Name: Sam Richell

Candidate Number: 1593

NEA PROJECT – VOLUNTEER DATABASE

Name: Sam Richell

Candidate Number: 1593

Centre Number: 64930

CONTENTS

ANALYSIS

THE SPECIFIC PROBLEM

Lightwater Connected is a society for volunteers to sign up to in Lightwater so that they can be part of a greater volunteer network within Lightwater. Lightwater Connected keep track of all volunteers and their details along with events that they run and the tasks that they carry out. Examples of these duties which the volunteers carry out include running the annual Lightwater Village Fete, offering trips to hospitals for those who would otherwise have to pay for expensive travel services and making sure the local area is well kept by picking up rubbish, looking after memorials and repairing broken benches. The members at Lightwater Connected have shown that they find it difficult to use the current system at play, since the software currently in use is not designed at all for how Lightwater Connected are using it.

(Lightwater Connected, 2023)

HOW WILL THIS PROBLEM BE FIXED?

The most obvious solution is a database which will store all volunteer data from names and emails to local events and who has volunteered when and where. If a bespoke database, designed specifically for Lightwater Connected and Lightwater Connected's needs specifically, is implemented correctly within Lightwater Connected, it could solve many of the problems which they have faced.

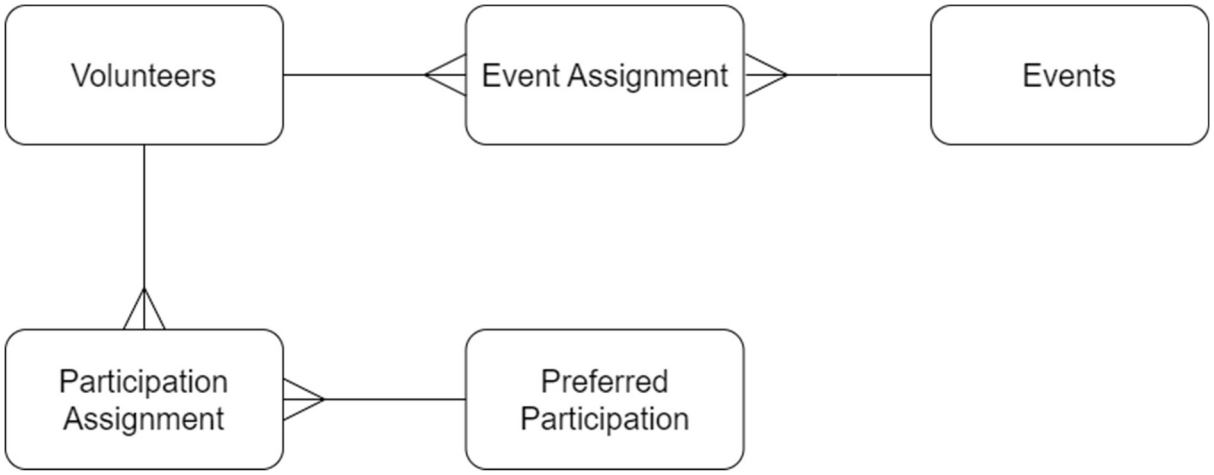
WHAT ARE DATABASES?

Databases are used by every business in the modern day, from grocery stores and restaurants to banks and hospitals. Every website has a database behind it as well. There is a huge demand for the storage of data all around the world, whether it be storing account details, current stock, or code for a website. The ability to store substantial amounts of data automatically all in one place is extremely useful. Oracle are a huge cooperation worth £297.5 billion (macrotrends, 2023) whose flagship product is Oracle Database (Wikipedia, 2023).

WHAT SHOULD THE FINAL PRODUCT LOOK LIKE?

The final system should involve a bespoke program which interacts directly with the database and should be able to manipulate the database. This includes retrieving data from the database, altering data in the database, adding, or removing tables and even grouping data together, the software must also be able to automate sending emails to the volunteers. Another part of the system will include a web page in which volunteers can enter their details and their preferred participation. Finally, the system will include an SQL database storing all the data that Lightwater Connected need to function.

I have created a sketch of an ERD (Entity Relationship Diagram) for what the database might look like:



Having discussed briefly with the leadership at Lightwater Connected, I can conclude that they biggest problem that I must solve is simplicity. The people who volunteer at Lightwater Connected tend to be older and less experienced at using technology, this requirement rules out plenty of other options that Lightwater Connected could use such as something considered simple like Microsoft Access.

