Candidate Number:
3097

# Prototype:

## 4.1: Aims for the prototype

The aim of this prototype is to get a base for the program that I will be developing, it will also allow me to develop the functionality and basic GUI of the program to see how it functions. Once I am finished with the prototype, I will adjust it into a finished project with functional tables for everything. It will also help my client understand what their program will function like in case anything needs to be adjusted. This section is important in creating the backbone to my program and will also help me see how a lot of the staff members feel about he choices I have made. It will also test the aims that I have created in my project and will let me see if I complete these objectives. This is a very important section in the development process of the program that I am creating.

## 4.2: Areas that will be included in the prototype

### Login tab

The base menu for everything, will be the first window you see when you start the program. Allows the user to login and it will adjust via views which tables they are allowed to see and what they are allowed to do with them. I will use this in the future to connect to tables I add later in the final program. This menu must be clear and easy to use and should be inspired by the desk-based solutions that I have seen within the Investigation section. I hope to make this very intuitive and for all the tabs that this frame switches to, to be easy to navigate through and change.

### Client's tab

1. Search for a client
2. Add a client
3. Delete a client
4. View all clients

This will show a table of all the clients at the company. It will be shown once the user is logged in. It will allow the user to create and delete and view users. As well as search for a specific keyword under a specified column, this search result will be outputted both in the program and also within a text file. I will try to focus on making all the functions be as intuitive and easy to use as possible with as little hassle for the user while minimising the number of clicks required to complete a function. I will use this a basic understanding on how I will show the data in the final program and how I plan to create and adjust records.

### Staff's tab

1. Search for a staff member
2. Add a staff member
3. Delete a staff member
4. View all staff members

This will show a table of all staff members at the company and can be accessed by clicking the tab for staff. Similar to the client, it will allow the user to create, delete and search for staff members. These are very important features and I hope to that the use of these features will all be intuitive and easy to navigate for the user as that is what my focus is on when creating this Prototype.

### Database creation

I will create a rough database for the staff and clients table. I will use the same structure that I used within the Design section and thus they will follow the rules of third normalised form, I hope to significantly minimise any data redundancy throughout the program and increase efficiency where possible.

## 4.3: Areas to be omitted in the prototype

### Remaining tabs

I will be omitting the creation a tab for medical records, transactions and appointments, this is because the program I am creating is just the Prototype and not the final product and thus I just want to get an idea of what the program may be like in the future. I still have two tabs in my program as that will still tell me how more than one table in the program will function and how they will intertwine with each other.

### Edit records

I will also be omitting the ability to edit records. I will already have the ability to add, delete, search and view records and I feel like that should be a good enough idea to understand how the program that I plan on creating will work and function. Some of the functions that I will be creating will already be similar to the edit function, such as the ability to add a record and thus I feel like I should omit this area and keep the prototype simple, so I can just understand how the program works instead of seeing every single function within it.

### Validation

Finally, I will omit validation. Validation in my program isn't necessary to see how the program will function. Validation is time consuming and unnecessary in seeing how my programmed solution should look and function and as a result I will not be implementing validation into my prototype. I believe I should focus more on how the program should work than instead preventing invalid data from being put into the program.

## 4.4: Explanation of areas of the program to be included in the prototype

The main things that I will be holding in my program will be the tables for user information and its ability to manipulate it. The main purpose of my prototype will be to see how navigation works in my program, how to connect my database to tkinter, how to implement views into my program, and how manipulating data in my program would be used. I believe the areas that I have included in my prototype will successfully cover these aims.

The program will also be used to see if the success criteria that I had created in the Investigation stage is realistic, creating the prototype will see how the functions will look and work and will allow me to have a good base of my program. I also would like to see how the GUI will actually look and I will try to compare it to the design stage. I also believe that I have chosen a suitable number of areas to be included in my prototype and that these areas will make the software development section significantly easier.

Overall, I think I have chosen good areas to include and omit in my prototype and I hope that these will be the bedrock for my software development. I will also implement realistic data into my program to see how my program works with it.

## 4.5: Map of windows shown

Login screen

- Client page
  - View clients
  - Create a client
  - Delete a client
  - Search for a client
- Staff page
  - View staff
  - Create a staff
  - Delete a staff member
  - Search for a staff member

Candidate Number:
3097

## 1.4: Database

```
# coding=utf-8
import sqlite3

db = sqlite3.connect('clinic.db')
cursor = db.cursor()
"""
creates the entire database with sutiable columns
"""
cursor.execute('''CREATE TABLE IF NOT EXISTS clients(
                ClientID INTEGER PRIMARY KEY AUTOINCREMENT,
                Prefix TEXT,
                FirstName TEXT,
                Surname TEXT,
                DOB TEXT,
                Telephone TEXT,
                Address TEXT,
                Postcode TEXT,
                ''')

cursor.execute('''CREATE TABLE IF NOT EXISTS staff(
                StaffID INTEGER PRIMARY KEY AUTOINCREMENT
                Prefix TEXT,
                FirstName TEXT,
                Surname TEXT,
                DOB TEXT,
                Telephone TEXT,
                Address TEXT,
                Postcode TEXT,
                Username TEXT unique,
                Position TEXT,
                LoggedIn BOOLEAN,
                Password TEXT)
                ''')

db.commit()

cursor.execute("""UPDATE staff SET LoggedIn = ?""", ("False",)) #puts loggedin to false

db.commit()
```

Creates a database with 2 tables (more will be added in the final program) and specifies the data type of each column. Also sets LoggedIn to False as all staff are currently not logged in. LoggedIn will be used for views to find out which user is currently logged in and sets views based on the that users' position.

Candidate Number:
3097

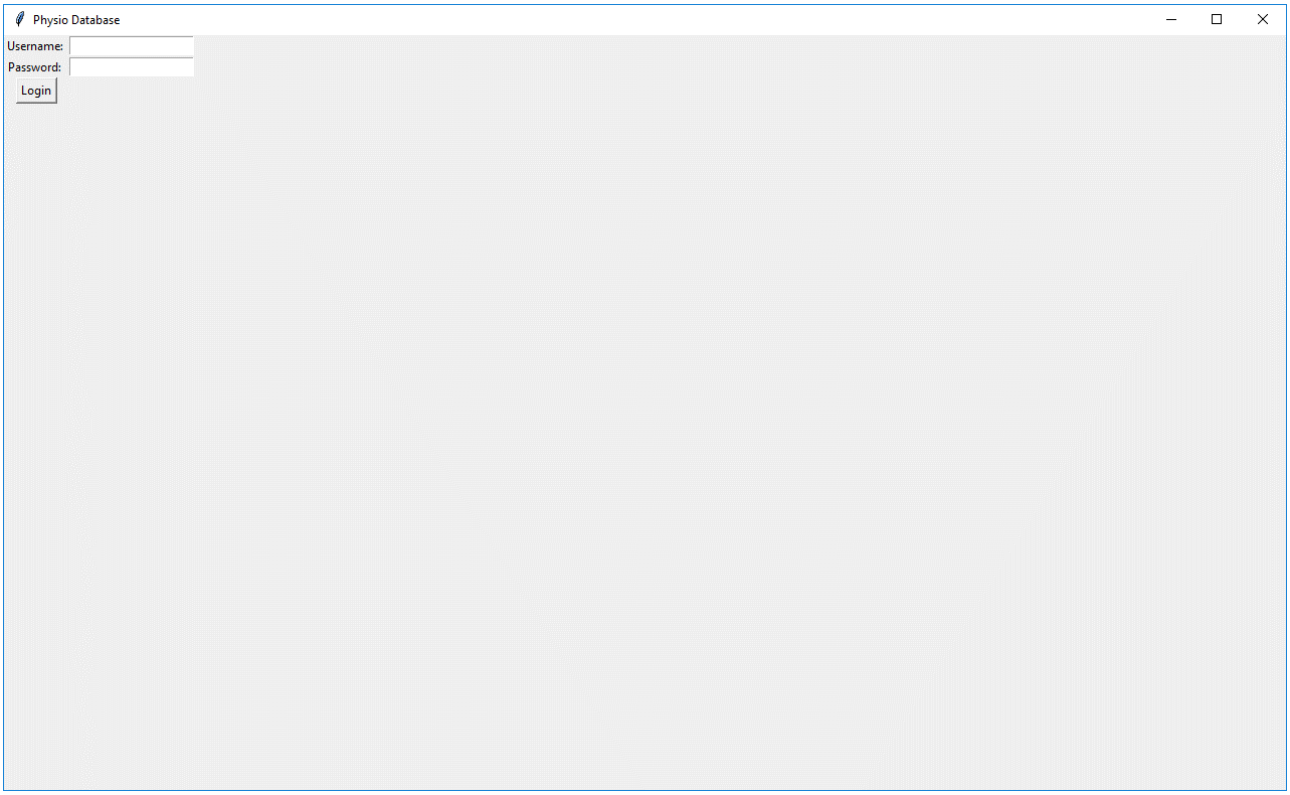
This code creates the following (it does not create the records, this has been data inputted by me):

