):
                row = app.getGridRow("g1", i)#gets each row in the grid
                print(row)
                if all(val == "" for val in row):
                   continue # skip empty rows
                if row[8] != "January":
                    empty_row = [""] * 9#creates an empty list of 9 values
                    app.replaceGridRow("g1", i, empty_row)#adds the empty
rows to the grid if the row[8]!=January
                else:
                    return TransferDailyTable(button)


   if (button == "PD_January" or button == "PD_February" or button ==
"PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June"
or button=="PD_July" or button=="PD_August" or button=="PD_September" or
button=="PD_October" or button=="PD_November" or button=="PD_December") and
today.month == 2:#checks if the current month is February
             for i in range(rowsg1):
                row = app.getGridRow("g1", i)#gets each row in the grid
                print(row)
                if all(val == "" for val in row):
                   continue # skip empty rows
                if row[8] != "February":
                    empty_row = [""] * 9#creates an empty list of 9 values
                    app.replaceGridRow("g1", i, empty_row)#adds the empty
rows to the grid if the row[8]!=February
                else:
                     return TransferDailyTable(button)

   if (button == "PD_January" or button == "PD_February" or button ==
"PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June"
or button=="PD_July" or button=="PD_August" or button=="PD_September" or
```

```
button=="PD_October" or button=="PD_November" or button=="PD_December") and
today.month == 3:#checks if the current month is March
             print("printing",rowsg1)
             for i in range(rowsg1):
               row = app.getGridRow("g1", i)#gets each row in the grid
               print(row)
               if all(val == "" for val in row):
                continue # skip empty rows
               if row[8] != "March":
                   print("creating blank rows")
                   empty_row = [""] * 9#creates an empty list of 9 values
                   app.replaceGridRow("g1", i, empty_row)#adds the empty
rows to the grid if the row[8]!=March
               else:
                     return TransferDailyTable(button)
```

```
   if (button == "PD_January" or button == "PD_February" or button ==
"PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June"
or button=="PD_July" or button=="PD_August" or button=="PD_September" or
button=="PD_October" or button=="PD_November" or button=="PD_December") and
today.month == 4:#checks if the current month is April
             for i in range(rowsg1):
               row = app.getGridRow("g1", i)#gets each row in the grid
               print(row)
               if all(val == "" for val in row):
                  continue # skip empty rows
               if row[8] != "April":
                   empty_row = [""] * 9#creates an empty list of 9 values
                   app.replaceGridRow("g1", i, empty_row)#adds the empty
rows to the grid if the row[8]!=April
               else:
                   #pass
                   return TransferDailyTable(button)
```

```
   if (button == "PD_January" or button == "PD_February" or button ==
"PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June"
or button=="PD_July" or button=="PD_August" or button=="PD_September" or
button=="PD_October" or button=="PD_November" or button=="PD_December") and
today.month == 5:#checks if the current month is May
             for i in range(rowsg1):
               row = app.getGridRow("g1", i)#gets each row in the grid
               print(row)
               if all(val == "" for val in row):
                  continue # skip empty rows
               if row[8] != "May":
                   empty_row = [""] * 9#creates an empty list of 9 values
                   app.replaceGridRow("g1", i, empty_row)#adds the empty
rows to the grid if the row[8]!=May
               else:
                   return TransferDailyTable(button)
```

```
   if (button == "PD_January" or button == "PD_February" or button ==
"PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June"
or button=="PD_July" or button=="PD_August" or button=="PD_September" or
button=="PD_October" or button=="PD_November" or button=="PD_December") and
today.month == 6:#checks if the current month is June
              for i in range(rowsg1):
                row = app.getGridRow("g1", i)#gets each row in the grid
                print(row)
                if all(val == "" for val in row):
                   continue # skip empty rows
                if row[8] != "June":
                    empty_row = [""] * 9#creates an empty list of 9 values
                    app.replaceGridRow("g1", i, empty_row)#adds the empty
rows to the grid if the row[8]!=June
                else:
                    return TransferDailyTable(button)



   if (button == "PD_January" or button == "PD_February" or button ==
"PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June"
or button=="PD_July" or button=="PD_August" or button=="PD_September" or
button=="PD_October" or button=="PD_November" or button=="PD_December") and
today.month == 7:#checks if the current month is July
              for i in range(rowsg1):
                row = app.getGridRow("g1", i)#gets each row in the grid
                print(row)
                if all(val == "" for val in row):
                   continue # skip empty rows
                if row[8] != "July":
                    empty_row = [""] * 9#creates an empty list of 9 values
                    app.replaceGridRow("g1", i, empty_row)#adds the empty
rows to the grid if the row[8]!=July
                else:
                    return TransferDailyTable(button)


   if (button == "PD_January" or button == "PD_February" or button ==
"PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June"
or button=="PD_July" or button=="PD_August" or button=="PD_September" or
button=="PD_October" or button=="PD_November" or button=="PD_December") and
today.month == 8:#checks if the current month is August
              for i in range(rowsg1):
                row = app.getGridRow("g1", i)#gets each row in the grid
                print(row)
                if all(val == "" for val in row):
                   continue # skip empty rows
                if row[8] != "August":
                    empty_row = [""] * 9#creates an empty list of 9 values
                    app.replaceGridRow("g1", i, empty_row)#adds the empty
rows to the grid if the row[8]!= August
                else:
                    return TransferDailyTable(button)


   if (button == "PD_January" or button == "PD_February" or button ==
"PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June"
or button=="PD_July" or button=="PD_August" or button=="PD_September" or
button=="PD_October" or button=="PD_November" or button=="PD_December") and
today.month == 9:#checks if the current month is September
```