WHO WILL BENEFIT?

Those who are most likely to benefit from this new system will be the leadership at Lightwater Connected and the volunteers at Lightwater Connected. Those who are associated with Lightwater Connected (both leadership and volunteers) are more commonly older people who have less experience with technology, to combat this, the front end of the system will provide a user-friendly interface. This will enable the leadership to easily access and manage the data without having to hire an expert to manage a database for them.

Another demographic that will benefit from this new system would be the general population of Lightwater. The new system would help save time and resources meaning the volunteers at Lightwater Connected can spend more time helping around the village and therefore Lightwater will be a nicer place for everyone.

ADDRESSING IMPAIRMENTS

A key consideration for the new system is of course inclusivity. To ensure accessibility and inclusivity, the system will be designed with these considerations:

- Visual impairments
    - To combat the struggles that those with visual impairments, the new system could include high contrast modes or adjustable font sizes.
- Hearing impairments
    - To combat the struggles of those with hearing impairments, the new system could ensure that there is no content which those with hearing impairments will miss out on such as audio files or videos.
- Movement impairments
    - To combat the struggles of those with movement impairments, the new system could include keyboard shortcuts and the interface could be designed to minimize precise mouse movements.

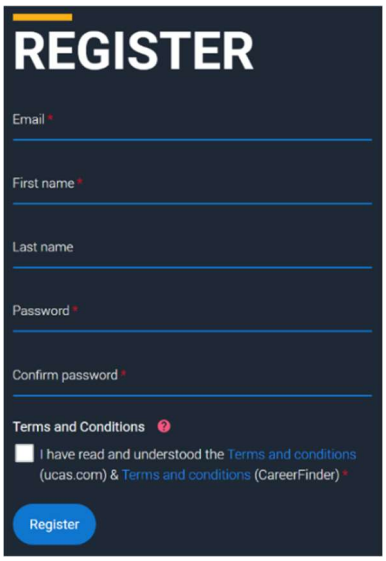
By addressing these issues, it will ensure that the volunteers at Lightwater Connected will feel included and may help bring more volunteers to the group further benefitting the whole community.

Her is some other web pages designed with accessibility in mind:

(BBC, 2023)

The sign-up page for the BBC includes:

- Clear labels
- Clear and complete instructions
- Information on what they use the data for
- Obvious button
- Clear contrast for what needs to stand out.



The sign-up page for UCAS includes:

- Clear labels
- Clear and complete instructions
- Obvious button
- Extra information

(UCAS, 2023)

CURRENT SYSTEM

Currently, they rely on 'Mailchimp' which is an email marketing program. Mailchimp is an expensive and complicated program with an array of features which are useless or too complicated for use by some of the less "tech savvy" members of Lightwater Connected.

Mailchimp is an immensely powerful piece of software with features including:

- Easy email automation
- Generating emails with AI
- Targeting specific demographics with email

(Mailchimp, 2023)

Mailchimp is clearly an email marketing software and therefore very clearly not built for what Lightwater Connected need. Although you can store details of - in this case - the volunteers, it is extremely inefficient and overly expensive and a waste of resources.

Having spoken to the head of Lightwater Connected, he claims that using Mailchimp is complicated and useless and a waste of money. The head of Lightwater Connected would like a simple piece of software that he, his colleagues and the volunteers can use with ease.

INTERVIEW

As part of my research, I interviewed the main stakeholders (leadership) at Lightwater Connected to find out more about what is needed for the system.

INTERVIEW QUESTIONS

1. What is Lightwater Connected aiming to achieve?
2. What specific features do you need the new system to include?
3. What features do you want the new system to include?
4. What are the main problems you have with the current system?
5. What are your main concerns with the new system?
6. What other options have you looked at?
7. What would lead you to consider a new system as a success?
8. Do you have any insight that you believe may help?

INTERVIEW TRANSCRIPT

SR – Sam Richell (me)
LC – Lightwater Connected leader.