| ClientID | Prefix | FirstName | Surname | DOB | Telephone | Address | Postcode |
|---|---|---|---|---|---|---|---|
| Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | Mr | Kai | Holloway | 1934-07-11 | 07863701077 | 23 Ermin Street | RH18 9WX |
| 2 | Mr | Andrew | Rove | 1938-04-12 | 07920991649 | 19 Middlewich... | TN5 9JE |
| 3 | Mrs | Aaliyah | Price | 1978-04-30 | 07935747494 | 34 Bishopthor... | SA44 7LU |
| 4 | Ms | Paisley | Russel | 2002-08-26 | 07830280919 | 12 Sutton Wic... | CA14 1DQ |
| 5 | Mr | Barack | Obama | 1997-10-12 | 07702989653 | 98 Newport R... | IV12 5WH |
| 6 | Mr | Cathal | Gorman | 1954-09-01 | 07038789458 | 49 Ash Lane | EX23 7WL |
| 7 | Dr | Abigail | Dunn | 1977-08-31 | 07777827240 | 5 Manor Close | LL54 0UD |
| 8 | Mr | Leo | Barker | 1999-03-16 | 07956646717 | 69 Cefn Road | RG7 4FJ |
| 9 | Mrs | Jasmine | Powell | 1941-01-04 | 07728974291 | 72 South Cres... | DY9 9XX |
| 10 | Mr | Dayyan | OBrien | 2001-01-31 | 07767862981 | 10 Windsor Hill | BT34 1ER |
| 12 | Dr | Steve | Jobs | 1980-01-02 | 07605407812 | 5 Apple Lane | RZ7 2GS |

| | StaffID | Prefix | FirstName | Surname | DOB | Telephone | Address | Postcode | Username | Position | LoggedIn | Password |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | 1 | Dr | Kirsten | Freener | 2001-01-01 | 07901611233 | 10 Windsor Hill | BT341ER | owneruser | Owner | 1 | ownerpass |
| 2 | 2 | Mr | Scott | Randall | 1946-04-22 | 07916795451 | 92 Sea Road | CO4 3AX | itusername | IT | False | itpassword |
| 3 | 3 | Dr | Freya | Winter | 1951-08-31 | 07750976324 | 1 Foregate St... | KT11 1EF | nurseuser | Nurse | False | nursepass |
| 4 | 4 | Dr | Adam | Pochinki | 1966-08-30 | 07744443297 | 97 Crown Str... | SO32 5EL | physiouser | Physiotherapist | False | physiopass |
| 5 | 5 | Mr | Jay | Baldwin | 1977-01-03 | 07814692250 | 77 East Street | SN9 2TX | receptionistuser | Receptionist | False | receptionistpass |
| 6 | 6 | Mrs | Laura | Lyons | 1959-10-29 | 07825964889 | 14 Warren St | YO8 2NG | llyons886 | IT | False | nie7ohteiNu |
| 7 | 7 | Ms | Chloe | Coleman | 1973-11-28 | 07005303213 | 21 Folestone ... | BA14 0PA | Starman | Nurse | False | chaiwu0H |
| 8 | 8 | Dr | Hayden | Oliver | 1972-02-22 | 07819248729 | 19 City Walls Rd | SY7 2DD | Agescits1972 | Physiotherapist | False | Su5jaht9ie |
| 9 | 9 | Mr | Jamie | Webb | 1986-04-25 | 07842073019 | 43 Bouverie R... | SY4 3UB | Haters1986 | Receptionist | False | phahz1Tee3 |

## 4.6: Login screen

This menu will be shown once you start the program, it will give you two entry boxes: username & password. It will then check these passwords matched with the username. If they are valid it will log the client in.




Username: You will input your username in the entry box to the right of 'Username:'

Password: You will input your password in the entry box to the right of 'Password:'

Login: Once both inputs have been entered you press this button, if it is correct it will change the screen to the client's page, however if it is incorrect it will give a messagebox error saying, "Username and/or password incorrect"

Candidate Number:
3097

```
class LoginFrame(tk.Frame):
    """
    creates the login page frame, allows the user to input the username
    and password and when the button is pressed it switches the frame
    to HomepageTab and shows all the tab options
    """

    def __init__(self, master):
        tk.Frame.__init__(self, master)
        self.master = master
        tk.Label(self, text="Username: ").grid(row=0, column=0, padx = (500,0), pady = (250,0))
        tk.Label(self, text="Password: ").grid(row=1, column=0, padx = (500,0))
        self.un_entry = tk.Entry(self)
        self.un_entry.grid(row=0, column=1, padx = (0,500), pady = (250,0))
        self.pw_entry = tk.Entry(self, show = '*') #use stars for password
        self.pw_entry.grid(row=1, column=1, padx = (0,500))

        self.username = self.un_entry.get()
        self.password = self.pw_entry.get()

        tk.Button(self, text="Login", command=self.check_login).grid

    def check_login(self):
        """
        checks if the username and password is
        correct and then sets LoggedIn = True
        """

        username = self.un_entry.get()
        password = self.password = self.pw_entry.get()
        cursor.execute("""SELECT LoggedIn FROM staff WHERE Username = ? AND Password = ?""", (username, password))
        result = cursor.fetchone()
        if result:
            cursor.execute("""UPDATE staff SET LoggedIn = ? WHERE Username = ? AND Password = ?""",
                           (True, username, password))
            db.commit()
            self.master.switch_frame(TabFrame) #puts tab frame on top
        else:
            messagebox.showinfo("Error", "Username and/or password incorrect")
```

The following checks the SQLite database to see if the inputs are found within the database, if they are it will set LoggedIn = True and then switch the frame to the one containing tabs

```
class SwitchFrame(tk.Tk):
    """
    the initially run class, it creates the ability
    frame of the window without creating a new one
    as well as making the initial frame LoginPage
    """

    def __init__(self):
        tk.Tk.__init__(self)
        self._frame = None
        self.switch_frame(LoginFrame)
        self.geometry("1275x750")
        self.title("Physio Database")

    def switch_frame(self, frame_class):
        """destroys the current frame and replaces i
        new_frame = frame_class(self)
        if self._frame is not None:
            self._frame.destroy()
        self._frame = new_frame
        self._frame.grid(row=0, column=0)
```

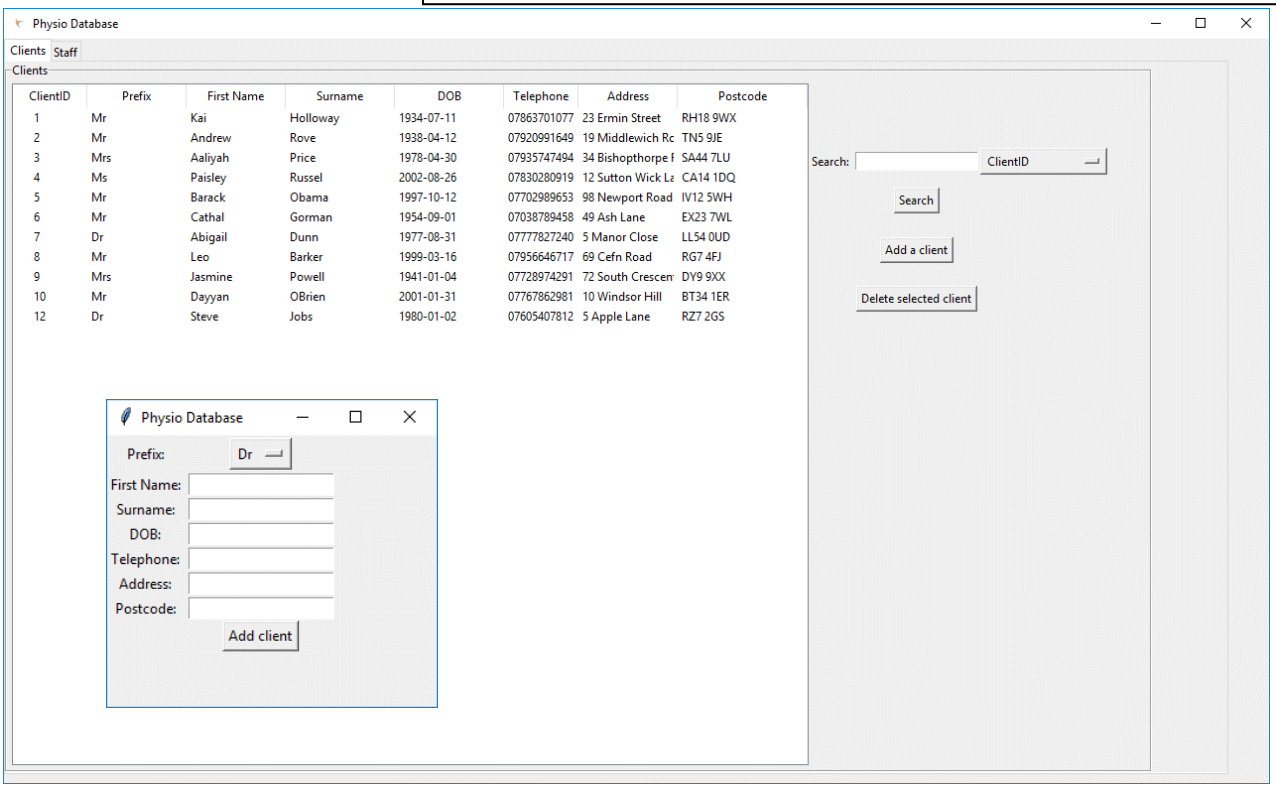
switch_frame will run once the user has logged in, what it will do is it will take the current frame (LoginFrame), which can just be called self as it is the current one and it will replace it with TabFrame. It will do this by destroying the Login window, I did this instead of stacking one above the other, as destroying it which will prevent it showing when you show all windows currently open.

```
class TabFrame(tk.Frame):
    """
    this is the frame which holds all the tabs, this allows users
    to select which tab they want to use
    """

    def __init__(self, parent, *args, **kwargs):
        tk.Frame.__init__(self, parent, *args, **kwargs)

        notebook = ttk.Notebook(parent)

        cursor.execute("""SELECT Position FROM staff WHERE LoggedIn = ?""",(True,))
        pos = cursor.fetchone()[0] #does views to see which tabs to show
        if pos == 'IT                    '  or pos == 'Owner                 ' or pos == 'Nurse                 '
            notebook.add(ClientsFrame(notebook), text='Clients') #shows client tab
        if pos == 'IT                    '  or pos == 'Owner                 ':
            notebook.add(StaffFrame(notebook), text='Staff') #shows staff tab


        notebook.grid(row=0, colu
```

This will create a notebook and add the ClientFrame and StaffFrame to this, this basically means it will add tabs named clients and staff when changing the frame as we don't want the tabs showing prior to logging in. In addition to this, the if statements are views. When logging in, it sets the LoggedIn of the record to True, this is then checked in views where it calls any record that has LoggedIn = True, so if, for example the owner was logged in, its position is in both if statements so both the client and staff frame show. If, however you are a nurse, only the clients frame would show. Please note this image is cropped as there are more positions in the clients view but it would not fit the screenshot.