```
            for i in range(rowsg1):
              row = app.getGridRow("g1", i)#gets each row in the grid
              print(row)
              if all(val == "" for val in row):
                 continue # skip empty rows
              if row[8] != "September":
                  empty_row = [""] * 9#creates an empty list of 9 values
                  app.replaceGridRow("g1", i, empty_row)#adds the empty
rows to the grid if the row[8]!= September
              else:
                  return TransferDailyTable(button)


   if (button == "PD_January" or button == "PD_February" or button ==
"PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June"
or button=="PD_July" or button=="PD_August" or button=="PD_September" or
button=="PD_October" or button=="PD_November" or button=="PD_December") and
today.month == 10:#checks if the current month is October
            for i in range(rowsg1):
              row = app.getGridRow("g1", i)#gets each row in the grid
              print(row)
              if all(val == "" for val in row):
                 continue # skip empty rows
              if row[8] != "October":
                  empty_row = [""] * 9#creates an empty list of 9 values
                  app.replaceGridRow("g1", i, empty_row)#add empty rows to
the grid if the row[8]!= October
              else:
                   return TransferDailyTable(button)

   if (button == "PD_January" or button == "PD_February" or button ==
"PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June"
or button=="PD_July" or button=="PD_August" or button=="PD_September" or
button=="PD_October" or button=="PD_November" or button=="PD_December") and
today.month == 11:#checks if the current month is November
            for i in range(rowsg1):
              row = app.getGridRow("g1", i)#gets each row in the grid
              print(row)
              if all(val == "" for val in row):
                 continue # skip empty rows
              if row[8] != "November":
                  empty_row = [""] * 9#creates an empty list of 9 values
                  app.replaceGridRow("g1", i, empty_row)#adds the empty
rows to the grid if the row[8]!= November
              else:
                   return TransferDailyTable(button)

   if (button == "PD_January" or button == "PD_February" or button ==
"PD_March" or button=="PD_April" or button=="PD_May" or button=="PD_June"
or button=="PD_July" or button=="PD_August" or button=="PD_September" or
button=="PD_October" or button=="PD_November" or button=="PD_December") and
today.month == 12:#checks if the current month is December
            for i in range(rowsg1):
              row = app.getGridRow("g1", i)#gets each row in the grid
              print(row)
              if all(val == "" for val in row):
                 continue # skip empty rows
              if row[8] != "December":
                  empty_row = [""] * 9#creates an empty list of 9 values
                  app.replaceGridRow("g1", i, empty_row)#adds the empty
rows to the grid if the row[8]!= December
```

```
            else:
                return TransferDailyTable(button)



def TransferDailyTable(button):
    seconds = time.time()#gets the current time in seconds
    local_time = time.ctime(seconds)#converts the time in seconds into a
readable format
    print("Local time:", local_time)#prints the readable time
    current_time = time.localtime()#gets the struct_time function in
format(year, month, day, hour, minute, second, weekday, day of the year,
daylight saving)
    today = datetime.date.today() #gets today's date
    print(today)
    hour = time.strftime("%H:%M:%S",current_time)#puts the current time in
H/M/S format
    str_hour = (str(hour))#makes hour variable a string
    str_hour=str_hour.strip()
    hour,minute,second=str_hour.split(":")#splits the time into single
integers of hour,minute and second
    if hour >="19" and hour <"20":#if hour is between 7pm and 8pm run the
funtion
        print("hey")
        rows = app.getGridRowCount("g1")#gets number of rows with data
        for i in range(rows):
            row = app.getGridRow("g1", i)
            print(row)
            print(rows)
            rowData=[]#empty list to transfer data
            for eachRow in row:
             if any(row):#for any row in the table
                        consultant_id = row[0]#assigns each row index to
the corresponding field name
                        ward_id = row[1]
                        Room_No = row[2]
                        Time_Of_Admission = row[3]
                        transfer_date = row[4]
                        transfer_time = row[5]
                        reason_for_transfer = row[6]
                        comments = row[7]
                        Month = row[8]

        conn = sqlite3.connect('PatientTransferAudit.db')#connect db
        cur = conn.cursor()
        cur.execute("INSERT INTO tbl_CurrentMonth (ConsultantID, WardID,
RoomNo, TimeOfAdmission, TransferDate, TransferTime, ReasonsForTransfer,
Comments , Month) VALUES (?, ?, ?, ?, ?, ?, ?, ?,?)", (consultant_id,
ward_id, Room_No, Time_Of_Admission, transfer_date, transfer_time,
reason_for_transfer,comments,Month))
        conn.commit()#commits all changes into the tbl_Daily in my database
        print("complete")



        for x in range(rows):
                data=app.getGridRow("g1",x)#for each row in the table store
values in the variable data
                rowData.append([data])#2D list
        print("row data",rowData)
```

```
        if row[8] == "January" or row[8] == "Febraury" or row[8] == "March"
or row[8] == "April" or row[8] == "May" or row[8] == "June" or row[8] ==
"July" or row[8] == "August" or row[8] == "September" or row[8] ==
"October" or row[8] == "November" or row[8] == "December":
            for aList in rowData:#for each list in rowData
                app.addGridRow("g2",aList[0])#adds each list in the 2D list
to current month

        print(rows)
        replace_g1 = app.getGridRowCount("g1")#count number of rows in g1
        for i in range(replace_g1):
            app.replaceGridRow("g1",i,"")#for each row replace g1 with
empty values