SR: “What is Lightwater Connected aiming to achieve?”
LC: “A flexible working list of volunteers and their prospective jobs. The interest is on flexibility. With the ability to communicate to volunteers of all matters. Communicating via preferred participation”
SR: “What specific features do you need the new system to include?”
LC: “Entering details. Communication is the biggest point, using the database on a practical basis for everyday activities. I’d rather have specifics.”
SR: “What are the main problems you have with the current system?”
LC: “Communication. Communication and changing the data within the database. If I’m sending an email to volunteers the current system cannot handle attachments.”
SR: “What are your main concerns with the new system?”
LC: “That maintenance will be difficult once you’ve moved on. Who is going to do the ongoing maintenance. How is it going to be hosted, where will it be run and how do we access it and how will it be kept up to date with security patches.”
SR: “What do you mean by security?”
LC: “Making sure that if this is available as a website, we need to be confident that hackers cannot access or damage the database”
SR: “What other options have you looked at?”
LC: “We haven’t, we have it as an idea, and we need someone to do it.”
SR: “What would lead you to consider a new system as a success?”
LC: “It would be a success if we can have people volunteering, updating their data and there not being a hassle.”

QUESTIONNAIRE

Do you find the current system confusing? *

- ◯ Yes
- ◯ No
- ◯ Don't Know

What information would you be comfortable sharing with Lightwater Connected? * (Tick all that apply)

- ☐ Full name
- ☐ E-mail
- ☐ Mobile number
- ☐ Address

How would you rather receive information? *

- ◯ E-Mail
- ◯ Phone

How often do you check your personal email? *

- ◯ More than once a day
- ◯ Once a day
- ◯ Less than once a day

How comfortable would you be with an online account with Lightwater Connected? *

	1	2	3	4	5	
Very Uncomfortable	○	○	○	○	○	Very Comfortable

Other comments

Your answer

OTHER SOLUTIONS

Having attempted to research databases for storing the data that is needed I have discovered that there are not any options that could be used specifically for what Lightwater Connected need, only other programs which offer a main service which would not be useful to Lightwater Connected. As stated earlier, these programs must also be simple to use which rules out plenty of other options for Lightwater Connected.

Despite this, other solutions that Lightwater Connected could implement include:

- Microsoft Access
    - Pros:
        - Built into all Windows PCs.
        - Simpler than most solutions.
    - Cons:
        - Requires an understanding of databases.
- Wix
    - Pros:
        - Includes a website.
    - Cons:
        - Expensive
        - Not designed specifically for what Lightwater Connected need.
- Oracle Database
    - Pros:
        - Designed as a database that would help Lightwater Connected
    - Cons:
        - Very expensive

Small businesses often pay copious amounts for an external business to build a database for them with the same sort of requirements as the requirements of Lightwater Connected. Lightwater Connected does not have this money to spend on a bespoke software developed by an external business.

From this we can conclude that any software that is available at this time will not be helpful for Lightwater Connected and that a new system should be developed to aid them.

OBJECTIVES

From the research that has be conducted, an objectives list can now be formed highlighting what the end users will need the final solution to provide.

1. Sign-up page for new volunteers.
    1.1. Must be a web page.
    1.2. Must be user-friendly.
        1.2.1. Clear labels telling volunteers exactly where to input information.
        1.2.2. Clear button to submit application when finished.
    1.3. Must be able to enter details
        1.3.1. Must be able to enter full name.
        1.3.2. Must be able to enter email address.
        1.3.3. Must be able to create password.
        1.3.4. Must be able to enter mobile number.
        1.3.5. Must be able to enter full address.
        1.3.6. Must be able to enter preferred participation data.
            1.3.6.1. Must update when new preferred participation is added.
2. Program designed to access volunteer data.
    2.1. Program must only be able to be accessed by leadership at Lightwater Connected.
        2.1.1. Those trying to access system must use a pre-defined password.
        2.1.2. Password page must be user friendly.
            2.1.2.1. Clearly defined labels.
    2.2. Program must be able to access database.
    2.3. Program must be able to return a list based on certain selected parameters.
        2.3.1. Must return a list based on name.
        2.3.2. Must return a list based on postcode.
        2.3.3. Must return all data of each specific volunteer
    2.4. Program must be able to add preferred participation.
    2.5. Program must be able to delete preferred participation.
    2.6. Program must be able to add events.
    2.7. Program must be able to delete events.
    2.8. Program must be user-friendly.
        2.8.1. Program must have clear labels.
        2.8.2. Program must show obvious contrast as to what each button does.
        2.8.3. Program must produce easy to read data.
        2.8.4. Program must produce printable data.
            2.8.4.1. Program must produce CSV file that can be opened and printed within excel.
    2.9. Program must be able to contact volunteers.
        2.9.1. Program must be able to send emails.
3. Volunteers must be able to login.
    3.1. Must be web page.
    3.2. Must be user-friendly.
        3.2.1. Must have clear labels.
        3.2.2. Must have obvious buttons.
    3.3. Must be able to enter email.
    3.4. Must be able to enter password.

