Candidate Number:
3097

Design:

3.1: Design

I've already created the investigation and discussion of the program and created suitable specifications for that. I will create a design of the overall program to split what I plan to do into smaller parts and see the data that I will create.

This has a lot of advantages, for example it will allow the program to be created in bite-sized chunks making it significantly easier to model. I will create the tables and foreign keys and see the flow of data throughout the program, I will use visual aid for this to make it easier to understand. I will also create pseudocode for a large part of the program to reduce the amount of logical problems I will have when developing my prototype and final product. I will create a relational database in third normal form as this would make the program significantly faster and easier to use it will also decrease any data redundancy that I may have throughout the program.

I will link the table with foreign keys and each table will contain a primary one, in addition to the tables I will create a wide array of validation to the program to insure invalid data cannot be entered into it.

Overall, the design is a significant part of the project and will create the foundation of the program and will allow any future developers to understand the logic of the program.

3.2: Modules

Login:

The login page will be a large GUI with its own frame. It will be simple with a box to input your username and password in the center of the screen in addition to a login button. It will:

- check if login details are valid
- if they are it will use views to only show the user appropriate information about the program
- change frame and create the tabs of all tables the views allow

Candidate Number:
3097

Clients:

The clients page will be the initial table the user sees when logging into the program, it will be the first of the 5 tabs. It will be a GUI and the table will be created in a treeview with this table to the left side of the screen. To the right of it will be buttons to search, create, edit and delete clients. It will:

- allow the user to view all clients
- allow the user to search for clients by any of the columns
- allow to user to output the search to a text file
- allow the user to create a client
- will create a foreign key for medical records when a client is created
- allow the user to edit a client's details
- allow the user to delete a selected client
- contain validation for any inputs

Staff:

The staff page will be one of the tabs after the login page switches. It will be a GUI and the table will be created in a treeview with this table to the left side of the screen. To the right of it will be buttons to search, create, edit and delete staff. It will:

- allow the user to view all staff
- allow the user to search for staff by any of the columns
- allow to user to output the search to a text file
- allow the user to create a staff
- allow the user to edit a staff detail
- allow the user to delete a selected staff
- contain validation for any inputs

Appointments:

The appointment page will be one of the tabs after the login page switches. It will be a GUI and the table will be created in a treeview with this table to the left side of the screen. To the right of it will be buttons to search, create and cancel appointments. It will:

- allow the user to view all appointments
- allow the user to search for appointments by any of the columns
- allow to user to output the search to a text file
- allow the user to create an appointment
- will create a foreign key for transactions when an appointment is created
- allow the user to cancel a selected appointment
- contain validation for any inputs

Candidate Number:
3097

Medical Records:

The medical records page will be one of the tabs after the login page switches. It will be a GUI and the table will be created in a treeview with this table to the left side of the screen. To the right of it will be buttons to search, edit and delete medical records. It will:

- allow the user to view all medical record
- allow the user to search for medical records by any of the columns
- allow to user to output the search to a text file
- allow the user to edit a medical record
- allow the user to delete medical record
- contain validation for any inputs

Transactions:

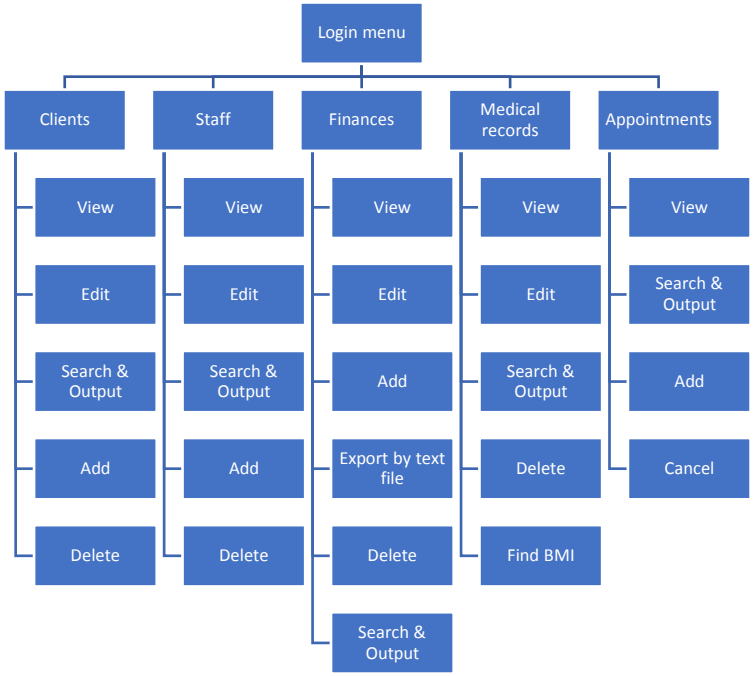
The transactions page will be one of the tabs after the login page switches. It will be a GUI and the table will be created in a treeview with this table to the left side of the screen. To the right of it will be buttons to search, create, edit, delete and save a transaction from a specified date. It will:

- allow the user to view all transactions
- allow the user to search for transactions by any of the columns
- allow to user to output the search to a text file
- allow the user to create a transaction
- allow the user to edit a transactions details
- allow the user to delete a selected transaction
- allow the user to save a transaction from a certain date
- calculate the overall difference from that date
- contain validation for any inputs

Candidate Number:
3097

3.3: Discussion of design

Menu diagram



3.4: Required files and data structures

Data dictionaries

Clients:

Field name	Default value	Data type	Field length	Description	Validation
ClientID	0	INTEGER	-	Primary key, unique identifier for the table	Automatically created, none required
MedicalRecordID	0	INTEGER	-	Foreign key, links to the MedicalRecordID table	Automatically created, none required
Prefix	{Dr/Mr/Mrs/Ms/ Mx/Prof/Rev}	TEXT	4	The client's prefix	From dropdown menu, so none required
FirstName	{first name}	TEXT	19	The client's first name	Presence, length (more than 2, less than 20), alphabetic characters
Surname	{surname}	TEXT	19	The client's surname	Presence, length (more than 2, less than 20), alphabetic characters
DOB	{YYYY-MM-DD}	DATE	10	The client's DOB	Presence, format
Telephone	00000000000	TEXT	11	The client's telephone number	Digit, length (exactly 11)
Address	{address}	TEXT	49	The client's address	Length (more than, 5 less than 50)
Postcode	{AA0A 0AA/ A0A 0AA/ A0 0AA/ A00 0AA/ AA0 0AA/ AA00 0AA}	TEXT	8	The client's postcode	Presence, format

Staff:

Field name	Default value	Data type	Field length	Description	Validation
StaffID	0	INTEGER	-	Primary key, unique identifier for the table	Automatically created, none required
Prefix	{Dr/Mr/Mrs/Ms/ Mx/Prof/Rev}	TEXT	4	The staff's prefix	From dropdown menu, so none required
FirstName	{first name}	TEXT	19	The staff's first name	Presence, length(more than 2, less than 20), alphabetic characters
Surname	{surname}	TEXT	19	The staff's surname	Presence, length (more than 2, less than 20), alphabetic characters
DOB	{YYYY-MM-DD}	DATE	10	The staff's DOB	Presence, format
Telephone	00000000000	TEXT	11	The staff's telephone number	Digit, length (exactly 11)
Address	{address}	TEXT	49	The staff's address	Length (more than, 5 less than 50)
Postcode	{AA0A 0AA/ A0A 0AA/ A0 0AA/ A00 0AA/ AA0 0AA/ AA00 0AA}	TEXT	8	The staff's postcode	Presence, format
Position	{Owner/IT/Nurse/ Physiotherapist/ Receptionist}	TEXT	15	The staff's position	From dropdown menu, so none required
Username	{username}	TEXT		The staff's username	Presence, length (more than 6)
Password	{password}	TEXT		The staff's password	Presence, length (more than 6)
LoggedIn	{True/False}	BOOLEAN		The staff's login status	Automatically created, none required

Medical records:

Field name	Default value	Data type	Field length	Description	Validation
MedicalRecordID	0	INTEGER	-	Primary key, unique identifier for the table	Automatically created, none required
ClientID	0	INTEGER	-	Foreign key, links to the ClientID table	Automatically created, none required
Sex	{M/F}	TEXT	1	The client's sex	From dropdown menu, so none required
Gender	{Male/Female/ Other}	TEXT	6	The client's gender	From dropdown menu, so none required
BloodType	{A+/A-/B+/B-/ O+/O-/AB+/AB-}	TEXT	3	The client's blood type	From dropdown menu, so none required
Height	0.00	FLOAT	4	The client's height	Digit, value (more than 1 less than 2.5), presence
Mass	000.00	FLOAT	6	The client's mass	Digit, value (more than 30, less than 500), presence