MedicalReason_Total = 0 #total for medical reasons
PatientRequest_Total = 0#total for patient requests
LateDischarge_Total = 0#total for late discharges
ConsultantRequest_Total = 0#total for consultant requests
SocialReason_Total = 0#total for social reasons
Other_Total = 0#total for other/unknown reasons
def AuditGraph():
    global MedicalReason_Total#reference the global variables
    global PatientRequest_Total
    global LateDischarge_Total
    global ConsultantRequest_Total
    global SocialReason_Total
    global Other_Total
    seconds = time.time()#gets the current time in seconds
    local_time = time.ctime(seconds)#converts the time in seconds into a
readable format
    print("Local time:", local_time)#prints the readable time
    current_time = time.localtime()#gets the struct_time function in
format(year, month, day, hour, minute, second, weekday, day of the year,
daylight saving)
    hour = time.strftime("%H:%M:%S",current_time)#puts the current time in
H/M/S format
    str_hour = (str(hour))#makes hour variable a string
    str_hour=str_hour.strip()
    hour,minute,second=str_hour.split(":")#splits the time into single
integers of hour,minute and second
    today = datetime.date.today()#gets todays date
    last_day_of_month = calendar.monthrange(today.year,today.month)[1]#gets
the exact number of days in the current month
    print(last_day_of_month)
    date = datetime.date(2023, 4, 30)
    if  today.day == last_day_of_month and today.month == 1 and (hour >=
"20" and hour < "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "January"
    elif today.day == last_day_of_month and today.month == 2 and (hour >=
"20" and hour < "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "February"
    elif today.day == last_day_of_month and today.month == 3 and (hour >=
"20" and hour < "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
```

```
        month = "March"
    elif today.day == last_day_of_month and today.month == 4 and (hour >=
"20" and hour < "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "April"
    elif today.day == last_day_of_month and today.month == 5 and (hour >=
"20" and hour < "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "May"
    elif today.day == last_day_of_month and today.month == 6 and (hour >=
"20" and hour < "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "June"
    elif today.day == last_day_of_month and today.month == 7 and (hour >=
"20" and hour < "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "July"
    elif today.day == last_day_of_month and today.month == 8 and (hour >=
"20" and hour < "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "August"
    elif today.day == last_day_of_month and today.month == 9 and (hour >=
"20" and hour < "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "September"
    elif today.day == last_day_of_month and today.month == 10 and (hour >=
"20" and hour < "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "October"
    elif today.day == last_day_of_month and today.month == 11 and (hour >=
"20" and hour < "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "November"
    elif today.day == last_day_of_month and today.month == 12 and (hour >=
"20" and hour < "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "December"

    print("today is last day")
    rows_g2 = app.getGridRowCount("g2")
    print(rows_g2)
    for x in range(rows_g2):
        data=app.getGridRow("g2",x)#for each row in the table store values
in the variable data
        print(data)
        if data and (data[8]==month):
            consultant_id = data[0]#assigns each row index to the
corresponding field name
            ward_id = data[1]
            Room_No = data[2]
            Time_Of_Admission = data[3]
            transfer_date = data[4]
            transfer_time = data[5]
            reason_for_transfer = data[6]
            comments = data[7]
            print(reason_for_transfer)



            if "1" in reason_for_transfer:
```

```
                MedicalReason_Total+=1 #increment medical reason tally by
one

            elif "2" in reason_for_transfer:
                PatientRequest_Total+=1#increment patient request reason
tally by one

            elif "3" in reason_for_transfer:
                LateDischarge_Total+=1#increment late discharge reason
tally by one

            elif "4" in reason_for_transfer:
                ConsultantRequest_Total+=1#increment consultant request
tally by one

            elif "5" in reason_for_transfer:
                SocialReason_Total+=1#increment social reason tally by one


            elif "6" in reason_for_transfer:
                Other_Total+=1#increment other reason tally by one


    print("total is " , SocialReason_Total)



new_consultantList = 0#the different consultants in the current month grid
new_reasonsList = 0#the number of patient transfers in the current month
grid
def ConsultantGraph():
    seconds = time.time()#gets the current time in seconds
    local_time = time.ctime(seconds)#converts the time in seconds into a
readable format
    print("Local time:", local_time)#prints the readable time
    current_time = time.localtime()#gets the struct_time function in
format(year, month, day, hour, minute, second, weekday, day of the year,
daylight saving)
    hour = time.strftime("%H:%M:%S",current_time)#puts the current time in
H/M/S format
    str_hour = (str(hour))#makes hour variable a string
    str_hour=str_hour.strip()
    hour,minute,second=str_hour.split(":")#splits the time into single
integers of hour,minute and second
    today = datetime.date.today()#gets todays date
    last_day_of_month = calendar.monthrange(today.year,today.month)[1]#gets
the exact number of days in the current month
    print(last_day_of_month)
    date = datetime.date(2023, 4, 30)
    global new_consultantList#refrences global variables to be used outisde
the function
    global new_reasonsList#refrences global variables to be used outisde
the function
    if  date.day == last_day_of_month and today.month == 1 and (hour >=
"20" and hour <="21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "January"
    elif today.day == last_day_of_month and today.month == 2 and (hour >=
"20" and hour <= "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
```