## 4.7: Client screen

The below will show when logging in as an owner, it will show all records (currently none) as well as buttons that allow you to search, add and delete clients.



Search: Allows you to input the keyword that you would like to search by. And below that you can click to show all results that contain that keyword, this will be shown on the right-hand side of the screen. It will also output everything into a text file.

Search drop down: This allows you to choose which column you would like to search by, these are:

ClientID,

Prefix,

First Name,

Surname,

DOB,

Telephone,

Address,

Postcode

Add a client: This will open a small window which allows you to enter a client and allows you to input data for each of the columns available par ClientID, which is created automatically by the program.

Delete selected client: This will delete the client you highlight within the table on the right-hand side

Within add a client there is:

Prefix: You will select your prefix in the drop-down menu to the right of 'Prefix:'

First name: You will input your first name in the entry box to the right of 'First name:'

Surname: You will input your surname in the entry box to the right of 'Surname:'

DOB: You will input your DOB in the entry box to the right of 'DOB:'

Telephone: You will input your telephone in the entry box to the right of 'Telephone:'

Address: You will input your address in the entry box to the right of 'Address :'

Postcode: You will input your postcode in the entry box to the right of 'Postcode:'

Add client: Once all inputs have been entered it will add the client.

If all the above are correct, it will then add the client and display them on the main window.

Candidate Number:
3097

```
OPTIONS = [
    "ClientID               ",
    "Prefix                   ",
    "First Name           ",
    "Surname              ",
    "DOB                      ",
    "Telephone          ",
    "Address                ",
    "Postcode             ",
]  # creates the options for the dropdown menu on serach

variable = StringVar(clients_frame)
variable.set(OPTIONS[0])  # starts the dropdown menu on ClientID

# Set the treeview
clients_frame.tree = ttk.Treeview(clients_frame, height="33", selectmode='browse',
                                  columns=(
                                      'Prefix', 'First Name', 'Surname', 'DOB', 'Telephone',
                                      'Address',
                                      'Postcode'))  # creates the treeview
ClientsFrame.tree = clients_frame.tree
clients_frame.tree.heading('#0', text='ClientID')
clients_frame.tree.heading('#1', text='Prefix')
clients_frame.tree.heading('#2', text='First Name')
clients_frame.tree.heading('#3', text='Surname')
clients_frame.tree.heading('#4', text='DOB')
clients_frame.tree.heading('#5', text='Telephone')
clients_frame.tree.heading('#6', text='Address')
clients_frame.tree.heading('#7', text='Postcode')
clients_frame.tree.column('#0', stretch=Tkinter.YES, width="75", minwidth="50
clients_frame.tree.column('#1', stretch=Tkinter.YES, width="100", minwidth="5
clients_frame.tree.column('#2', stretch=Tkinter.YES, width="75", minwidth="10
clients_frame.tree.column('#3', stretch=Tkinter.YES, width="110", minwidth="85")
clients_frame.tree.column('#4', stretch=Tkinter.YES, width="110", minwidth="85")
clients_frame.tree.column('#5', stretch=Tkinter.YES, width="75", minwidth="75")
clients_frame.tree.column('#6', stretch=Tkinter.YES, width="100"
clients_frame.tree.column('#7', stretch=Tkinter.YES, width="155"
clients_frame.tree.grid(row=5, columnspan=50, rowspan=50, sticky
clients_frame.treeview = clients_frame.tree

search = tk.Label(clients_frame, text="Search: ")
search.grid(row=10, column=50)
search_box = clients_frame.search_entry = tk.Entry(clients_frame
search_box.grid(row=10, column=51)
dropdownsearch = OptionMenu(clients_frame, variable, *OPTIONS)
dropdownsearch.grid(row=10, column=52)

tk.Button(clients_frame, text="Search", command=self.search_table).grid(row=11, column=51)

self.db = sqlite3.connect('clinic.db')
self.cursor = self.db.cursor()

self.cursor.execute("""SELECT Position FROM staff WHERE LoggedIn = ?""", (True,))
pos = self.cursor.fetchone()[0]
if pos == 'Owner               ' or pos == 'Nurse               ' or pos == 'Physiotherapist' or pos == 'Receptionist     ':
    tk.Button(clients_frame, text="Add a client", command=CreateClientFrame).grid(row=13, column=51)


    tk.Button(clients_frame, text="Delete selected client", command=self.delete_client).grid(row=15, column=51)

pad = tk.Label(clients frame, text="")
```

This will show the options available in the drop-down menu, some contain extra spaces, and this is because when some have different lengths it can change the frame size of the program. And spaces balance them out. Below it shows variable.set(OPTIONS[0]) and what this does is make ClientID show as the default option.

Creates all the columns in the program and their width along with all their names. This is done with treeview as I believe that it is the best way to display data in Python, this is due to its intuitive look, and its ease of use.

As you can see the ‘add a client’ and ‘delete selected client’ buttons show for non-IT staff members. This means that when IT logs in they’ll be able to view clients, but not manipulate it. You can see proof of this at the bottom of this section, in addition to other views.

Candidate Number:
3097

```
class CreateClientFrame(tk.Frame):
    """
    creates a new window and allows the user to create a new client,
    when completed it will update the treeview in ClientTab
    """

    def __init__(self):
        tk.Frame.__init__(self)
        self.tree = ClientsFrame.tree
        create_client_window = tk.Toplevel(self)
        create_client_window.geometry("280x230")

        PREFIXOPTIONS = [
            "Dr",
            "Mr",
            "Mrs",
            "Ms",
            "Mx",
            "Prof",
            "Rev"
        ]

        variable = StringVar(create_client_window)
        variable.set(PREFIXOPTIONS[0])

        prefix = tk.Label(create_client_window, text="Prefix: ")
        prefix.grid(row=0, column=0)
        dropdownsearch = OptionMenu(create_client_window, variable, *PREFIXOPTIONS)
        dropdownsearch.grid(row=0, column=1)

        first_name = tk.Label(create_client_window, text="First Name: ")
        first_name.grid(row=1, column=0)
        first_name_box = create_client_window.search_entry = tk.Entry(create_client_window)
        first_name_box.grid(row=1, column=1)

        surname = tk.Label(create_client_window, text="Surname: ")
        surname.grid(row=2, column=0)
        surname_box = create_client_window.search_entry = tk.Entry(create_client_window)
        surname_box.grid(row=2, column=1)

        dob = tk.Label(create_client_window, text="DOB (YYYY-MM-DD): ")
        dob.grid(row=3, column=0)
        dob_box = create_client_window.search_entry = tk.Entry(create_client_window)
        dob_box.grid(row=3, column=1)

        telephone = tk.Label(create_client_window, text="Telephone: ")
        telephone.grid(row=4, column=0)
        telephone_box = create_client_window.search_entry = tk.Entry(create_client_window)
        telephone_box.grid(row=4, column=1)

        address = tk.Label(create_client_window, text="Address: ")
        address.grid(row=5, column=0)
        address_box = create_client_window.search_entry = tk.Entry(create_client_window)
        address_box.grid(row=5, column=1)
```

Creates the entire frame as well as search boxes. There is also an add client button at the end, and a postcode, but this couldn't fit within the screenshot. You can see the rest of this in the next image.

Candidate Number:
3097

```
    postcode = tk.Label(create_client_window, text="Postcode: ")
    postcode.grid(row=6, column=0)
    postcode_box = create_client_window.search_entry = tk.Entry(create_client_window)
    postcode_box.grid(row=6, column=1)

    search_button = tk.Button(create_client_window, text="Add client", command=self.add_client)
    search_button.grid(row=7, column=1)

    self.variable = variable
    self.first_name_box = first_name_box
    self.surname_box = surname_box
    self.dob_box = dob_box
    self.telephone_box = telephone_box
    self.address_box = address_box
    self.postcode_box = postcode_box
    self.create_client_window = create_client_window
    self.db = sqlite3.connect('clinic.db')
    self.cursor = self.db.cursor()

def add_client(self):
    """
    checks if add client results are valid and then updates tables
    """
    self.cursor.execute("""INSERT INTO clients(Prefix, FirstName, Surname, DOB, Telephone, Address, Postcode) VALUES (?,?,?,?,?,?,?)""",
                        (self.variable.get(), self.first_name_box.get(), self.surname_box.get(),
                        self.dob_box.get(), self.telephone_box.get(), self.address_box.get(),
                        self.postcode_box.get(),))

    self.db.commit()
    self.create_client_window.destroy()
    ClientsFrame.update_table(self)
```

Inserts the data into a SQLite3 record.