Finances:

Field name	Default value	Data type	Field length	Description	Validation
TransactionID	0	INTEGER	-	Primary key, unique identifier for the table	Automatically created, none required
Difference	{+/- 000.00}	INTEGER	-	The difference of the transaction	Format, presence
DateAndTime	{YYYY-MM-DD HH:MM:SS}	TEXT	19	The date and time of the transaction	Presence, format
TransactionStatus	{Success/Fail}	TEXT	7	The status of the transaction	Automatically created, none required

Appointments:

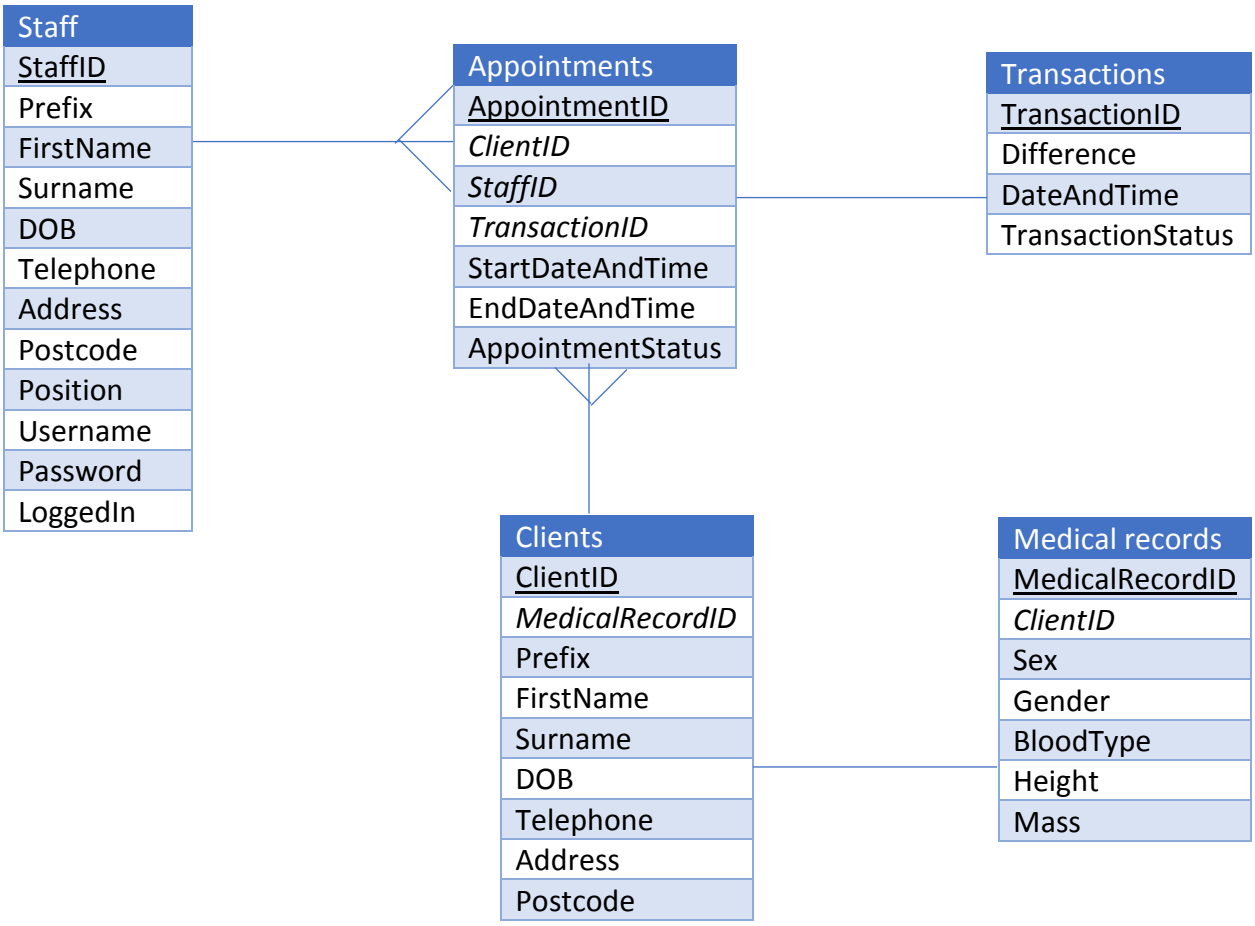
Field name	Default value	Data type	Field length	Description	Validation
AppointmentID	0	INTEGER	-	Primary key, unique identifier for the table	Automatically created, none required
ClientID	0	INTEGER	-	Foreign key, links to the ClientID table	Automatically created, none required
StaffID	0	TEXT	1	Foreign key, links to the StaffID table	Automatically created, none required
TransactionID	0	TEXT	6	Foreign key, links to the TransactionID table	Automatically created, none required
StartDateAndTime	{YYYY-MM-DD HH:MM:SS}	TEXT	19	The start date and time of the appointment	Format, presence
EndDateAndTime	{YYYY-MM-DD HH:MM:SS}	TEXT	19	The end date and time of the appointment	Format, presence
AppointmentStatus	{Active/ Cancelled}	FLOAT	6	The appointment status	Automatically created, none required

Methods of access

The data will be stored in a SQL table called ‘clinic.db’, this table will be used to store all the data of the program and SQLite3 commands will be used throughout the program and will be connected to the database to view, create, edit and delete records. Each field will have an appropriate data type and the staff table will be used for views.

Candidate Number:
3097

Entity relationship diagram



Candidate Number:
3097

Normalisation

UNF

StaffPrefix, StaffFirstName, StaffSurname, StaffDOB, StaffTelephone, StaffAddress, Position, Username, Password, LoggedIn, StartDateAndTime, EndDateAndTime, AppointmentStatus, Difference, DateAndTime, TransactionStatus, ClientPrefix, ClientSurname, ClientDOB, ClientTelephone, ClientAddress, ClientPostcode, Sex, Gender, Height, Mass

1NF

CLIENTS(ClientID, *MedicalRecordID*, Prefix, FirstName, Surname, DOB, Telephone, Address)

STAFF(StaffID, Prefix, FirstName, Surname, DOB, Telephone, Address, Position, Username, Password, LoggedIn)

APPOINTMENTS(AppointmentID, *StaffID* , *MedicalRecordID*, StartDateAndTime, EndDateAndTime, AppointmentStatus, Difference, DateAndTime, TransactionStatus)

MEDICALRECORDS(MedicalRecordID, *ClientID*, Sex, Gender, Height, Mass)

2/3NF

CLIENTS(ClientID, *MedicalRecordID*, Prefix, FirstName, Surname, DOB, Telephone, Address)

STAFF(StaffID, Prefix, FirstName, Surname, DOB, Telephone, Address, Position, Username, Password, LoggedIn)

TRANSACTIONS(TransactionID, Difference, DateAndTime, TransactionStatus)

APPOINTMENTS(AppointmentID, *StaffID*, *TransactionID*, *MedicalRecordID*, StartDateAndTime, EndDateAndTime, AppointmentStatus)

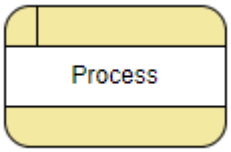
MEDICALRECORDS(MedicalRecordID, *ClientID*, Sex, Gender, Height, Mass)