```
        month = "February"
    elif today.day == last_day_of_month and today.month == 3 and (hour >=
"20" and hour <= "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        print("hello")
        month = "March"
    elif today.day == last_day_of_month and today.month == 4 and (hour >=
"20" and hour <= "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "April"
    elif today.day == last_day_of_month and today.month == 5 and (hour >=
"20" and hour <= "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "May"
    elif today.day == last_day_of_month and today.month == 6 and (hour >=
"20" and hour <= "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "June"
    elif today.day == last_day_of_month and today.month == 7 and (hour >=
"20" and hour <= "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "July"
    elif today.day == last_day_of_month and today.month == 8 and (hour >=
"20" and hour <= "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "August"
    elif today.day == last_day_of_month and today.month == 9 and (hour >=
"20" and hour <= "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "September"
    elif today.day == last_day_of_month and today.month == 10 and (hour >=
"20" and hour <= "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "October"
    elif today.day == last_day_of_month and today.month == 11 and (hour >=
"20" and hour <= "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "November"
    elif today.day == last_day_of_month and today.month == 12 and (hour >=
"20" and hour <= "21"):#if the day is the last of the month and it is
between 8pm and 9pm count the totals for the current month
        month = "December"

    print("today is last day")
    rows_g2 = app.getGridRowCount("g2")
    print(rows_g2)
    for i in range(rows_g2):
        row=app.getGridRow("g2",i)
        consultants = []#makes an empty list for consultants
        reasons = []#makes an empty list for reasons
        if row and (row[8]==month):#if row[8] is equal to the corresponding
month
            for x in range(rows_g2):
                data = app.getGridRow("g2",x)#iterate through the grid and
get each row
                consultant_id = data[0]#assigns the first row index to the
consultant ID
                reason_for_transfer = data[6]
                consultants.append([consultant_id])#adds data to the new
list
```

```
                reasons.append([reason_for_transfer])#adds reasons to new
list
    print("consultants are", consultants)
    print("reasons are", reasons)
    new_consultantList = [x[0] for x in consultants if x != ['']]#remove
any empty values from the created list
    print(new_consultantList)

    new_reasonsList = [x[0] for x in reasons if x != ['']]#remove any empty
values from the created list
    print(new_reasonsList)

def PlotConsultantGraph():
    current_time = time.localtime()#gets the struct_time function in
format(year, month, day, hour, minute, second, weekday, day of the year,
daylight saving)
    hour = time.strftime("%H:%M:%S",current_time)#puts the current time in
H/M/S format
    str_hour = (str(hour))#makes hour variable a string
    str_hour=str_hour.strip()
    hour,minute,second=str_hour.split(":")#splits the time into single
integers of hour,minute and second
    today = datetime.date.today()#gets todays date
    last_day_of_month = calendar.monthrange(today.year,today.month)[1]#gets
the exact number of days in the current month
    date = datetime.date(2023, 4, 30)
    if today.day == last_day_of_month and hour >= "00": #and hour <
"21"):#if the day is the last of the month and it is between 8pm and 9pm
count the totals for the current month



        x = np.array(new_consultantList)#x-axis
        y = np.array(new_reasonsList)#y-axis


        plt.figure(figsize=(10,6))#size of bar graph
        plt.bar(x, y, color="orange")#plot bar graph and make the bars
orange
        plt.xlabel("Consultants")#labels x axis
        plt.ylabel("Number of patient transfers")#labels y axis
        plt.show()#show bar graph



def MonthlyGraph():
    current_time = time.localtime()#gets the struct_time function in
format(year, month, day, hour, minute, second, weekday, day of the year,
daylight saving)
    hour = time.strftime("%H:%M:%S",current_time)#puts the current time in
H/M/S format
    str_hour = (str(hour))#makes hour variable a string
    str_hour=str_hour.strip()
    hour,minute,second=str_hour.split(":")#splits the time into single
integers of hour,minute and second
    today = datetime.date.today()#gets todays date
    last_day_of_month = calendar.monthrange(today.year,today.month)[1]#gets
the exact number of days in the current month
    date = datetime.date(2023, 4, 30)
```

```
    if today.day == last_day_of_month and hour >= "00": #and hour <
"21"):#if the day is the last of the month and it is between 8pm and 9pm
count the totals for the current month


        x = np.array(["Medical Reason", "Patient request", "Late
Discharge","Consultant Request","Social Reason","Other"])#x-axis
        y = np.array([MedicalReason_Total, PatientRequest_Total,
LateDischarge_Total,ConsultantRequest_Total,SocialReason_Total,Other_Total]
)#y-axis

        plt.figure(figsize=(10,6))#size of bar graph
        plt.bar(x, y, color="lightblue")#plot bar graph and make the bars
orange
        plt.xlabel("Reason For Transfer")#labels x axis
        plt.ylabel("Number of patient transfers")#labels y axis
        plt.show()#show bar graph


max_value = 0#sets max_value to 0 so it can be used as a global variable
highest_category = 0#sets max_value to 0 so it can be used as a global
variable
common_consultant = 0#sets max_value to 0 so it can be used as a global
variable
def GraphSummary():
    current_time = time.localtime()#gets the struct_time function in
format(year, month, day, hour, minute, second, weekday, day of the year,
daylight saving)
    hour = time.strftime("%H:%M:%S",current_time)#puts the current time in
H/M/S format
    str_hour = (str(hour))#makes hour variable a string
    str_hour=str_hour.strip()
    hour,minute,second=str_hour.split(":")#splits the time into single
integers of hour,minute and second
    today = datetime.date.today()#gets todays date
    last_day_of_month = calendar.monthrange(today.year,today.month)[1]#gets
the exact number of days in the current month
    date = datetime.date(2023, 4, 30)
    if today.day == last_day_of_month and hour >= "00": #and hour <
"21"):#if the day is the last of the month and it is between 8pm and 9pm
count the totals for the current month
        global max_value#makes variables global
        global highest_category
        global common_consultant
        reasons_for_transfer = {"MedicalReason": MedicalReason_Total,
                                "PatientRequest": PatientRequest_Total,
                                "LateDischarge": LateDischarge_Total,
                                "ConsultantRequest":
ConsultantRequest_Total,
                                "SocialReason": SocialReason_Total,
                                "Other": Other_Total}#dictionary is made
to store the variables with its respected values

        highest_reason_for_transfer = max(reasons_for_transfer,
key=reasons_for_transfer.get)#key which gets the name of the reason for
transfer with the highest value
        highest_number =
reasons_for_transfer[highest_reason_for_transfer]#gets the number where the
reason for transfer contains the highest value
```