This destroys the current window and calls the update_table

```
def update_table(self):
    """
    updates the treeview and fills it with all
    in the ClientID table
    """
    db = sqlite3.connect('clinic.db')
    cursor = db.cursor()
    cursor.execute("""SELECT * FROM clients""")
    result = cursor.fetchall()
    self.tree.delete(*self.tree.get_children())
    for item in result:
        self.tree.insert('', 'end', text=item[0], values=item[1:])
```

Updates the entire table, it first connects to SQLite and selects all of the records, then it deletes all records on the treeview as we wouldn't want the same records stacking on each other over and over and finally, it inserts all new ones into the treeview, one by one

Candidate Number:
3097

```
def search_table(self):
    """
    searches for a keyword from a selected column and shows in treeview
    """
    drop_down = self.variable.get()
    search = self.search_box.get()
    if drop_down == "ClientID                ":
        self.cursor.execute("""SELECT * FROM clients WHERE ClientID LIKE ?""", ('%' + search + '%',))
        drop_down = "ClientID"
    elif drop_down == "MedicalRecordID":
        self.cursor.execute("""SELECT * FROM clients WHERE MedicalRecordID LIKE ?""", ('%' + search + '%',))
    elif drop_down == "Prefix                  ":
        self.cursor.execute("""SELECT * FROM clients WHERE Prefix LIKE ?""", ('%' + search + '%',))
        drop_down = "Prefix"
    elif drop_down == "First Name            ":
        self.cursor.execute("""SELECT * FROM clients WHERE FirstName LIKE ?""", ('%' + search + '%',))
        drop_down = "First Name"
    elif drop_down == "Surname               ":
        self.cursor.execute("""SELECT * FROM clients WHERE Surname LIKE ?""", ('%' + search + '%',))
        drop_down = "Surname"
    elif drop_down == "DOB                     ":
        self.cursor.execute("""SELECT * FROM clients WHERE DOB LIKE ?""", ('%' + search + '%',))
        drop_down = "DOB"
    elif drop_down == "Telephone           ":
        self.cursor.execute("""SELECT * FROM clients WHERE Telephone LIKE ?""", ('%' + search + '%',))
        drop_down = "Telephone"
    elif drop_down == "Address               ":
        self.cursor.execute("""SELECT * FROM clients WHERE Address LIKE ?""", ('%' + search + '%',))
        drop_down = "Address"
    elif drop_down == "Postcode              ":
        self.cursor.execute("""SELECT * FROM clients WHERE Postcode LIKE ?""", ('%' + search + '%',))
        drop_down = "Postcode"
    self.tree.delete(*self.tree.get_children()) #clears treeview table
    rows = self.cursor.fetchall()
```

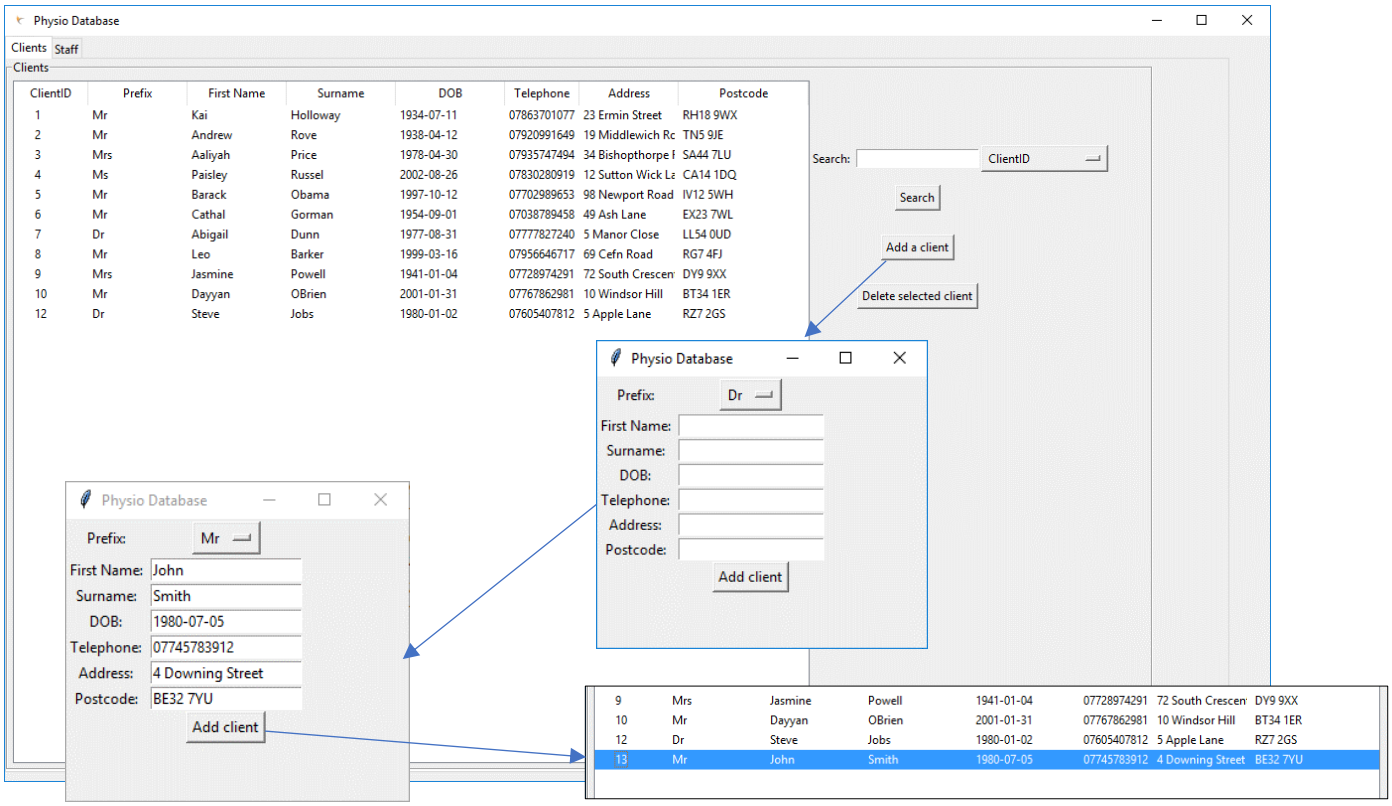
First finds the selected drop-down menu, then it will look for all matches of any column containing that keyword. Once this is done it clears the treeview and updates the table with the returned results. It also outputs all the results of the searched records into a text file

```
def delete_staff(self):
    """
    deletes highlighted client
    """
    iid_selected = self.tree.focus()
    client_id = self.tree.item(iid_selected, 'text')

    db = sqlite3.connect('clinic.db')
    cursor = db.cursor()
    cursor.execute("""DELETE from clients WHERE ClientID = ? """, (client_id,))
    db.commit()
    self.update_table()
```

Finds the highlighted record, and from that the ClientID. Then it opens SQL, deletes the record with that ClientID and updates the table so it disappears.