Candidate Number:
3097

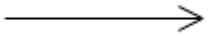
3.5: Data flow diagrams

External entity

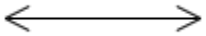
Data store



Process

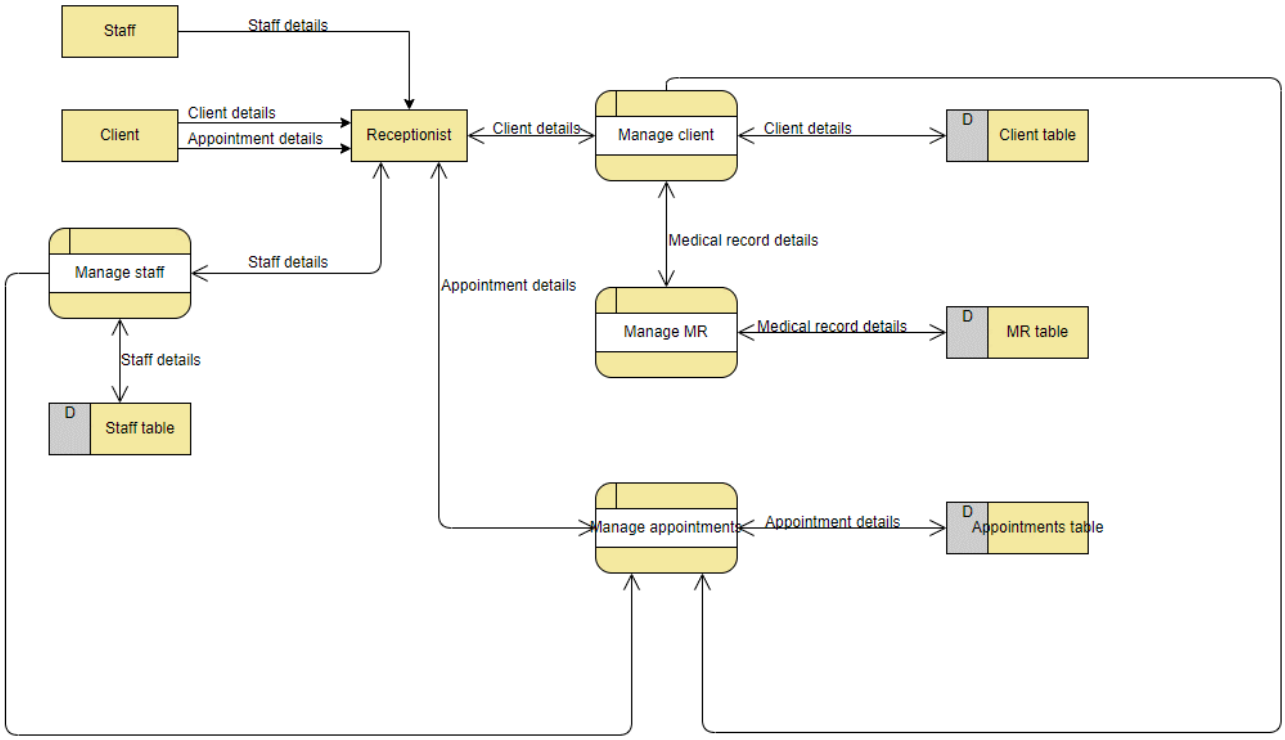


Data flow



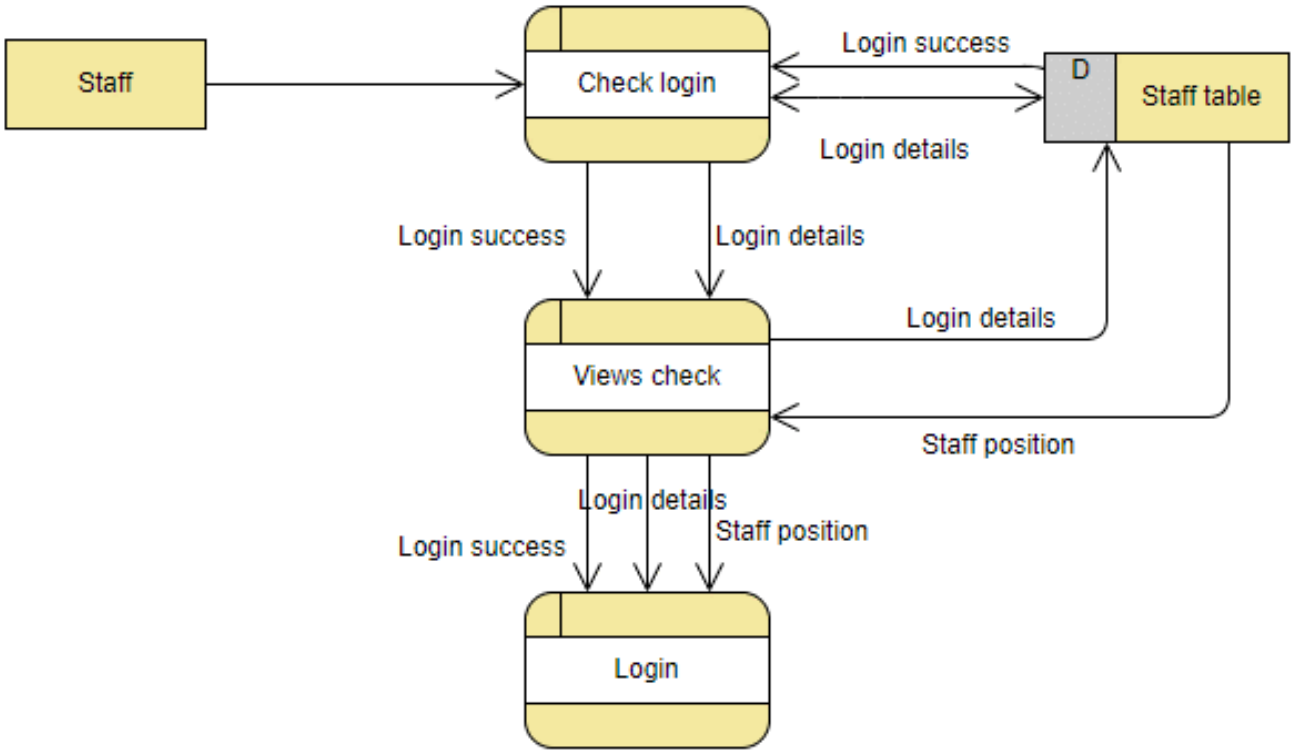
Bidirectional data flow

Level 0 DFD for the whole program:



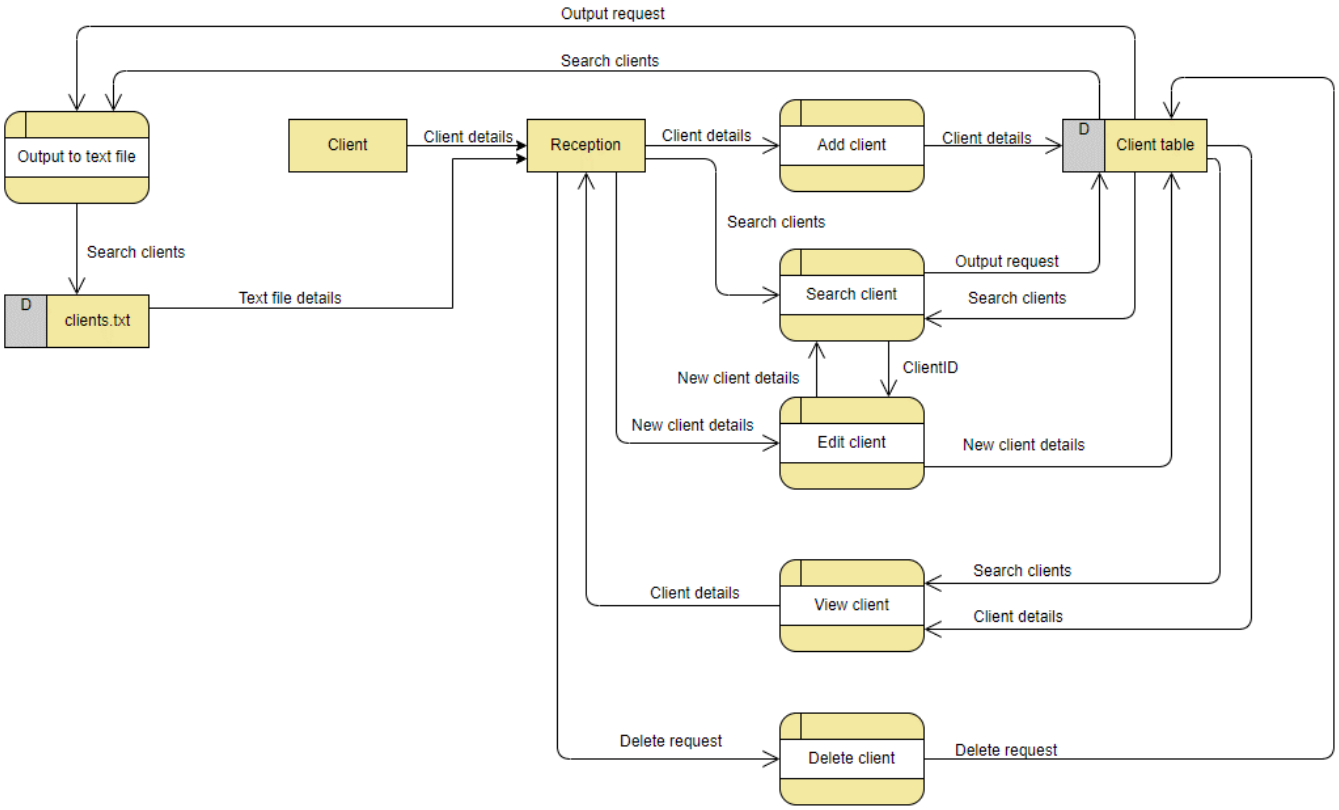
Candidate Number:
3097

Level 1 Login DFD



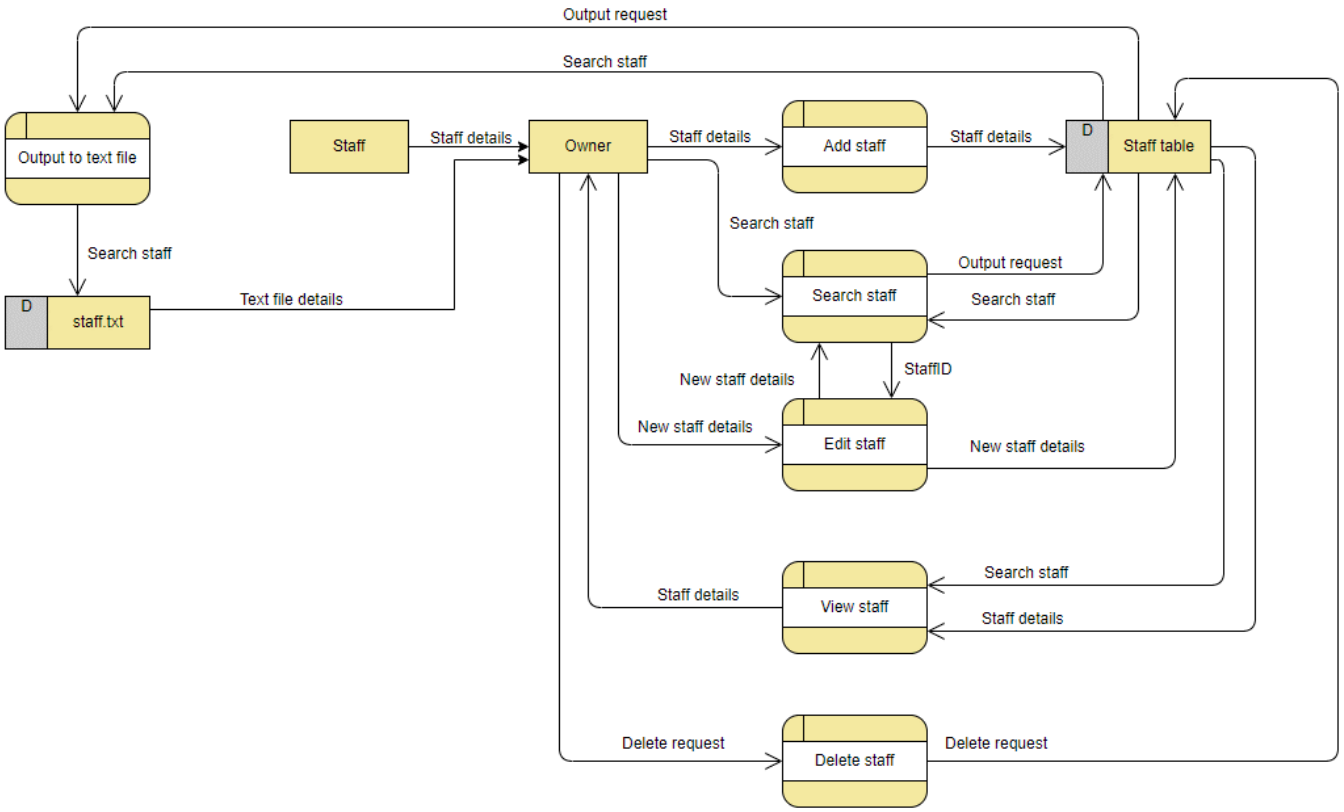
Candidate Number:
3097

Level 2 Clients DFD



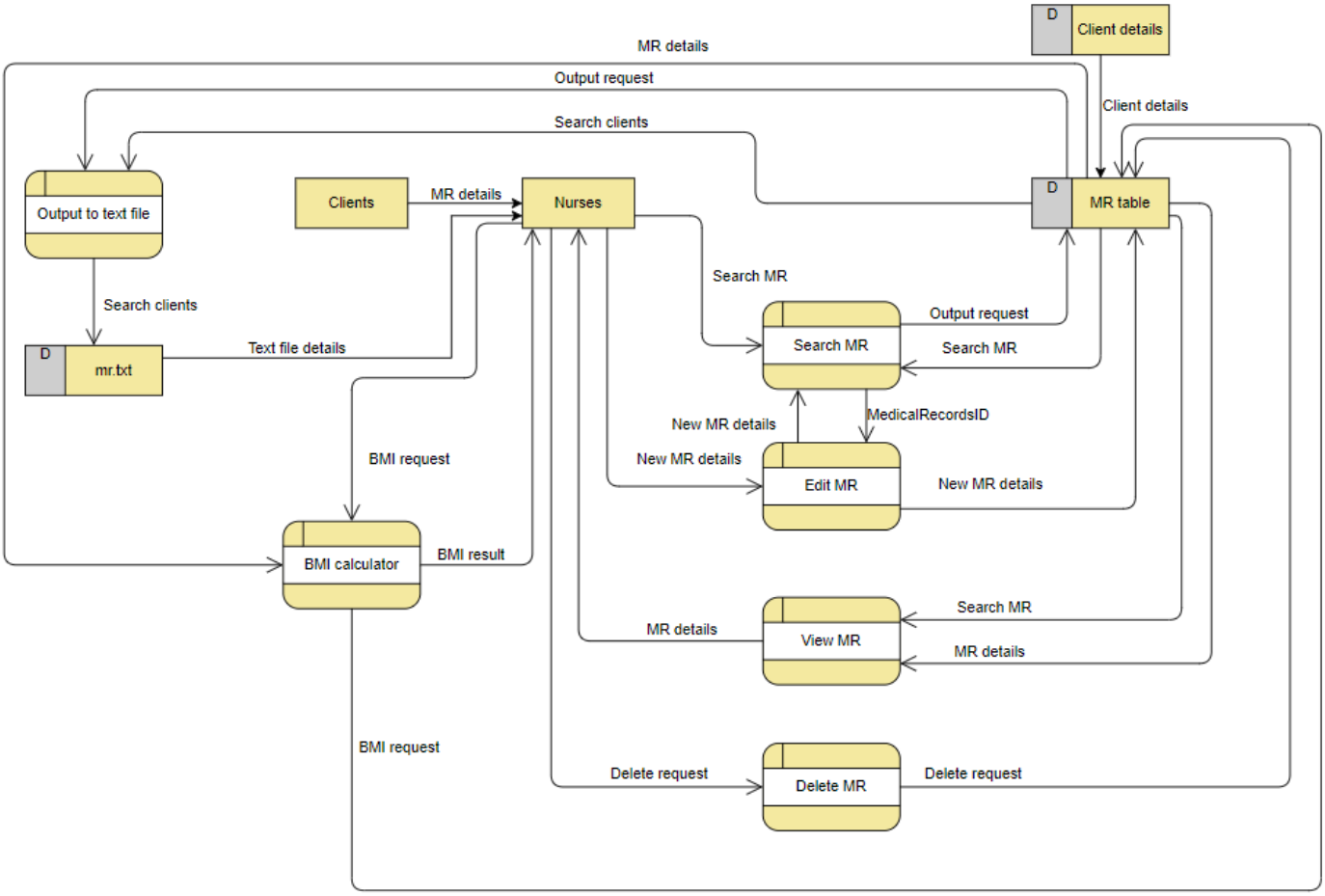
Candidate Number:
3097

Level 3 Staff DFD



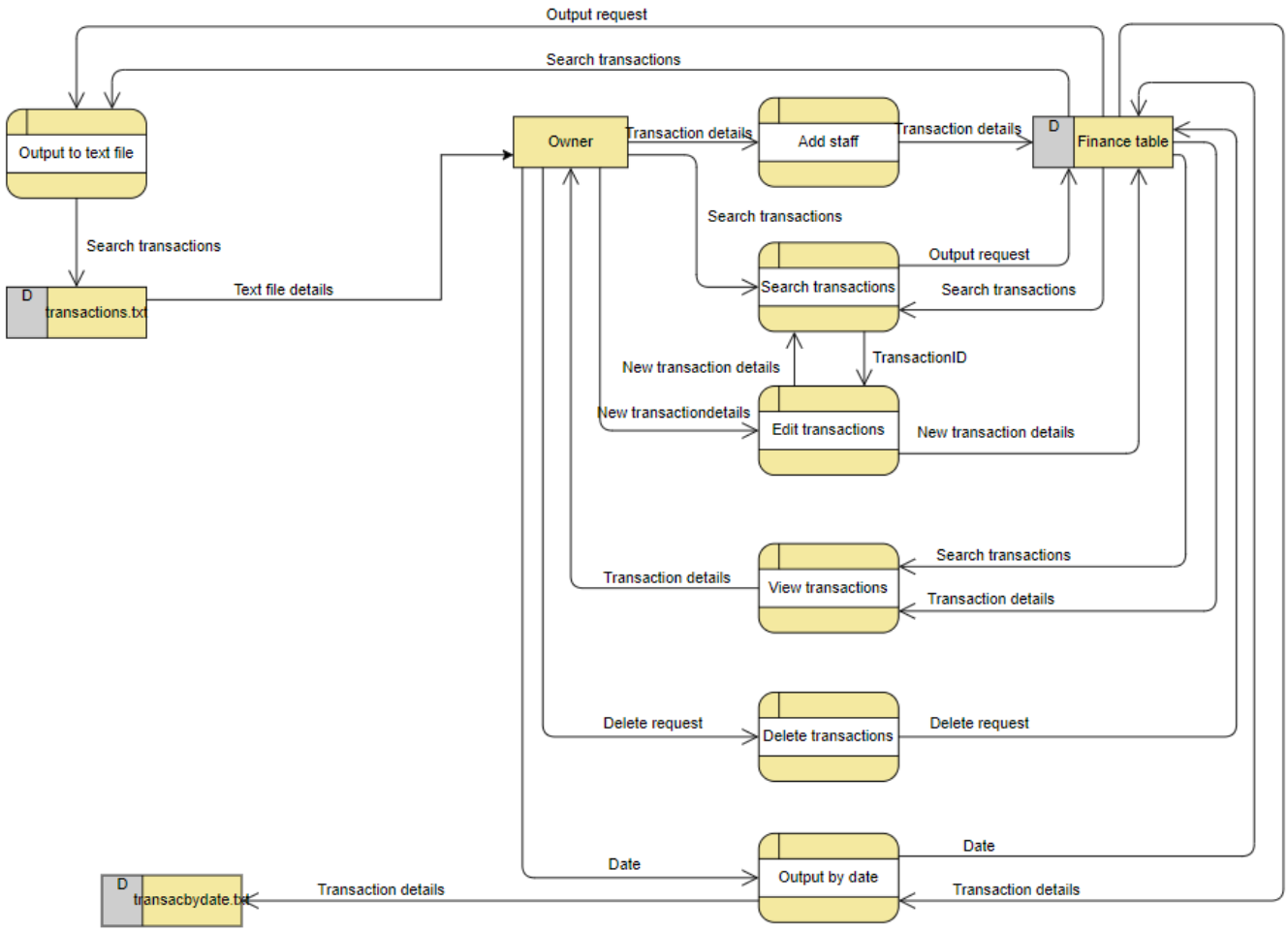
Candidate Number:
3097

Level 4 Medical records DFD



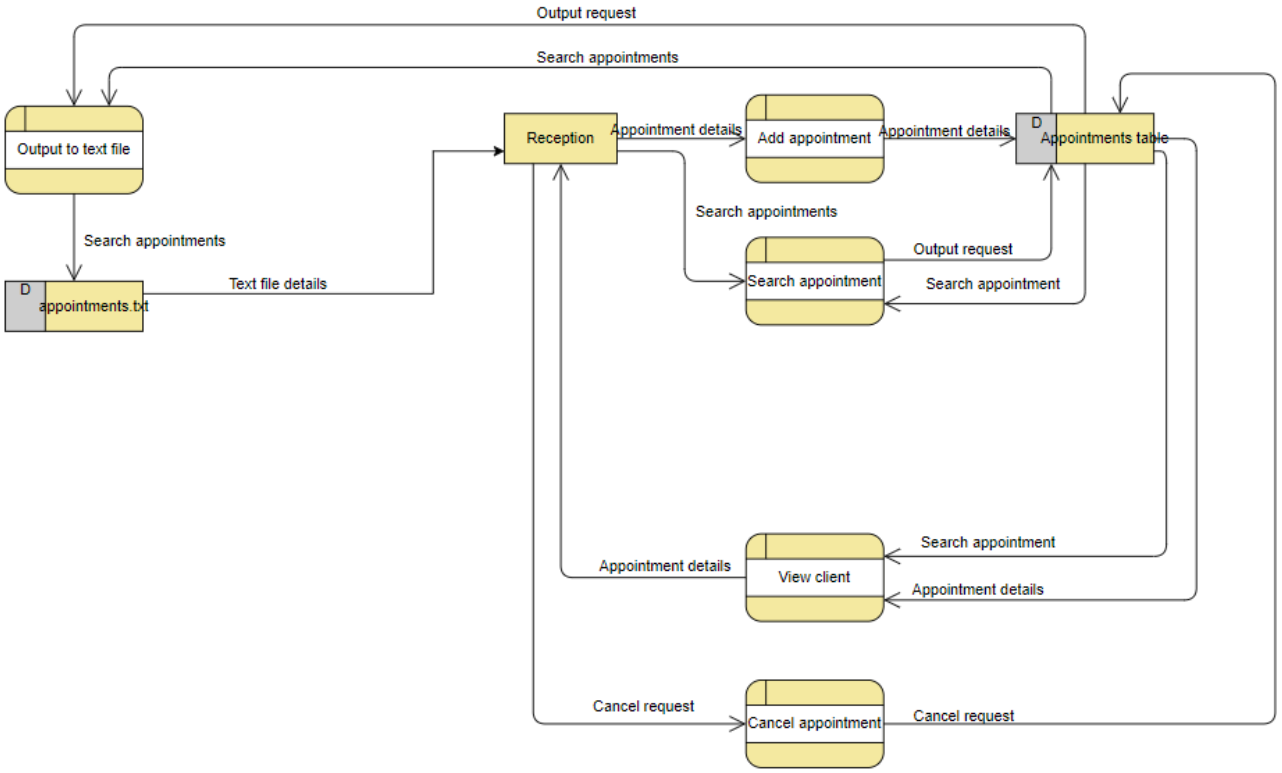
Candidate Number:
3097

Level 5 Transaction DFD



Candidate Number:
3097

Level 6 Appointments DFD

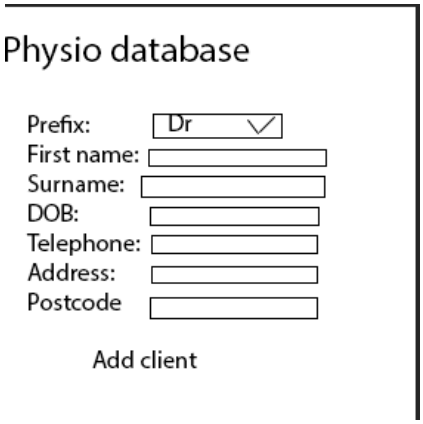


3.6: Screens of the system

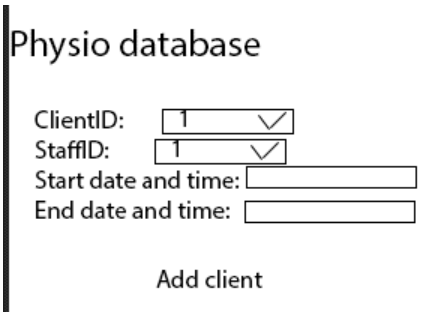
Input forms

Adding

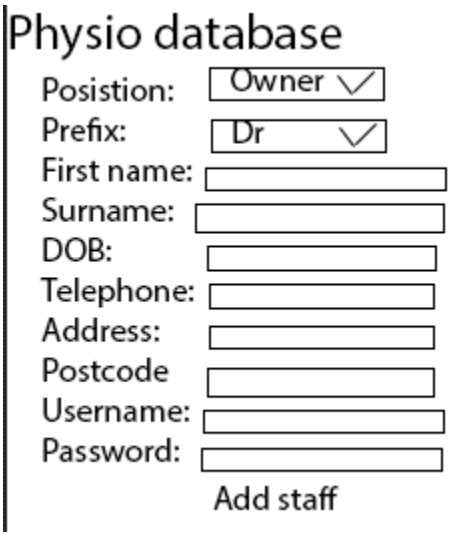
Client



Appointment

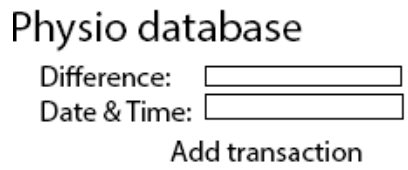


Staff



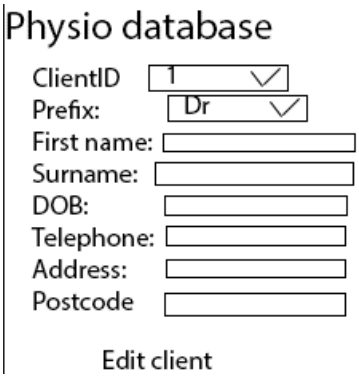
Candidate Number:
3097

Transactions