```
        max_value = highest_number#stores the highest_number in variable
max_value
        highest_category = highest_reason_for_transfer#stores the
highest_reason_for_transfer in highest_category
        common_consultant = max(new_reasonsList)
        print(max_value)
        print(highest_category)
        app.setLabel("maxVal", max_value)# updates max_value label
        app.setLabel("highest_category",highest_category)#updates highest
category label
        app.setLabel("Most Common Consultant",common_consultant)#updates
the Most Common Consultant label

def checkPass():

    app.hideEntry("visiblePassword")
    app.hideButton("eyehide")
    app.showButton("eye")


def showPass(btn):
    app.hideEntry("hiddenPassword")
    pwVar = app.getEntry("hiddenPassword")#sets hidden password variable to
password entered by the user
    app.showEntry("visiblePassword")
    app.setEntry("visiblePassword",pwVar)
    app.showButton("eyehide")
    app.hideButton("eye")



def hidePass(btn):
    app.hideEntry("visiblePassword")
    pwVar = app.getEntry("visiblePassword")#sets visible password to hidden
password entered by the user
    app.showEntry("hiddenPassword")
    app.setEntry("hiddenPassword",pwVar)
    app.showButton("eye")
    app.hideButton("eyehide")



#creates my interphase for the program
def createInterface():
################################################################Login
page#################################################################
    app.startSubWindow("Login")
    app.addLabel("title","DSW Transfer Audit" ,0,1)
    app.getLabelWidget("title").config(font=("","20",
"bold","underline"))#sets font size and makes title bold and underlined
    app.setLabelBg("title", "white")
    app.setLabelFg("title", "blue")
    app.addLabelEntry("Username :",1,1)
    app.addSecretLabelEntry("hiddenPassword",2,1)
    app.setLabel("hiddenPassword","Password :")
    app.addLabelEntry("visiblePassword",2,1)
    app.setLabel("visiblePassword","password:")
    app.addImageButton("eye",showPass,"eye.png",2,2)
    app.addImageButton("eyehide",hidePass,"eyehide.png",2,2)
    app.hideButton("eyehide")
```

```
    app.enableEnter(validLogin)#enables the enter button to be used in only
the login page
    checkPass()
    app.setEntryBg("Username :", "lightblue")
    app.setEntryBg("hiddenPassword", "lightblue")
    app.setEntryBg("visiblePassword", "lightblue")
    app.setFocus("Username :")
    app.setBg("white")
    app.startLabelFrame("", 0, 2)
    app.addImage("tlc", "tlc.png", 0,2)#Adds the company logo
    app.setImageSize("tlc", 200,200)
    app.setGuiPadding(10,10)
    app.stopLabelFrame()
    app.setFont(18)
    app.addButton("Login",press, 3, 0)
    app.addButton("Forgot Login",press, 3, 2)
    app.setButtonBg("Login", "lightblue")
    app.setButtonBg("Forgot Login", "lightblue")
    app.showSubWindow("Login")
    app.stopSubWindow()

###################################################################Main
Menu###############################################################
    app.startSubWindow("MainMenu")
    app.setSize("fullscreen")
    app.addLabel("title2","DSW Transfer Audit",0,0,2,2)
    app.getLabelWidget("title2").config(font=("","20",
"bold","underline"))#sets font size and makes title bold and underlined
    app.setLabelBg("title", "white")
    app.setLabelBg("title2", "white")
    app.setLabelFg("title2", "blue")
    app.setBg("white")
    app.addImage("tlc2", "tlc.png", 0,2)#Adds the company logo
    app.addNamedButton("Patient Data","ButtonPD",press, 3, 1)
    app.addNamedButton("Audit Graphs","ButtonAG",press,3, 0)
    app.setButtonBg("ButtonPD", "lightblue")
    app.setButtonBg("ButtonAG", "lightblue")
    app.setSticky("sw")
    app.setPadding([40,40])
    app.addButtons(["Log Out"],logout)
    app.setButtonBg("Log Out", "lightblue")



    app.stopSubWindow()



###################################################################Year
Display AG#########################################################
    app.startSubWindow("YearDisplayAG")
    app.setSize("fullscreen")
    app.addLabel("title3","Year")
    app.getLabelWidget("title3").config(font=("","20",
"bold","underline"))#sets font size and makes title bold and underlined
    app.setLabelBg("title3", "white")
    app.setLabelFg("title3", "blue")
    app.addImage("tlc3", "tlc.png", 0,2)#Adds the company logo
    app.setFont(18)
    app.setBg("white")
    app.setStretch("columns")
```

```
    app.addNamedButton("2023","AG_2023",press,1,0)#Adds named buttons for
the years
    app.addNamedButton("2024","AG_2024",press,2,0)
    app.addNamedButton("2025","AG_2025",press,3,0)
    app.addNamedButton("2026","AG_2026",press,4,0)
    app.setButtonBg("AG_2023", "lightblue")
    app.setButtonBg("AG_2024", "lightblue")
    app.setButtonBg("AG_2025", "lightblue")
    app.setButtonBg("AG_2026", "lightblue")
    app.setSticky("sw")
    app.setPadding([40,40])
    app.addNamedButton("Back","backYAG",press)
    app.setButtonBg("backYAG", "lightblue")



    app.stopSubWindow()