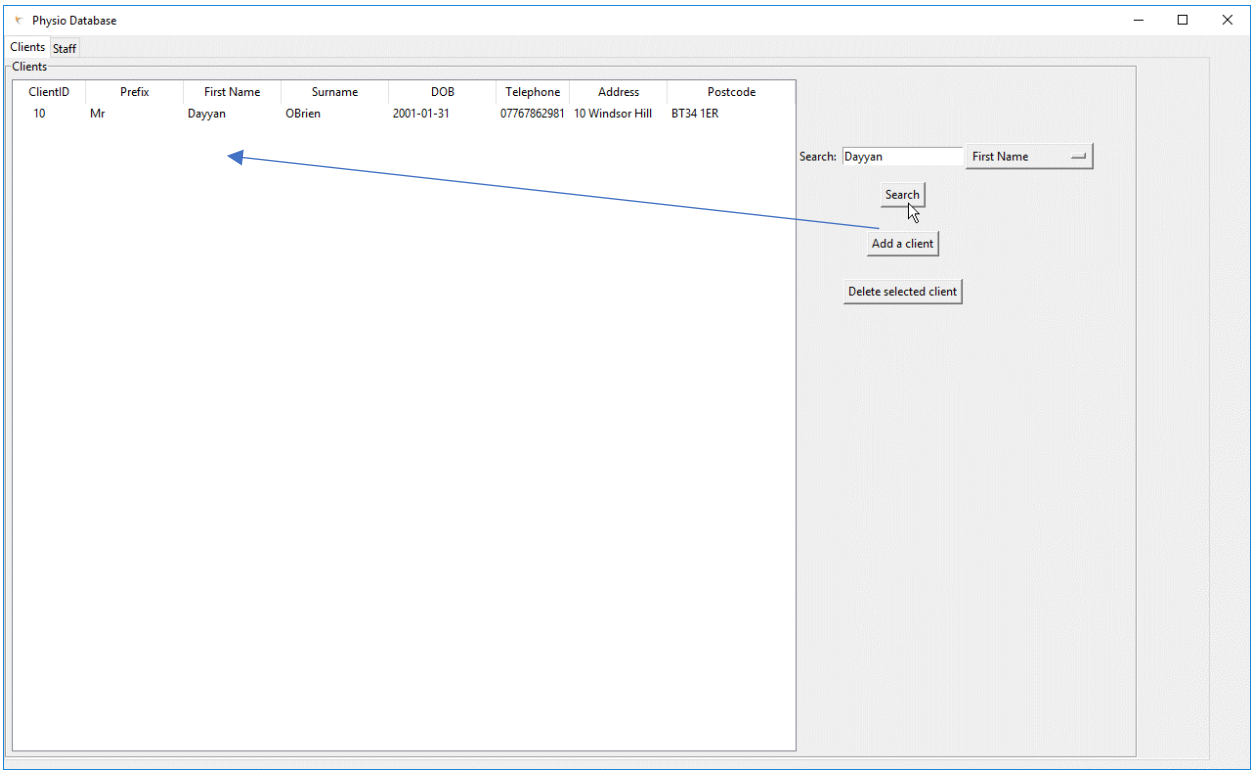
Candidate Number:
3097

## Adding a client:



Candidate Number:
3097

## Searching for a client:

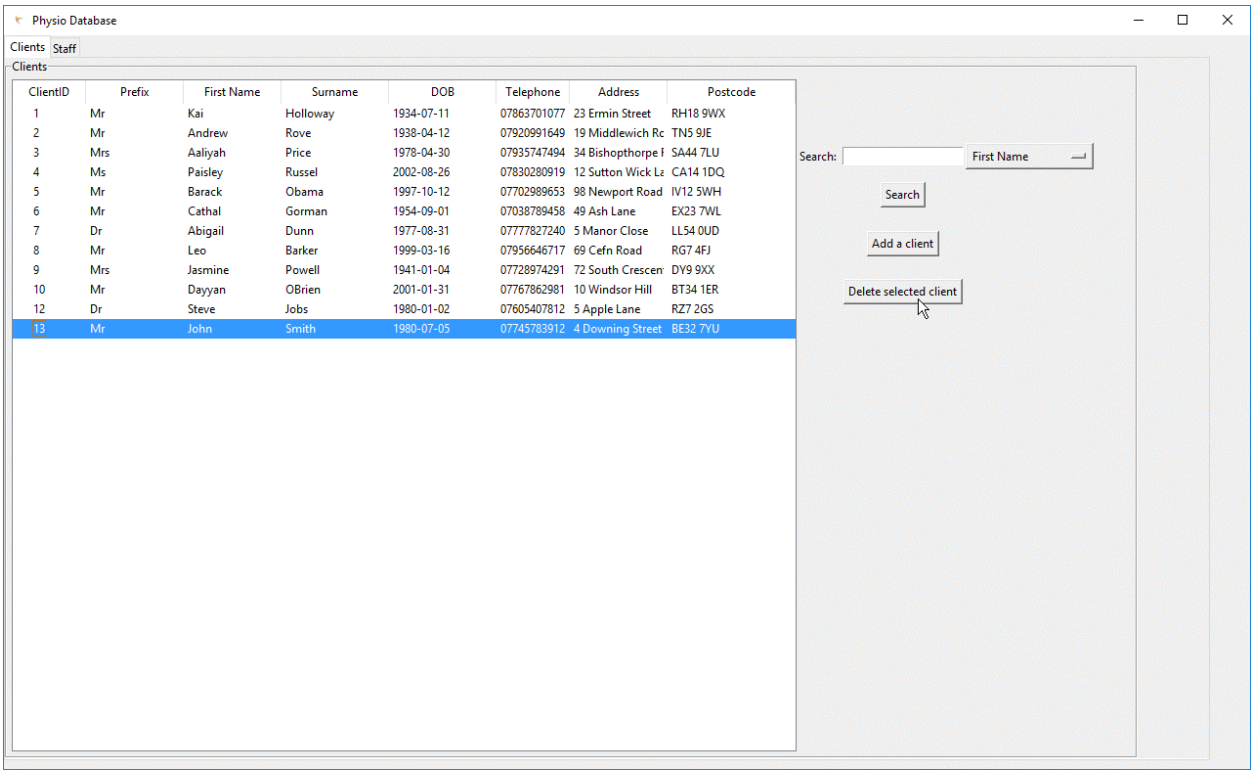



---------- Searched for 'Dayyan' by First Name----------

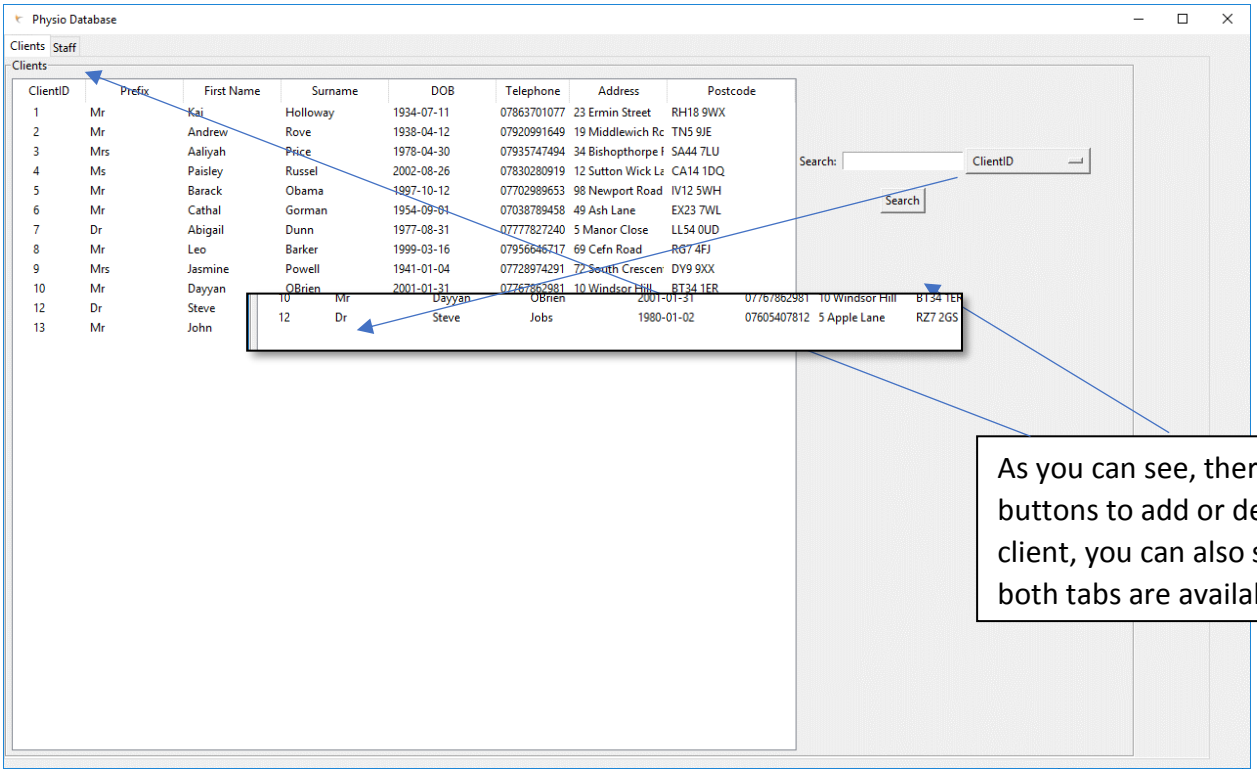
ClientID: 10
MedicalRecordID: Mr
Prefix: Dayyan
First Name: OBrien
Surname: 2001-01-31
DOB: 07767862981
Telephone: 10 Windsor Hill
Address: BT34 1ER
Postcode: 10

Candidate Number:
3097

Deleting a client:

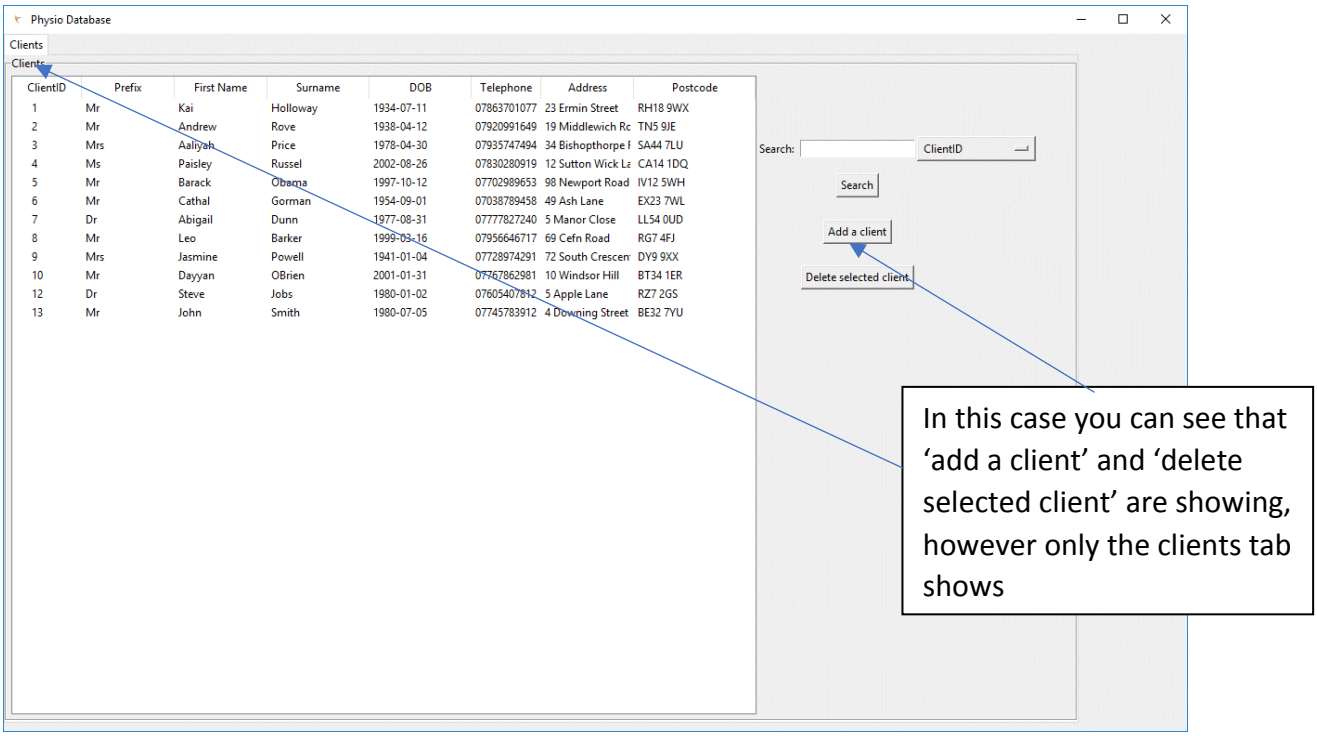



Logging in as IT:



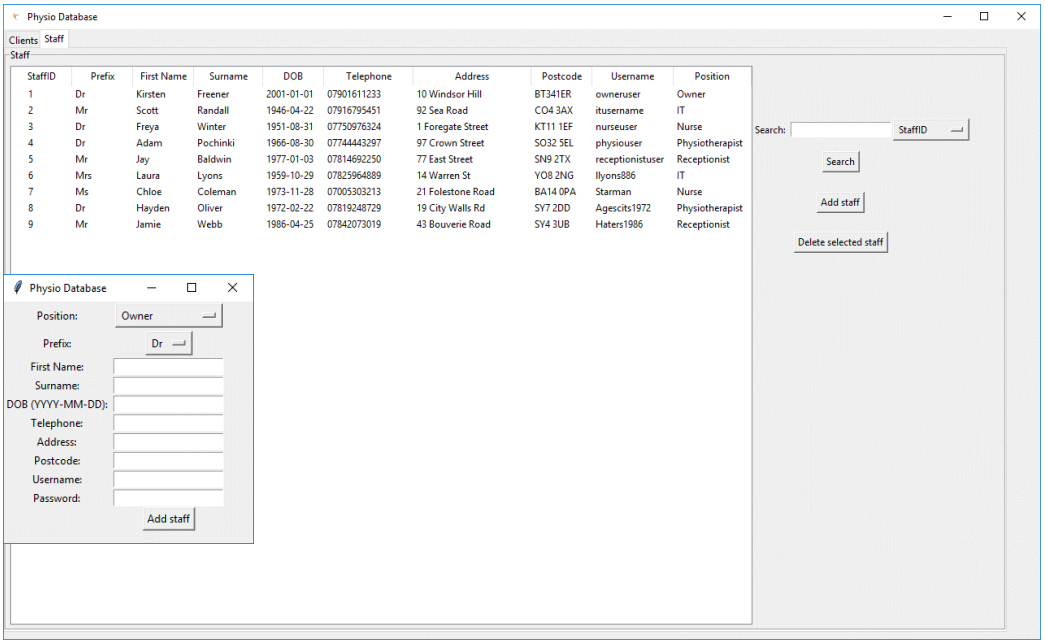
Candidate Number:
3097

Logging in as a nurse, physiotherapist or receptionist:




## 4.8: Staff screen

Below you can see the staff frame, it looks similar to the client's frame which creates consistency in the design of the program. Once again, you are able to view, search, add and delete staff members.



Search: Allows you to input the keyword that you would like to search by. And below that you can click to show all results that contain that keyword, this will be shown on the right-hand side of the screen.

Search drop down: This allows you to choose which column you would like to search by, these are:

StaffID,

Prefix,

First Name,

Surname,

DOB,

Telephone,

Address,

Postcode,

Username,

Position

Add staff: This will open a small window which allows you to enter a staff member and allows you to input data for each of the columns available par StaffID, which is created automatically by the program.

Delete selected staff: This will delete the staff you highlight within the table on the right-hand side

Within add staff there is:

Position: You will select your position in the drop-down menu to the right of ‘Position’

Prefix: You will select your prefix in the drop-down menu to the right of ‘Prefix:’

First name: You will input your first name in the entry box to the right of ‘First name:’

Surname: You will input your surname in the entry box to the right of ‘Surname:’

DOB: You will input your DOB in the entry box to the right of ‘DOB:’

Telephone: You will input your telephone in the entry box to the right of ‘Telephone:’

Address: You will input your address in the entry box to the right of ‘Address :’

Postcode: You will input your postcode in the entry box to the right of ‘Postcode:’

Username: You will input your username in the entry box to the right of ‘Username:’

Password: You will input your password in the entry box to the right of ‘Password:’

Add staff: Once all inputs have been entered it will add the staff.

If all the above are correct, it will then add the staff and display them on the main window.

Candidate Number:
3097

```
OPTIONS = [
    "StaffID      ",
    "Prefix         ",
    "First Name",
    "Surname    ",
    "DOB            ",
    "Telephone",
    "Address      ",
    "Postcode    ",
    "Username  ",
    "Position   "
]

variable = StringVar(staff_frame)
variable.set(OPTIONS[0])
```

This will show the options available in the drop-down menu, some contain extra spaces, and this is because when some have different lengths it can change the frame size of the program. And spaces balance them out. Below it shows variable.set(OPTIONS[0]) and what this does is make StaffID show as the default option.

```
# Set the treeview
staff_frame.tree = ttk.Treeview(staff_frame, height="33", selectmode='browse',
                                columns=(
                                    'Prefix', 'First Name', 'Surname', 'DOB', 'Telephone', 'Address',
                                    'Postcode', 'Username', 'Position'))
StaffFrame.tree = staff_frame.tree
staff_frame.tree.heading('#0', text='StaffID')
staff_frame.tree.heading('#1', text='Prefix')
staff_frame.tree.heading('#2', text='First Name')
staff_frame.tree.heading('#3', text='Surname')
staff_frame.tree.heading('#4', text='DOB')
staff_frame.tree.heading('#5', text='Telephone')
staff_frame.tree.heading('#6', text='Address')
staff_frame.tree.heading('#7', text='Postcode')
staff_frame.tree.heading('#8', text='Username')
staff_frame.tree.heading('#9', text='Position')
staff_frame.tree.column('#0', stretch=Tkinter.YES, width="75", minwidth
staff_frame.tree.column('#1', stretch=Tkinter.YES, width="75", minwidth
staff_frame.tree.column('#2', stretch=Tkinter.YES, width="75", minwidth="50")
staff_frame.tree.column('#3', stretch=Tkinter.YES, width="85", minwidth="50")
staff_frame.tree.column('#4', stretch=Tkinter.YES, width="75", minwidth="75")
staff_frame.tree.column('#5', stretch=Tkinter.YES, width="110", minwidth="100")
staff_frame.tree.column('#6', stretch=Tkinter.YES, width="145", minwidth='100')
staff_frame.tree.column('#7', stretch=Tkinter.YES, width="75", minwidth="75")
staff_frame.tree.column('#8', stretch=Tkinter.YES, width="100", minwidth='100')
staff_frame.tree.column('#9', stretch=Tkinter.YES, width="95", minwidth="50")
staff_frame.tree.grid(row=5, columnspan=50, rowspan=50, sticky='nsew')
staff_frame.treeview = staff_frame.tree

search = tk.Label(staff_frame, text="Search: ")
search.grid(row=10, column=50)
search_box = staff_frame.search_entry = tk.Entry(staff_frame)
search_box.grid(row=10, column=51)
dropdownsearch = OptionMenu(staff_frame, variable, *OPTIONS)
dropdownsearch.grid(row=10, column=52)

tk.Button(staff_frame, text="Search", command=self.search_table).grid(row=11, column=51)

tk.Button(staff_frame, text="Add staff", command=CreateStaffFrame).grid(row=13, column=51)

tk.Button(staff_frame, text="Delete selected staff", command=self.delete_staff).grid(row=15, column=51)

pad = tk.Label(staff_frame, text="")
pad.grid(row=10, column=55, padx=(0, 30))
```

Creates all the columns in the program and their width along with all their names. This is done with treeview as I believe that it is the best way to display data in Python, this is due to its intuitive look, and its ease of use.

Candidate Number:
3097

```
class CreateStaffFrame(tk.Frame):
    """
    creates a new window and allows the user to create a new staff,
    when completed it will update the treeview in staffTab
    """

    def __init__(self):
        tk.Frame.__init__(self)
        self.tree = StaffFrame.tree
        create_staff_window = tk.Toplevel(self)
        create_staff_window.geometry("280x270")

        PREFIXOPTIONS = [
            "Dr",
            "Mr",
            "Mrs",
            "Ms",
            "Mx",
            "Prof",
            "Rev"
        ]

        variable = StringVar(create_staff_window)
        variable.set(PREFIXOPTIONS[0])

        POSITIONOPTIONS = [
            "Owner               ",
            "IT                   ",
            "Nurse               ",
            "Physiotherapist",
            "Receptionist     "
        ]

        positionvariable = StringVar(create_staff_window)
        positionvariable.set(POSITIONOPTIONS[0])

        position = tk.Label(create_staff_window, text="Position: ")
        position.grid(row=0, column=0)
        positiondropdownsearch = OptionMenu(create_staff_window, positionvariable, *POSITIONOPTIONS)
        positiondropdownsearch.grid(row=0, column=1)

        prefix = tk.Label(create_staff_window, text="Prefix: ")
        prefix.grid(row=1, column=0)
        dropdownsearch = OptionMenu(create_staff_window, variable, *PREFIXOPTIONS)
        dropdownsearch.grid(row=1, column=1)

        first_name = tk.Label(create_staff_window, text="First Name: ")
        first_name.grid(row=2, column=0)
        first_name_box = create_staff_window.search_entry = tk.Entry(create_staff_window)
        first_name_box.grid(row=2, column=1)

        surname = tk.Label(create_staff_window, text="Surname: ")
        surname.grid(row=3, column=0)
        surname_box = create_staff_window.search_entry = tk.Entry(create_staff_window)
        surname_box.grid(row=3, column=1)

        dob = tk.Label(create_staff_window, text="DOB (YYYY-MM-DD): ")
        dob.grid(row=4, column=0)
        dob_box = create_staff_window.search_entry = tk.Entry(create_staff_window)
        dob_box.grid(row=4, column=1)
```

Creates the entire frame as well as search boxes and drop-down menus.