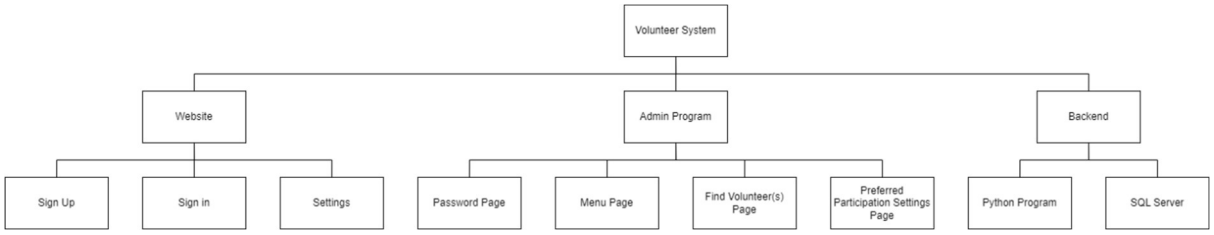
- 3.5. Page must authenticate users.
- 3.6. Page must direct users to their details.
- 3.7. Automated password recovery.
  - 3.7.1. Must send volunteer email with password.

4. Volunteers must be able to alter their own data.
   - 4.1. Must be web page.
   - 4.2. Must be user friendly.
     - 4.2.1. Must have clear labels.
     - 4.2.2. Must have obvious buttons.
   - 4.3. Must be able to alter volunteer data.
     - 4.3.1. Must be able to change name.
     - 4.3.2. Must be able to change email.
     - 4.3.3. Must be able to change password.
     - 4.3.4. Must be able to change mobile number.
     - 4.3.5. Must be able to change full address.
     - 4.3.6. Must be able to change preferred participation.
   - 4.4. Must be able to sign out.
5. Database must store data.
   - 5.1. Must store all volunteer data.
     - 5.1.1. Must store full names.
     - 5.1.2. Must store passwords.
     - 5.1.3. Must store emails.
     - 5.1.4. Must store mobile numbers.
     - 5.1.5. Must store full address.
   - 5.2. Must store preferred participation data.
     - 5.2.1. Must store name.
     - 5.2.2. Must store description.
     - 5.2.3. Must store assignment.
   - 5.3. Must store events.
     - 5.3.1. Must store name.
     - 5.3.2. Must store location.
     - 5.3.3. Must store date and time.
     - 5.3.4. Must store assignment.

DESIGN

To simplify the design, each point from the objectives (1-5) will be designed separately, sections within may be split up further.