##################################################################Year
Display PD#########################################################################
    app.startSubWindow("YearDisplayPD")
    app.setSize("fullscreen")
    app.addLabel("title5","Year")
    app.getLabelWidget("title5").config(font=("","20",
"bold","underline"))#sets font size and makes title bold and underlined
    app.setLabelBg("title5", "white")
    app.setLabelFg("title5", "blue")
    app.addImage("tlc5", "tlc.png", 0,2)#Adds the company logo
    app.setFont(18)
    app.setBg("white")
    app.setStretch("columns")
    app.addNamedButton("2023","PD_2023",press,1,0)#Adds named buttons for
the years
    app.addNamedButton("2024","PD_2024",press,2,0)
    app.addNamedButton("2025","PD_2025",press,3,0)
    app.addNamedButton("2026","PD_2026",press,4,0)
    app.setButtonBg("PD_2023", "lightblue")
    app.setButtonBg("PD_2024", "lightblue")
    app.setButtonBg("PD_2025", "lightblue")
    app.setButtonBg("PD_2026", "lightblue")
    app.setSticky("sw")
    app.setPadding([40,40])
    app.addNamedButton("Back","backYPD",press)
    app.setButtonBg("backYPD", "lightblue")

    app.stopSubWindow()



##################################################################Month
Display AG#########################################################################
    app.startSubWindow("MonthDisplayAG")
    app.setSize("fullscreen")
    app.addLabel("title4","Months")
    app.getLabelWidget("title4").config(font=("","20",
"bold","underline"))#sets font size and makes title bold and underlined
    app.setLabelBg("title4", "white")
```

```
    app.setLabelFg("title4", "blue")
    app.addImage("tlc8", "tlc.png", 0,2)#Adds the company logo
    app.addNamedButton("January","AG_January",monthGraph,1,0)
    app.addNamedButton("February","AG_February",monthGraph,2,0)
    app.addNamedButton("March","AG_March",monthGraph,3,0)
    app.addNamedButton("April","AG_April",monthGraph,4,0)
    app.addNamedButton("May","AG_May",monthGraph,5,0)
    app.addNamedButton("June","AG_June",monthGraph,6,0)
    app.addNamedButton("July","AG_July",monthGraph,7,0)
    app.addNamedButton("August","AG_August",monthGraph,8,0)
    app.addNamedButton("September","AG_September",monthGraph,9,0)
    app.addNamedButton("October","AG_October",monthGraph,10,0)
    app.addNamedButton("November","AG_November",monthGraph,11,0)
    app.addNamedButton("December","AG_December",monthGraph,12,0)
    app.setButtonBg("AG_January", "lightblue")#Adds lightblue colour
    app.setButtonBg("AG_February", "lightblue")
    app.setButtonBg("AG_March", "lightblue")
    app.setButtonBg("AG_April", "lightblue")
    app.setButtonBg("AG_May", "lightblue")
    app.setButtonBg("AG_June", "lightblue")
    app.setButtonBg("AG_July", "lightblue")
    app.setButtonBg("AG_August", "lightblue")
    app.setButtonBg("AG_September", "lightblue")
    app.setButtonBg("AG_October", "lightblue")
    app.setButtonBg("AG_November", "lightblue")
    app.setButtonBg("AG_December", "lightblue")
    app.setSticky("sw")
    app.setPadding([40,40])
    app.addNamedButton("Back","backMAG",press)
    app.setButtonBg("backMAG", "lightblue")



    app.stopSubWindow()


##################################################################Month
Display PD#######################################################################
    app.startSubWindow("MonthDisplayPD")
    app.setSize("fullscreen")
    app.addLabel("title6","Months")
    app.getLabelWidget("title6").config(font=("","20",
"bold","underline"))#sets font size and makes title bold and underlined
    app.setLabelBg("title6", "white")
    app.setLabelFg("title6", "blue")
    app.addImage("tlc6", "tlc.png", 0,2)#Adds the company logo
    app.setFont(18)
    app.setBg("white")
    app.addNamedButton("January","PD_January",press,1,0)#Adds named button
to the buttons
    app.addNamedButton("February","PD_February",press,2,0)
    app.addNamedButton("March","PD_March",press,3,0)
    app.addNamedButton("April","PD_April",press,4,0)
    app.addNamedButton("May","PD_May",press,5,0)
    app.addNamedButton("June","PD_June",press,6,0)
    app.addNamedButton("July","PD_July",press,7,0)
    app.addNamedButton("August","PD_August",press,8,0)
    app.addNamedButton("September","PD_September",press,9,0)
    app.addNamedButton("October","PD_October",press,10,0)
```

```
    app.addNamedButton("November","PD_November",press,11,0)
    app.addNamedButton("December","PD_December",press,12,0)
    app.setButtonBg("PD_January", "lightblue")
    app.setButtonBg("PD_February", "lightblue")
    app.setButtonBg("PD_March", "lightblue")
    app.setButtonBg("PD_April", "lightblue")
    app.setButtonBg("PD_May", "lightblue")
    app.setButtonBg("PD_June", "lightblue")
    app.setButtonBg("PD_July", "lightblue")
    app.setButtonBg("PD_August", "lightblue")
    app.setButtonBg("PD_September", "lightblue")
    app.setButtonBg("PD_October", "lightblue")
    app.setButtonBg("PD_November", "lightblue")
    app.setButtonBg("PD_December", "lightblue")
    app.setSticky("sw")
    app.setPadding([40,40])
    app.addNamedButton("Back","backMPD",press)
    app.setButtonBg("backMPD", "lightblue")

    app.stopSubWindow()