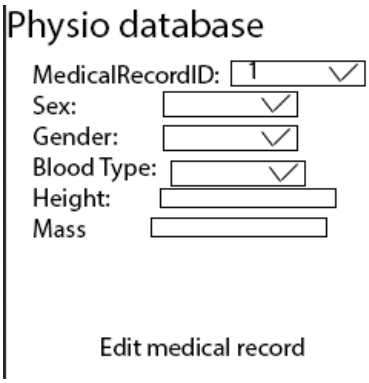


Editing

Client

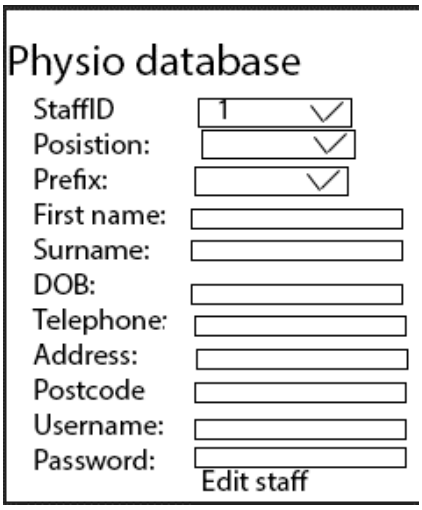


Medical records

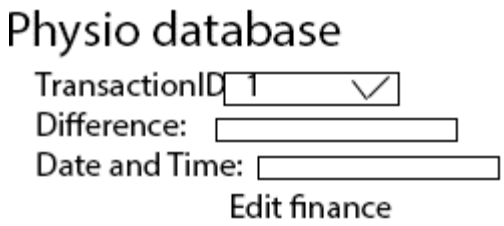


Candidate Number:
3097

Staff

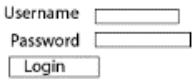


Transactions



Logging in

Login

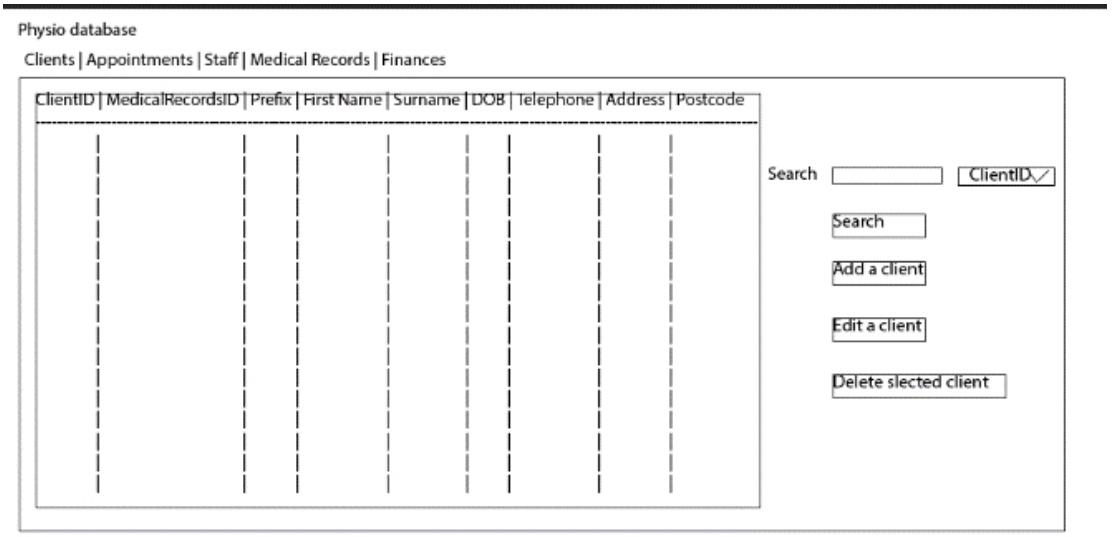


Candidate Number:
3097

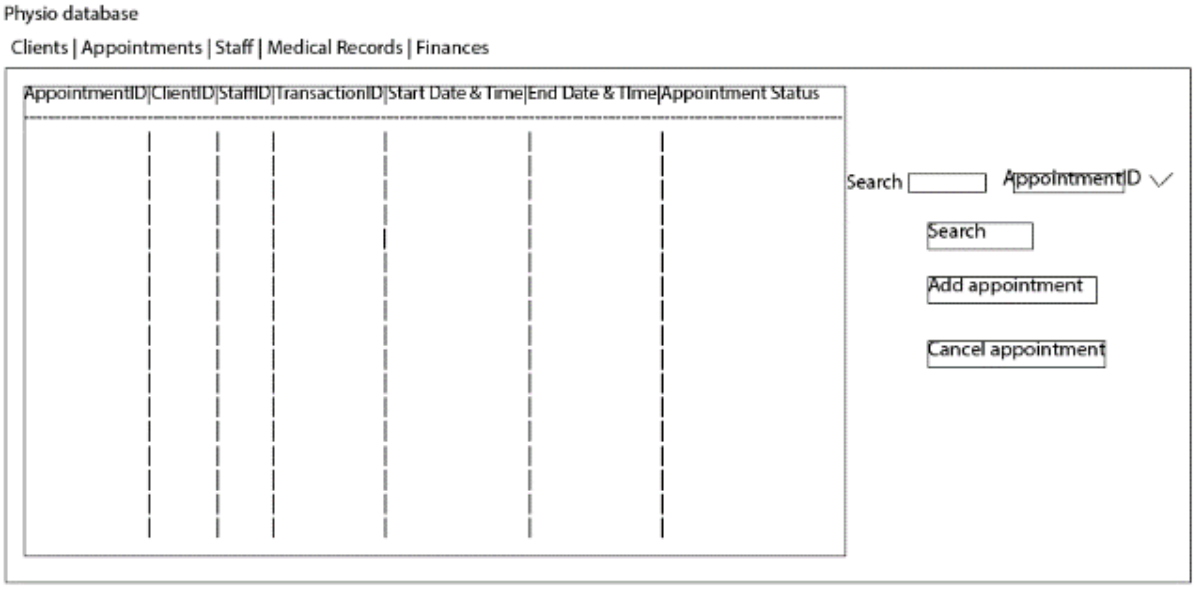
Output forms

View and search records

Client

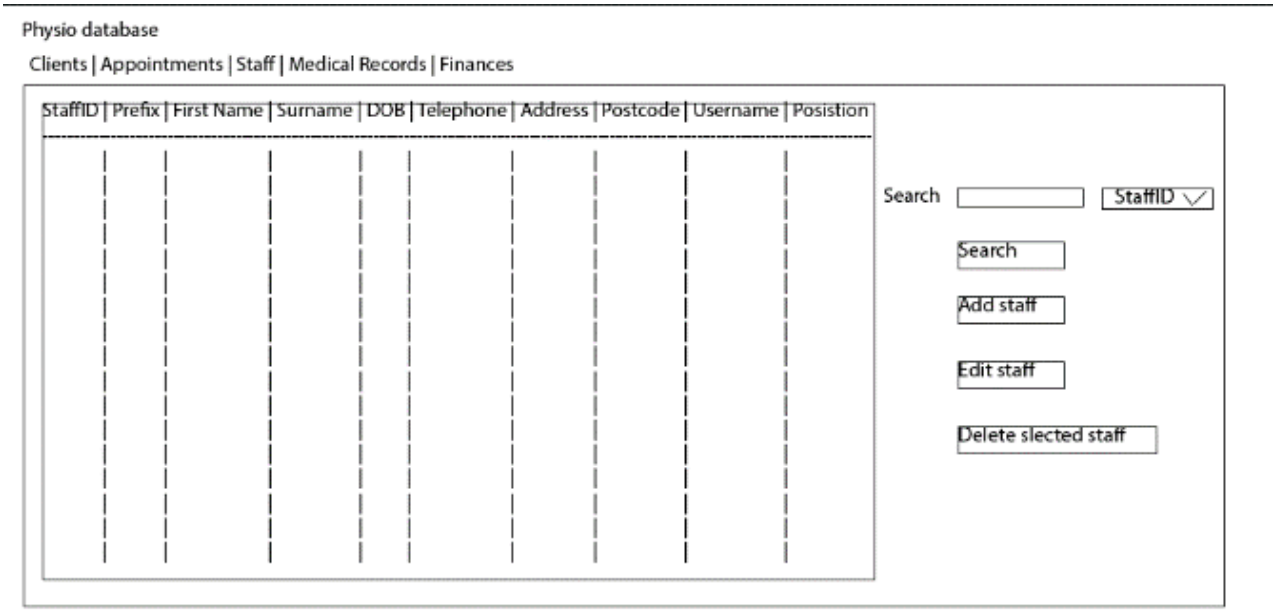


Appointment

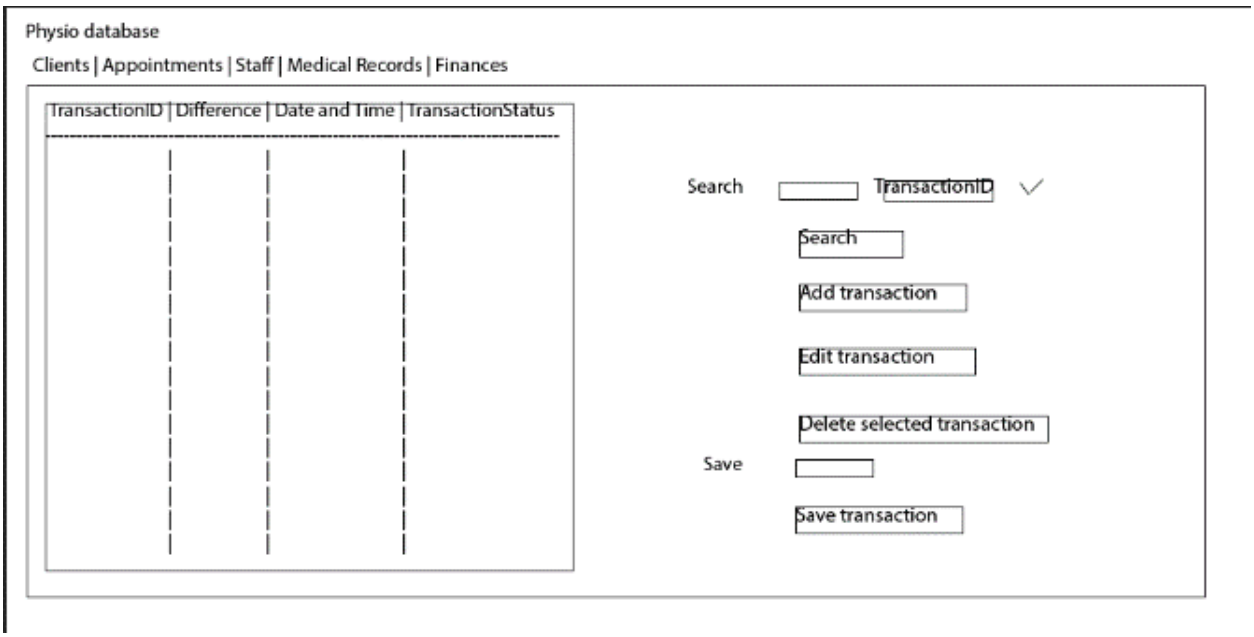


Candidate Number:
3097

Staff

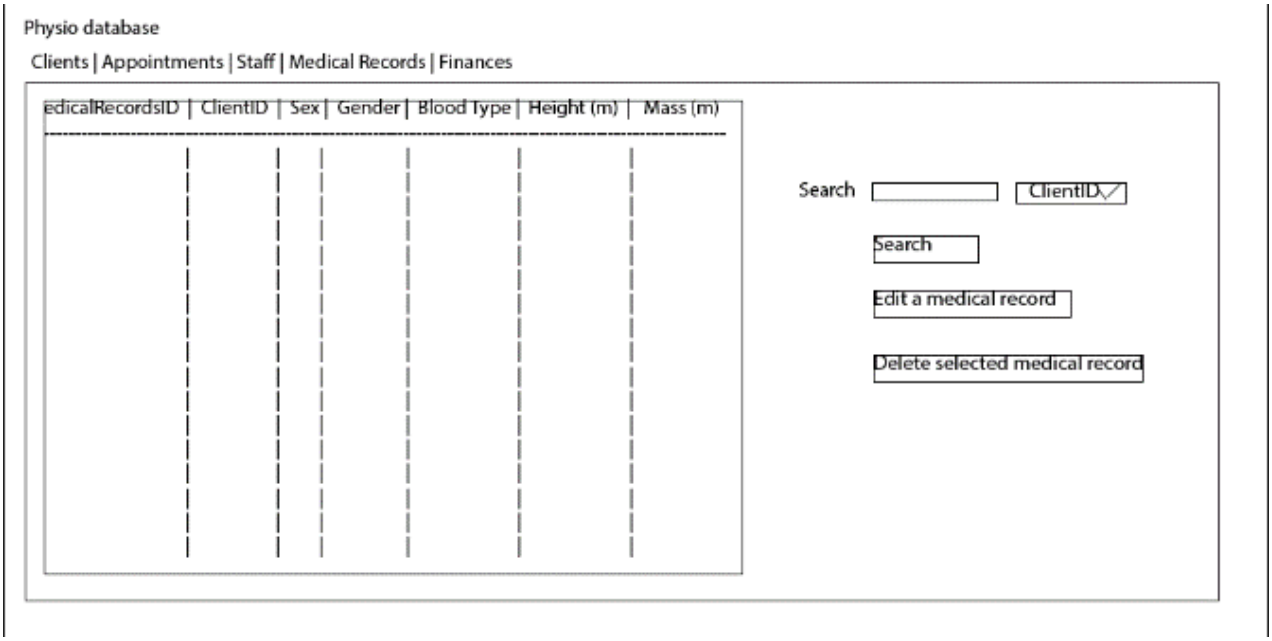


Transactions



Candidate Number:
3097

Medical records



This view will allow the user to see all the records in each of the tables and switch between each table by clicking the tab at the top of the screen. The data will be displayed within a treeview. In addition to this I will allow the user to search for a keyword for each of the columns. For example, you could use the dropdown menu to go to address and output all addresses containing '10'.

Justification of output data

I am going to output this data because it will allow the user to check the database in an intuitive and easy to use design. All data will be displayed automatically instead of other options where you would have to click a button to view the data, I have decided to do this as most of the purposes of opening the database will require looking at some of the data. In addition to viewing the data the user will also be able to search for some, this is extremely helpful and will be used on a regular basis by the staff at the clinic, for example they could look up a ClientID in the appointments table to see wen their next appointment is.

Candidate Number:
3097

Text files

Transactions for a date

```
--------------------YYYY-MM-DD--------------------
| TransactionID | Difference | TransactionStatus |
--------------------------------------------------
       0             000          Sucessful
       0             000          Sucessful
       0             000          Sucessful
       0             000          Sucessful

Overall difference: £000.0
```

This will output all the transactions of a text file, I have made a rough mockup of what this will look like above. In the program under the transaction tab there will be an input box where you can write a data and press a button to save it. It will also add the overall difference of all the transactions at the bottom.