STRUCTURE DIAGRAM

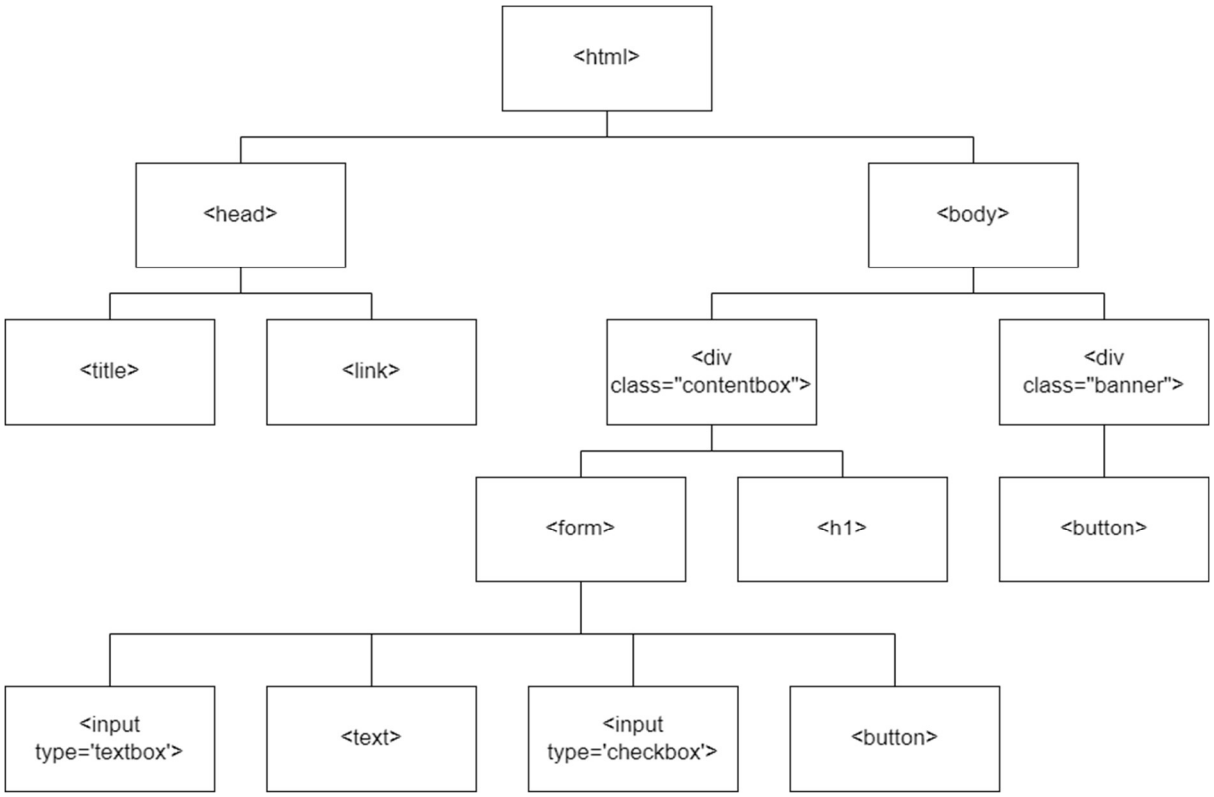


WEBSITE

Each page will inlcude:

- a content box in the centre.
- a banner across the top displaying a log out button and logo of organisation.