##################################################################Patient
Transfer
Databases#########################################################################
    app.startSubWindow("PatientTransferDatabases")
    app.setSize("fullscreen")
    app.addImage("tlc7", "tlc.png", 0,2)#Adds the company logo
    app.addLabel("title7","Patient Transfer Databases",0,0,2,2)
    app.getLabelWidget("title7").config(font=("","20",
"bold","underline"))#sets font size and makes title bold and underlined
    app.setLabelBg("title7", "white")
    app.setLabelFg("title7", "blue")
    app.setFont(18)
    app.setBg("white")
    app.addButton("Daily",press,1,0)
    app.addButton("Current Month Database",press,1,1)
    app.setButtonBg("Daily","lightblue")
    app.setButtonBg("Current Month Database","lightblue")
    app.setSticky("sw")
    app.setPadding([40,40])
    app.addNamedButton("Back","BackPTD",press)
    app.setButtonBg("BackPTD","lightblue")
    app.stopSubWindow()



##################################################################Daily
Database#########################################################################
    app.startSubWindow("DailyDatabase")
    app.setSize("fullscreen")
    app.addLabel("title8","Daily",0,1)#Title of the table
    app.getLabelWidget("title8").config(font=("","20",
"bold","underline"))#sets font size and makes title bold and underlined
    app.setLabelBg("title8", "white")
    app.setLabelFg("title8", "blue")
    app.setFont(18)
    app.setBg("white")
    app.addLabel("title9","Consultants Key",0,10,8).config(font=("","20",
"bold","underline"))
```

```
    app.setSticky("e")
    app.setFont(10)
    #Adds all the consultants under the consultants key
    app.addListBox("ConsultantsDaily", ["Mr Bhogal - MB",
                                        "Mr Westscott - MW",
                                        "Mr Mohammed - M",
                                        "Mr Pringle - P",
                                        "Mr Hamilton - RH",
                                        "Mr Lee - RL",
                                        "Mr Wong - RSW",
                                        "Mr Jain - J",
                                        "Mr Shah - SS",
                                        "Mr Trikha - ST",
                                        "Mr Saurabh Jain - SJ",
                                        "Mr Williamson - W",
                                        "Mr Kulkarni - AK",
                                        "Mr Mearza - AM",
                                        "Mr Mitry - DM",
                                        "Mr Elgohary - E",
                                        "Mr Ahmed - FA",
                                        "Mr Ratnarajan - GR",
                                        "Mr Henderson - HH",
                                        "Mr Duguid - IGD",
                                        "Mr Dowler - JD",
                                        "Prof Lim - Lim",
                                        "Prof Jackson",
                                        "Mr Khan - JK",
                                        "Mr LaidLaw - L",
                                        "Miss Goawalla - G",
                                        "Miss Jain - J",
                                        "Miss Mensah - MM",
                                        "Miss Zakir - RZ",
                                        "Miss Saw - S",
                                        "Mr El-Amir - AEA",
                                        "Mr Mearza - AM",
                                        "Mr Mitry - DM",
                                        "Mr Elgohary - E",
                                        "Mr F Ahmed - FA"],1,10,10,8)
    app.setLabelBg("title9", "white")
    app.setLabelFg("title9", "blue")
    app.setSticky("nw")
    app.addLabel("title10","Transfer reasons
key",0,0).config(font=("","20", "bold","underline"))
    app.setLabelBg("title10", "white")
    app.setLabelFg("title10", "blue")
    app.addLabel("medical reason","1 - Medical Reason",1,0)#Adds the
different reasons for patient transfers to the user interface
    app.addLabel("patient request","2 - Patient Request",2,0)
    app.addLabel("late discharge","3 - Late discharge",3,0)
    app.addLabel("consultant request","4 - Consultant Request",4,0)
    app.addLabel("social reason","5 - Social Reason",5,0)
    app.addLabel("other","6 - Other",6,0)
    app.setFont(20)
    #creates a grid where data can be inputted
    app.setSticky("ew")
    app.addGrid("g1",

    [["ConsultantID", "WardID","RoomNo",
"TimeOfAdmission","TransferDate","TransferTime","ReasonForTransfer","Commen
ts","Month"],
```

```
        ["","","","","","","","","",""],

        ["","","","","","","","","",""],

        ["","","","","","","","","",""],

        ["","","","","","","","","",""]],
#Allows for the features of the grid to exist and when user right-clicks on
an entry box a sub-menu pops up which allows for changes to the grid
        action=SaveTableg1,showMenu=True,actionButton
="save",addRow=addGridRowg1,row=1,column=1, rowspan=9, colspan=9)
        app.setSticky("sw")
        app.addNamedButton("Back","BackDD",press,10,0)
        app.setButtonBg("BackDD","lightblue")



        app.stopSubWindow()



##################################################################Current
Month
Database#########################################################################
        app.startSubWindow("CurrentMonthDatabase")
        app.setSize("fullscreen")
        app.addLabel("title11","Current Month",0,1)#Title of the table
        app.getLabelWidget("title11").config(font=("","20",
"bold","underline"))#sets font size and makes title bold and underlined
        app.setLabelBg("title11", "white")
        app.setLabelFg("title11", "blue")
        app.setFont(18)
        app.setBg("white")
        app.addLabel("title12","Consultants Key",0,10,8).config(font=("","20",
"bold","underline"))
        app.setLabelBg("title12", "white")
        app.setLabelFg("title12", "blue")
        app.setSticky("e")
        app.setFont(10)
        #Adds all the consultants under the consultants key
        app.addListBox("ConsultantsMonth", ["Mr Bhogal - MB",
                                            "Mr Westscott - MW",
                                            "Mr Mohammed - M",
                                            "Mr Pringle - P",
                                            "Mr Hamilton - RH",
                                            "Mr Lee - RL",
                                            "Mr Wong - RSW",
                                            "Mr Jain - J",
                                            "Mr Shah - SS",
                                            "Mr Trikha - ST",
                                            "Mr Saurabh Jain - SJ",
                                            "Mr Williamson - W",
                                            "Mr Kulkarni - AK",
                                            "Mr Mearza - AM",
                                            "Mr Mitry - DM",
                                            "Mr Elgohary - E",
                                            "Mr Ahmed - FA",
                                            "Mr Ratnarajan - GR",
                                            "Mr Henderson - HH",
                                            "Mr Duguid - IGD",
                                            "Mr Dowler - JD",
```