Candidate Number:
3097

```
    telephone = tk.Label(create_staff_window, text="Telephone: ")
    telephone.grid(row=5, column=0)
    telephone_box = create_staff_window.search_entry = tk.Entry(create_staff_window)
    telephone_box.grid(row=5, column=1)

    address = tk.Label(create_staff_window, text="Address: ")
    address.grid(row=6, column=0)
    address_box = create_staff_window.search_entry = tk.Entry(create_staff_window)
    address_box.grid(row=6, column=1)

    postcode = tk.Label(create_staff_window, text="Postcode: ")
    postcode.grid(row=7, column=0)
    postcode_box = create_staff_window.search_entry = tk.Entry(create_staff_window)
    postcode_box.grid(row=7, column=1)

    username = tk.Label(create_staff_window, text="Username: ")
    username.grid(row=8, column=0)
    username_box = create_staff_window.search_entry = tk.Entry(create_staff_window)
    username_box.grid(row=8, column=1)

    password = tk.Label(create_staff_window, text="Password: ")
    password.grid(row=9, column=0)
    password_box = create_staff_window.search_entry = tk.Entry(create_staff_window)
    password_box.grid(row=9, column=1)

    search_button = tk.Button(create_staff_window, text="Add staff", command=self.add_staff)
    search_button.grid(row=10, column=1)

    self.variable = variable
    self.first_name_box = first_name_box
    self.surname_box = surname_box
    self.dob_box = dob_box
    self.telephone_box = telephone_box
    self.address_box = address_box
    self.postcode_box = postcode_box
    self.positionvariable = positionvariable
    self.username_box = username_box
    self.password_box = password_box
    self.create_staff_window = create_staff_window
    self.db = sqlite3.connect('clinic.db')
    self.cursor = self.db.cursor()

def add_staff(self):
    """
    checks if add staff results are valid and then updates tables
    """
    self.cursor.execute("""INSERT INTO staff(Prefix, FirstName, Surname, DOB, Telephone, Address,
                        Postcode, Username, Position, LoggedIn, Password) VALUES (?,?,?,?,?,?,?,?,?,?,?)""",
                        (self.variable.get(), self.first_name_box.g
                        self.surname_box.get(),
                        self.dob_box.get(), self.telephone_box.get(
                        self.address_box.get(),
                        self.postcode_box.get(), self.username_box.
                        self.positionvariable.get(), False,
                        self.password_box.get()))
    self.db.commit()
    self.create_staff_window.destroy()
    StaffFrame.update_table(self)
```

Inserts the data into a SQLite3 record.

Deletes the add a staff frame and updates the treeview.

Candidate Number:
3097

```
def update_table(self):
    """
    updates the treeview and fills it with a
    in the staffid table
    """
    self.cursor.execute("""SELECT * FROM sta
    result = self.cursor.fetchall()
    self.tree.delete(*self.tree.get_children())
    for item in result:
        self.tree.insert('', 'end', text=item[0], values=item[1:10])
```

Updates the entire table, it first connects to SQLite and selects all of the records, then it deletes all records on the treeview as we wouldn't want the same records stacking on each other over and over and finally, it inserts all new ones into the treeview, one by one

```
def delete_staff(self):
    """
    deletes highlighted staff
    """
    iid_selected = self.tree.focus()
    staff_id = self.tree.item(iid_selected, 'text')

    self.cursor.execute("""DELETE from staff WHERE StaffID = ? """, (staff_id,))
    self.db.commit()
def     self.update_table()
```

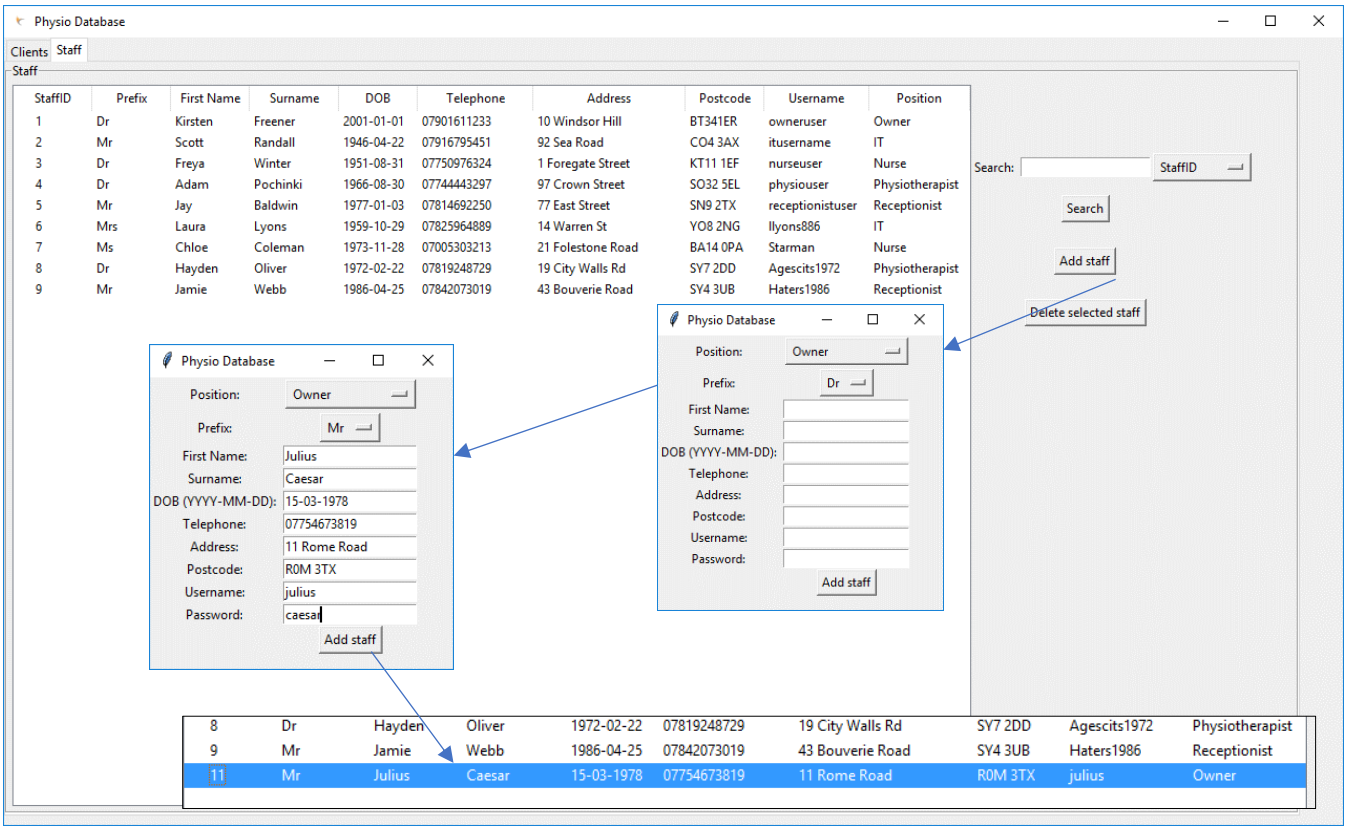
Finds the highlighted record, and from that the StaffID. Then it opens SQL, deletes the record with that StaffID and updates the table so it disappears.

```
    """
    drop_down = self.variable.get()
    search = self.search_box.get()
    if drop_down == "StaffID      ":
        self.cursor.execute("""SELECT * FROM staff WHERE StaffID LIKE ?""", ('%' + search + '%',))
        drop_down = "StaffID"
    elif drop_down == "Prefix         ":
        self.cursor.execute("""SELECT * FROM staff WHERE Prefix LIKE ?""", ('%' + search + '%',))
        drop_down = "Prefix"
    elif drop_down == "First Name":
        self.cursor.execute("""SELECT * FROM staff WHERE FirstName LIKE ?""", ('%' + search + '%',))
    elif drop_down == "Surname    ":
        self.cursor.execute("""SELECT * FROM staff WHERE Surname LIKE ?""", ('%' + search + '%',))
        drop_down = "Surname"
    elif drop_down == "DOB            ":
        self.cursor.execute("""SELECT * FROM staff WHERE DOB LIKE ?""", ('%' + search + '%',))
        drop_down = "DOB"
    elif drop_down == "Telephone":
        self.cursor.execute("""SELECT * FROM staff WHERE Telephone LIKE ?""", ('%' + search + '%',))
    elif drop_down == "Address      ":
        self.cursor.execute("""SELECT * FROM staff WHERE Address LIKE ?""", ('%' + search + '%',))
        drop_down = "Address"
    elif drop_down == "Postcode   ":
        self.cursor.execute("""SELECT * FROM staff WHERE Postcode LIKE ?""", ('%' + search + '%',))
        drop_down = "Postcode"
    elif drop_down == "Username  ":
        self.cursor.execute("""SELECT * FROM staff WHERE Username LIKE ?""", ('%' + search + '%',))
        drop_down = "Username"
    elif drop_down == "Position   ":
        self.cursor.execute("""SELECT * FROM staff WHERE Position LIKE ?""", ('%' + search + '%',))
        drop_down = "Position"
    self.tree.delete(*self.tree.get_children())
    rows = self.cursor.fetchall()
    f = open('staff.txt','w')
    f.write("------------------------------------ Searched for '" + search + "' by " + drop_down + "------------------------------------" + '\n')
    for row in rows:
        self.tree.insert('', 'end', text=row[0], values=row[1:])
        f.write('\n' + "StaffID: " + str(row[0]))
        f.write('\n' + "Prefix: " + str(row[1]))
        f.write('\n' + "First Name: " + str(row[2]))
        f.write('\n' + "Surname: " + str(row[3]))
        f.write('\n' + "DOB: " + str(row[4]))
        f.write('\n' + "Telephone: " + str(row[5]))
        f.write('\n' + "Address: " + str(row[6]))
        f.write('\n' + "Postcode: " + str(row[7]))
        f.write('\n' + "Username: " + str(row[8]))
        f.write('\n' + "Position: " + str(row[9]))
        f.write('\n')
    messagebox.showinfo("Alert", "Staff saved (staff.txt)")
    f.close()
```