All tables by search

Clients

```
------------------------------------ Searched for 'SEARCH' by COLUMN-------------------------------------

ClientID: ClientID
MedicalRecordID: MedicalRecordID
Prefix: Dr/Mr/Mrs/Ms/Mx/Prof/Rev
First Name: LLLLLLL
Surname: LLLLLL
DOB: YYYY-MM-DD
Telephone: NNNNNNNNNN
Address: XX XXXXXXXX XXXX
Postcode: AA0A 0AA/A0A 0AA/A0 0AA/A00 0AA/AA0 0AA/AA00 0AA
```

Candidate Number:
3097

Appointments

```
------------------------ Searched for 'SEARCH' by COLUMN------------------------

AppointmentID: AppointmentID
ClientID: ClientID
StaffID: StaffID
TransactionID: TransactionID
Start Date And Time: YYYY-MM-DD HH:MM:SS
End Date And Time: YYYY-MM-DD HH:MM:SS
Appointment Status: Active/Cancelled
```

Staff

```
------------------------------------ Searched for 'SEARCH' by COLUMN------------------------------------

StaffID: StaffID
Prefix: Dr/Mr/Mrs/Ms/Mx/Prof/Rev
First Name: LLLLLLL
Surname: LLLLLL
DOB: YYYY-MM-DD
Telephone: NNNNNNNNNNN
Address: XX XXXXXXXX XXXX
Postcode: AA0A 0AA/A0A 0AA/A0 0AA/A00 0AA/AA0 0AA/AA00 0AA
Username: LLLLLL
Position: LLLLLL
```

Transactions

```
-------------------------------------- Searched for 'SEARCH' by COLUMN--------------------------------------

TransactionID: TransactionID
Difference: 00.0
Date and Time: YYYY-MM-DD HH:MM:SS
Transaction Status: Success/Fail
```

Medical Records

```
-------------------------------------- Searched for 'SEARCH' by COLUMN--------------------------------------

MedicalRecordID: MedicalRecordID
ClientID: ClientID
Sex: M/F
Gender: Male/Female/Other
Blood Type: A+/A-/B+/B-/O+/O-/AB+/AB-
Height: 0.00
Mass: 00.00
```

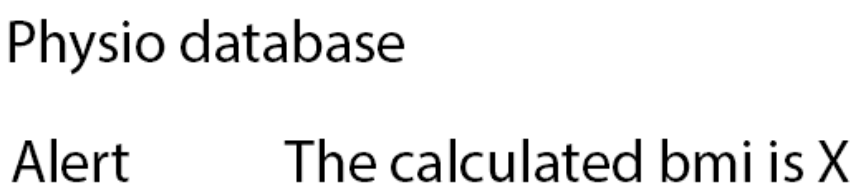
This will output a searched keyword into a textile. Above is the mockup of what this will look like and each part will be separated by a space. This will be created automatically when you search for a keyword.

Justification for output data

I've used this as it will easily allow the user to see the overall profits of a certain date in addition to being able to take the data out of the program. This will be much easier to share than a .db file and can open on more devices. It will also allow the user to copy and paste the data which you cannot do when viewing it within the program. You can also search by any keyword on the search to output all results of that keyword

Alert output

Medical records BMI calculator



When you have the medical records tab open, you can highlight a record and click the BMI button. An alert will then show for the BMI of the selected client.

Justification of output data

I will use this as it will show the staff members the BMI without having to calculate it manually and it will also be very intuitive as you simply highlight it and then the BMI simply pops up

3.7: Processing routines

Changing frame

```
INPUT current frame

INPUT new frame

DESTROY current frame

new frame = current frame
```

Logging in

```
IF login PRESSED:
        INPUT username box
        user = username box
        INPUT password box
        pass = password box
        SELECT LoggedIn WHERE Username = user AND Password = pass
        if exists:
                UPDATE LoggedIn = True WHERE Username = user box AND Password = pass
                ChangeFrame(Tab)
        else:
                OUTPUT Wrong Username/Password
        ENDIF
ENDIF
```

Candidate Number:
3097

Views

SELECT POSITION WHERE LoggedIn = TRUE

IF POSITION = IT OR = OWNER OR = RECEPTIONIST OR = PHYSIOTHERAPIST OR = NURSE:

    create client tab

IF POSITION = OWNER OR = RECEPTIONIST OR = PHYSIOTHERAPIST OR = NURSE:

    create appointment tab

IF POSITION = IT OR = OWNER:

    create staff tab

IF POSITION = OWNER OR = PHYSIOTHERAPIST OR = NURSE:

    create medical records tab

IF POSITION = OWNER:

    create finances tab

ENDIF

#IN CLIENT FRAME

SELECT POSITION WHERE LoggedIn = TRUE

IF POSITION = OWNER OR = RECEPTIONIST OR = PHYSIOTHERAPIST OR = NURSE:

    CREATE add client button

    CREATE edit client button

    CREATE delete client button

Updating a treeview

SELECT all FROM current table

CLEAR treeview

FOR every result IN table:

    ADD to treeview

ENDIF

Candidate Number:
3097

Adding a record

INSERT INTO table (ALL RECORDS)

update_treeview

Editing a record

UPDATE table SET column = column_box WHERE primary key = dropdown_menu

update_treeview

Deleting a record

INPUT highlighted box

GET primary key FROM highlighted box

DELETE from table WHERE primary key = highlighted box

update_treeview

Searching and saving a record

SELECT ALL FROM table WHERE column LIKE search_box

CLEAR treeview

CREATE table.txt

FOR every result IN table:

    ADD to treeview

    ADD to table.txt

ENDIF

Saving a transaction as a text file

SELECT ALL from table WHERE column = input_box

SELECT SUM of value

FOR row IN rows:

    WRITE TO TEXT FILE (row[0] + row[1] … + row[n-1] + row[n])

WRITE SUM of value

OUTPUT record saved

Finding the BMI of a medical record

INPUT highlighted box

GET primary key FROM highlighted box

SELECT mass, height FROM WHERE primary key = X

BMI = mass/height^2

OUTPUT BMI

3.8: Validation routines

Format check

Postcode format

INPUT postcode

match = [AA9A 9AA] OR [A9A 9AA] OR [A9 9AA] OR [A99 9AA] OR [AA9 9AA] OR [AA99 9AA]

IF match(postcode):

    return TRUE

ELSE:

    OUTPUT error

    return FALSE

ENDIF

Date format

INPUT date

match = [YYYY-MM-DD]

IF match(date):

    return TRUE

ELSE:

    OUTPUT error

    return FALSE

ENDIF

Candidate Number:
3097

Date and time format

```
INPUT dateandtime

match = [YYYY-MM-DD HH:MM:SS]

IF match(dateandtime):
        return TRUE
ELSE:
        OUTPUT error
        return FALSE
ENDIF
```

Difference format

```
INPUT difference

match = [+/-] + [DD]

IF match(difference):
        return TRUE
ELSE:
        OUTPUT error
        return FALSE
ENDIF
```

Length check

More than length check:

```
INPUT result

IF len(result) < max:
        return TRUE
ELSE:
        OUTPUT error
        return FALSE
ENDIF
```

Candidate Number:
3097

Less than length check:

```
INPUT result

IF len(result) > min:
        return TRUE
ELSE:
        OUTPUT error
        return FALSE
ENDIF
```

Exact length check:

```
INPUT result

IF len(result) = exact:
        return TRUE
ELSE:
        OUTPUT error
        return FALSE
ENDIF
```

Presence check

```
INPUT result

IF result = “”:
        OUTPUT error
        return FALSE
ELSE:
        return TRUE
ENDIF
```

Candidate Number:
3097

Range check

Larger than range check:

```
INPUT result

IF result < max:
        return TRUE
ELSE:
        OUTPUT error
        return FALSE
ENDIF
```

Less than range check:

```
INPUT result

IF result > min:
        return TRUE
ELSE:
        OUTPUT error
        return FALSE
ENDIF
```

Type check:

Integer check

```
INPUT result

IF result.isinteger():
        return TRUE
ELSE:
        OUTPUT error
        return FALSE
ENDIF
```

Candidate Number:
3097

Alphabetic character check

```
INPUT result

IF result.isalpha():
        return TRUE
ELSE:
        OUTPUT error
        return FALSE
ENDIF
```

Float check

```
INPUT result

IF result.isreal():
        return TRUE
ELSE:
        OUTPUT error
        return FALSE
ENDIF
```

Lookup check:

```
INPUT result

list = [0, 1, 2, n-1, n]

IF result IN list:
        return TRUE
ELSE:
        OUTPUT error
        return FALSE
ENDIF
```

Candidate Number:
3097

Editing without any values:

```
SELECT ID FROM table

TRY:

        cursor.fetchone()

EXCEPT IndexError:

        return FALSE

        OUTPUT error
```