HTML STRUCTURE DIAGRAM



PAGE BANNER

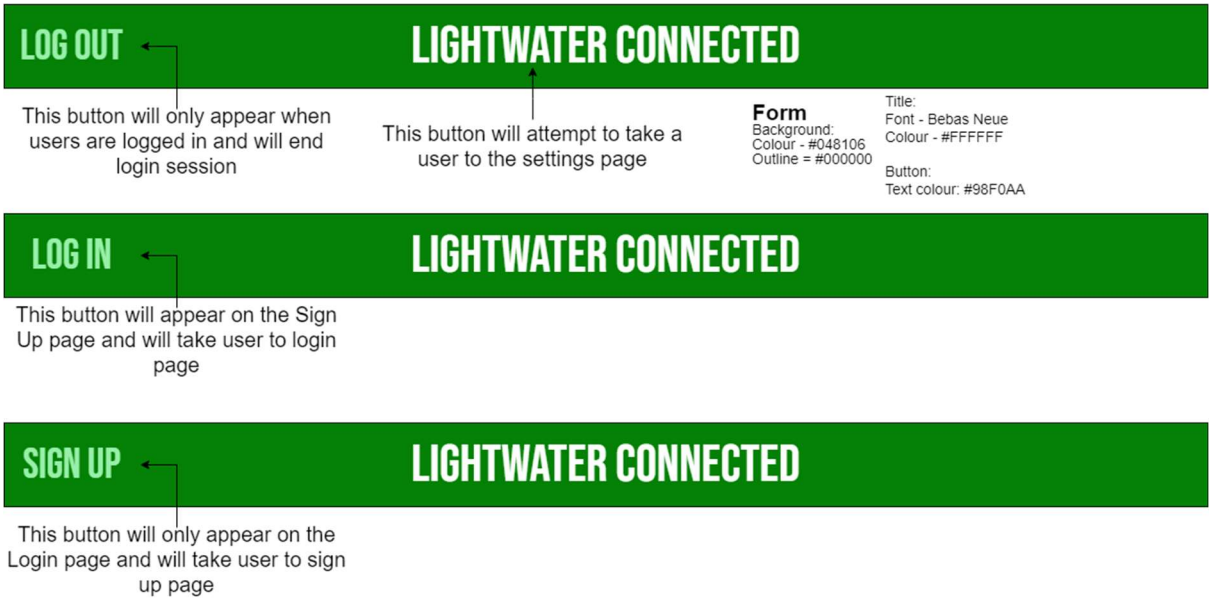
Name: Sam Richell

Candidate Number: 1593

IPSO CHART

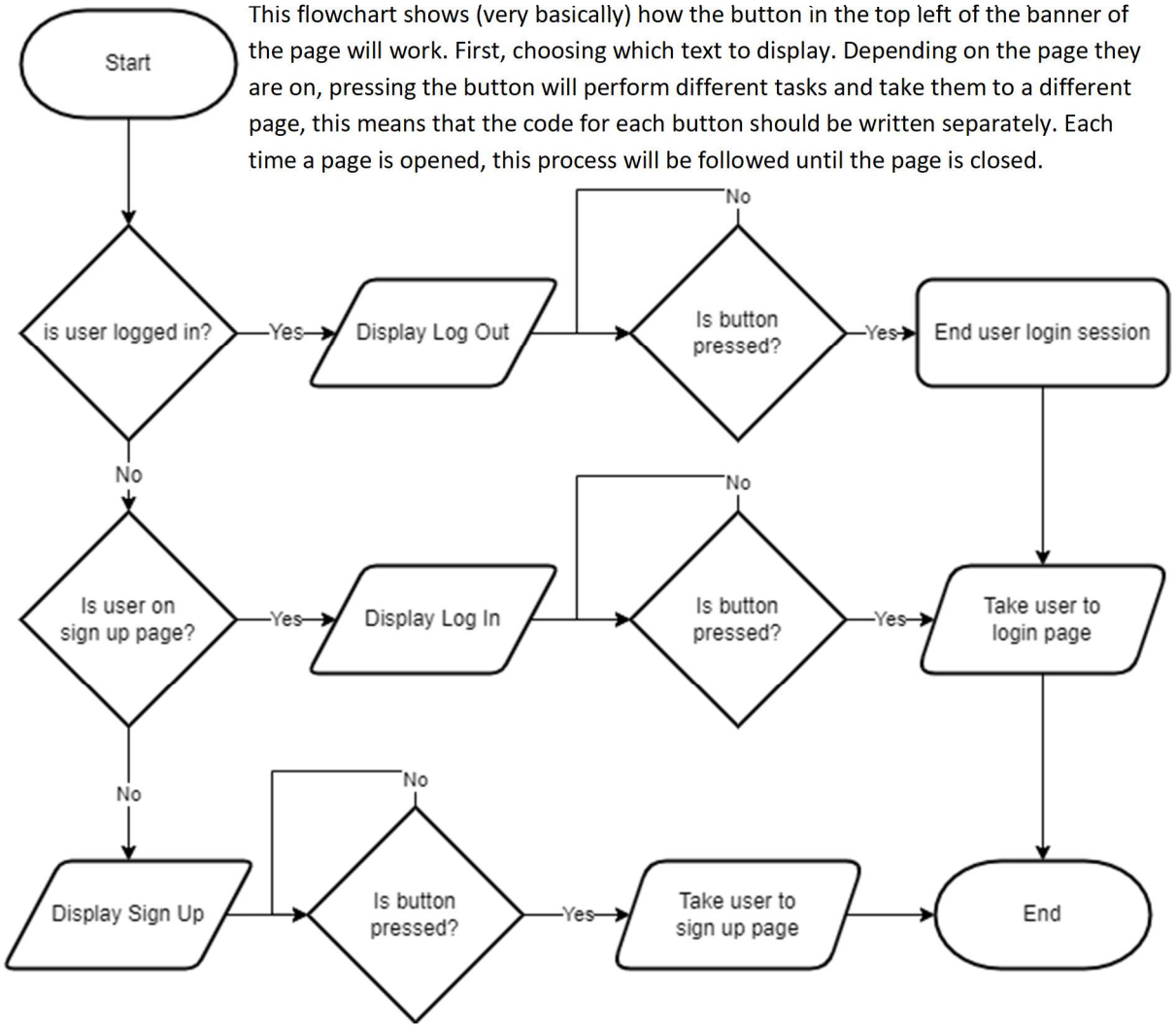
INPUTS	PROCESSES	STORAGE	OUTPUT
Log out button. Home button.	End login session when user logs out	Login session	Take to next page