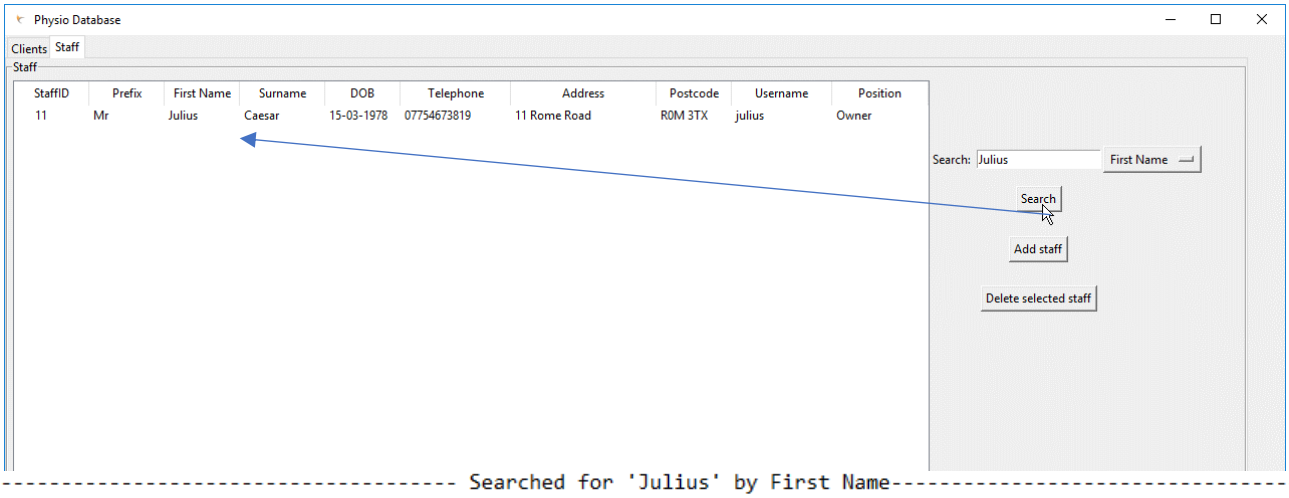
First finds the selected drop-down menu, then it will look for all matches of any column containing that keyword. Once this is done it clears the treeview and updates the table with the returned results. Finally, it outputs to a text file

Candidate Number:
3097

## Add a staff member:




## Searching for a staff member:

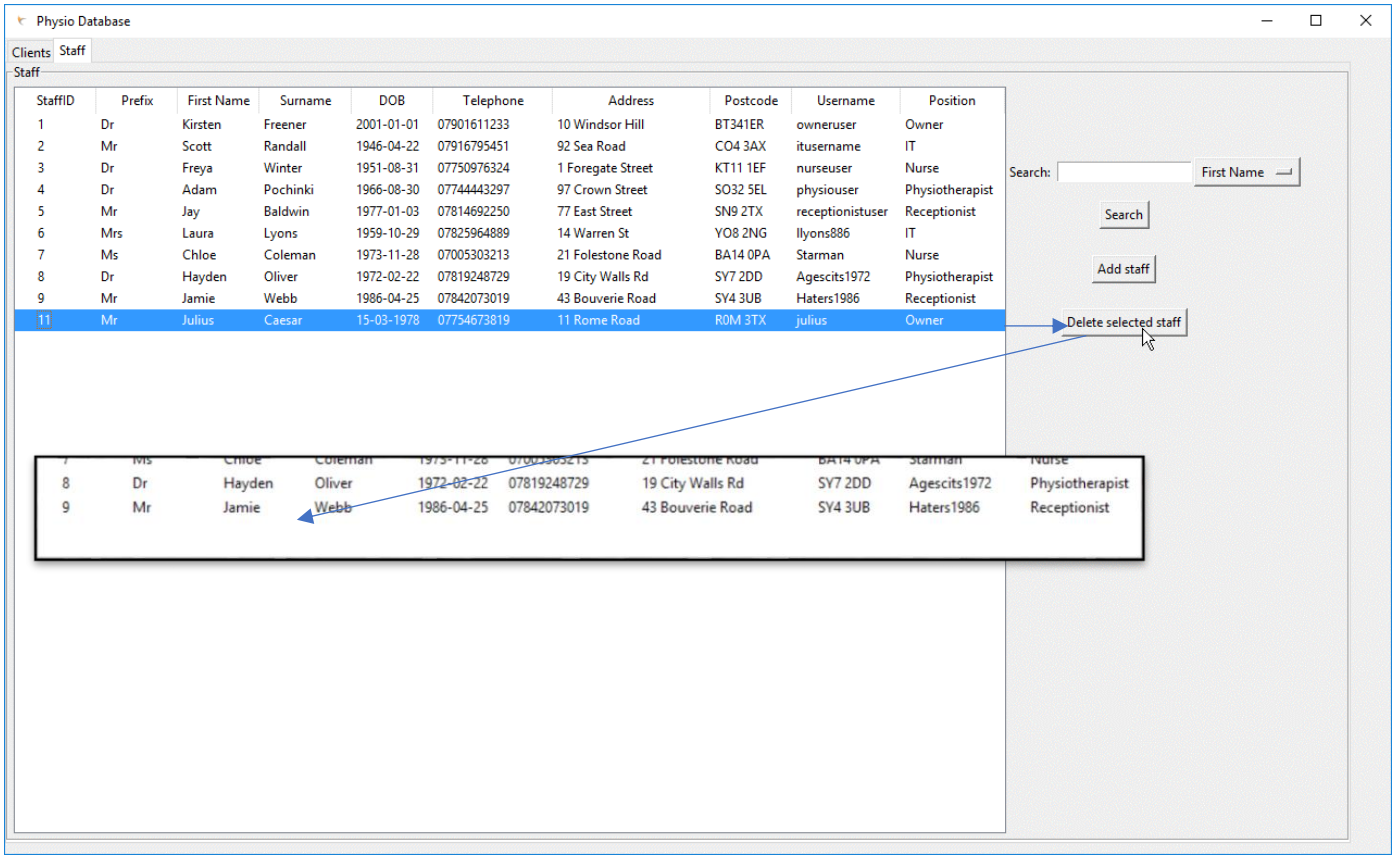



---------- Searched for 'Julius' by First Name----------

StaffID: 11
Prefix: Mr
First Name: Julius
Surname: Caesar
DOB: 15-03-1978
Telephone: 07754673819
Address: 11 Rome Road
Postcode: R0M 3TX
Username: julius
Position: Owner

Candidate Number:
3097

## Deleting a staff member:



# 4.9: Evaluation

## What are the positives of the prototype?

I am very proud of the prototype that I have created, and I am very happy with all the features that I have implemented into it. I have created the base of my software development and my prototype will allow me to gauge what the staff members think about the program I have created. I have implemented a variety of features. I have created a program which contains functional views with a login system which results in a high level of data security. I have created two tables that are separated by tabs and I have the entire program within the same window. This shows how intuitive and easy the navigation around the system is which was one of my main focus when designing the system. I have also created features that output a search based on a keyword in a column specified by the user in the treeview and in a text file for both the clients and staff tabs. I have implemented features that allow the user to add both staff and clients to the program and have made it so that these records are shown automatically within the treeview. I also have a system that allows the user to easily delete a record simply by highlighting the record that they intend to delete and then pressing 'delete selected record', I have implemented this function on both the clients and the staff table. Overall, I believe my prototype is highly effective and I am very pleased with what I have managed to create and all the features I have managed to implement in addition to creating a rough layout of what the future program will look and a strong base to build upon for anything added in the future.

## What are the negatives about the prototype?

The prototype does have a number of issues, the biggest of these issues is its lack of validation, this means you can insert random data into the program that shouldn't be typically accept. No validation can result in a lot of issues in my program. Invalid data entered can result in the breaking of the program, for example if you entered non-numerical values for your difference, the program would have an error if you tried to calculate the sum of a transaction. Another negative with my program is that it contains no ability to edit a record. This means that if I want to change certain features of a record I have to delete that record and then recreate it fully. This is time consuming and can take a while. It might also mean you have to recreate all records related to that value. This is a significant problem and I hope to fix this in the future. Another negative that I feel about my program is the format of the login page. I feel that it is not aesthetically pleasing as it is stuck in the top right corner. Finally, the program only contains two tabs instead of tabs for all of the possible tables, these new tables, being one for medical records, finances and appointments. The current program only allows you to hold data for user details, but it does not allow you to do anything, this means that the prototype does not allow you to book an appointment, create a finance recipt of find the BMI of a client.

### What can be done to improve the prototype?

I plan to take a number of steps to improve the prototype. I am going to look at the negatives and issues that I have with my current prototype in addition to any feedback that I will receive in the post-prototype section and take that information to improve the prototype to make a better software development in the end. The main areas that I plan on focusing to improve the prototype are on validation, the GUI and finally the addition of new tables. I plan on implementing a large array of validation checks throughout from range checks, type checks, length checks and format checks in addition to a number of other validation checks and checks for any exceptions. I will improve the GUI by centring the login pages GUI as this should look significantly more aesthetically pleasing in addition to matching the proposed screens I created in the design section of the program. New tables will be added for the entire program and these will all be held in third normalised form. In addition to the above, I will take on feedback from my post-prototype questionnaires and other methods of investigation to further improve the overall project.

## 1.11: Restructuring of the final program in light of the prototype

Due to the prototype, I am more confident about my final program and believe that little to not restructuring is necessary. I do feel that I will still be able to meet the success criteria that I initially created, and I will be able to improve the prototype based on the points that I have made. In the next section, the post-prototype I plan to see what other users think about my program and I plan to take this information to better evaluate my success criteria in the future. Overall, I believe little change is necessary for my program and that I will be able to meet all, or nearly all of the requirements in my broad aims, success criteria and broad aims.