```
                                    "Prof Lim - Lim",
                                    "Prof Jackson",
                                    "Mr Khan - JK",
                                    "Mr LaidLaw - L",
                                    "Miss Goawalla - G",
                                    "Miss Jain - J",
                                    "Miss Mensah - MM",
                                    "Miss Zakir - RZ",
                                    "Miss Saw - S",
                                    "Mr El-Amir - AEA",
                                    "Mr Mearza - AM",
                                    "Mr Mitry - DM",
                                    "Mr Elgohary - E",
                                    "Mr F Ahmed - FA"],1,10,10,8)

    app.setSticky("nw")
    app.addLabel("title13","Transfer reasons 
key",0,0).config(font=("","20", "bold","underline"))
    app.setLabelBg("title13", "white")
    app.setLabelFg("title13", "blue")
    app.addLabel("medical reason (month)","1 - Medical Reason",1,0)
    app.addLabel("patient request (month)","2 - Patient Request",2,0)
    app.addLabel("late discharge (month)","3 - Late discharge",3,0)
    app.addLabel("consultant request (month)","4 - Consultant Request",4,0)
    app.addLabel("social reason (month)","5 - Social Reason",5,0)
    app.addLabel("other (month)","6 - Other",6,0)
    app.setFont(20)
    app.setSticky("ew")
    app.addGrid("g2",

    [["ConsultantID", "WardID","RoomNo",
"TimeOfAdmission","TransferDate","TransferTime","ReasonForTransfer","Commen
ts","Month"],

    ["","","","","","","","",""],

    ["","","","","","","","",""],

    ["","","","","","","","",""],

    ["","","","","","","","",""]],
#Allows for the features of the grid to exist and when user right-clicks on
an entry box a sub-menu pops up which allows for changes to the grid

action=SaveTableg2,showMenu=True,actionButton="save",addRow=addGridRowg2,ro
w=1,column=1, rowspan=9, colspan=9)
    app.setSticky("sw")
    app.addNamedButton("Back","BackCD",press,10,0)
    app.setButtonBg("BackCD","lightblue")


    app.stopSubWindow()


###############################################################Audit
Graphs##############################################################
    app.startSubWindow("AuditGraphs")
    app.setSticky("n")
    app.setPadding([40,40])
    app.addLabel("title14","Audit Graphs",0,1)
```

```
    app.getLabelWidget("title14").config(font=("","20",
"bold","underline"))#sets font size and makes title bold and underlined
    app.setLabelBg("title14", "white")
    app.setLabelFg("title14", "blue")
    app.setSize("fullscreen")
    app.addImage("tlc9", "tlc.png", 0,2)#Adds the company logo
    app.setFont(18)
    app.setPadding([40,40])
    app.addNamedButton("View Graph Summary","VGS",press,1,1)#adds the View
Graph Summary button for any month
    app.setSticky("e")
    app.addNamedButton("Consultant Graph","CG",press,1,2)#adds the
consultant graph button for any month
    app.setButtonBg("CG","lightblue")
    app.setButtonBg("VGS","lightblue")
    app.setBg("white")
    app.setSticky("sw")
    app.setPadding([40,40])
    app.addNamedButton("Back","BackAGP",press)
    app.setButtonBg("BackAGP","lightblue")

###################################################################Graph
Summary###################################################################
    app.startSubWindow("GraphSummary")
    app.addLabel("title15","Graph Summary",0,0).config(font=("","20",
"bold","underline"))#sets font size and makes title bold and underlined
    app.setLabelBg("title15", "white")
    app.setLabelFg("title15", "blue")
    app.setSize("fullscreen")
    app.addImage("tlc10", "tlc.png", 0,2)#Adds the company logo
    app.addLabel("transfers","highest number of transfers = ",1,0)
    app.addLabel("maxVal",max_value,1,1)#adds maximum value of the reasons
for transfers to the gui
    app.setLabelBg("transfers", "white")
    app.setLabelFg("transfers", "blue")
    app.addLabel("category","highest catgory of transfers = ",2,0)
    app.addLabel("highest_category",highest_category,2,1)#adds the highest
category of the reasons for transfers to the gui
    app.setLabelBg("category", "white")
    app.setLabelFg("category", "blue")
    app.addLabel("common consultant","highest number of transfers for one
consultant = ",3,0)
    app.setLabelBg("common consultant", "white")
    app.setLabelFg("common consultant", "blue")
    app.addLabel("Most Common Consultant", common_consultant,3,1)#shows the
number of patient transfers for the most common consultant
    app.addLabel("Consultant name","Please see the Consultant Graph for
more information on the most common consultant")
    app.setLabelBg("Consultant name", "white")
    app.setLabelFg("Consultant name" , "blue")
    app.setFont(18)
    app.setBg("white")
    app.setSticky("sw")
    app.setPadding([40,40])
    app.addNamedButton("Back","BackGS",press)
    app.setButtonBg("BackGS","lightblue")

    app.stopSubWindow()

createInterface()
```