GUI DESIGN



FLOWCHARTS

BANNER BUTTON



This flowchart shows (very basically) how the button in the top left of the banner of the page will work. First, choosing which text to display. Depending on the page they are on, pressing the button will perform different tasks and take them to a different page, this means that the code for each button should be written separately. Each time a page is opened, this process will be followed until the page is closed.

SIGN UP PAGE

The sign-up page will be a website designed with HTML, styled with CSS and will communicate with the server using a separate php file.

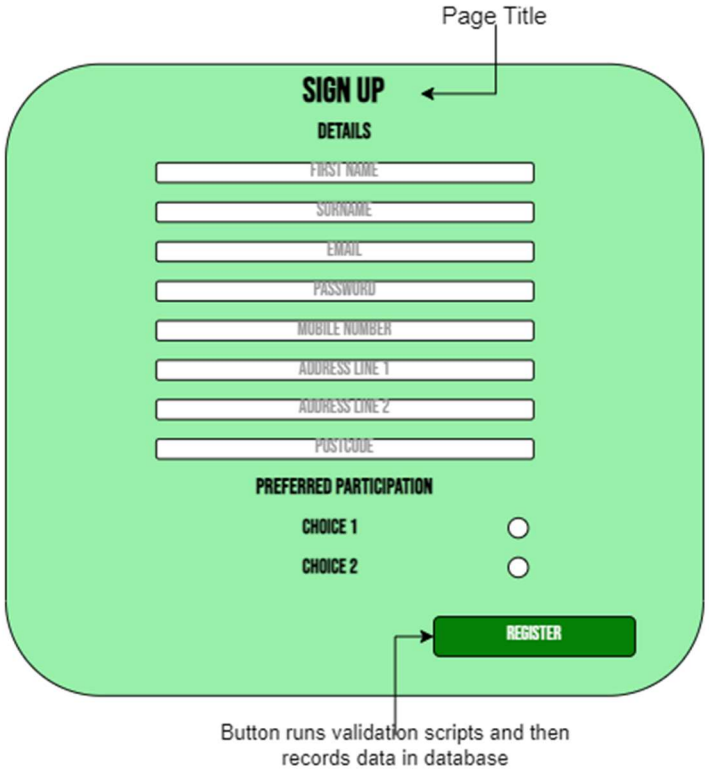
IPSO CHART

INPUTS	PROCESSES	STORAGE	OUTPUT
Full name Email address Password Mobile number	User has not already registered. Password is secure. Inputs are valid.	Store data (to SQL server)	Confirm or deny account set-up. Suggest changes.

Full address Preferred Participation Create Account button			

GUI DESIGN

DATA ITEM	DATA TYPE	VALIDATION/RESTRICTIONS
Forename	String	-Required -Only letters
Surname	String	-Required -Letters and spaces only
Email Address	String	-Required -Not already been used -Must follow common format
Password	String	-Required -Must be at least 8 characters -Must contain capital letter and number
Mobile Number	Integer	-Required -Must be 11 numbers
Full Address	String	-Required -2 address lines and postcode
Preferred Participation	Boolean	-Required -Checkboxes



PSEUDOCODE

SQL INSERT STATEMENT

```
INSERT INTO Volunteers (Forename, Surname, Email, Password, MobileNumber, Address, Postcode) VALUES ({entered forename}, {entered surname}, {entered email}, {entered password}, {entered mobile number}, {entered address}, {entered, postcode});
For ParticipationID in selected preferred participation
      INSERT INTO ParticipationAssignment (VolunteerID, ParticipationID) VALUES ({VolunteerID}, {ParticipationID});
```

LOGIN PAGE

The login page will be a web page for volunteers to log into their account.

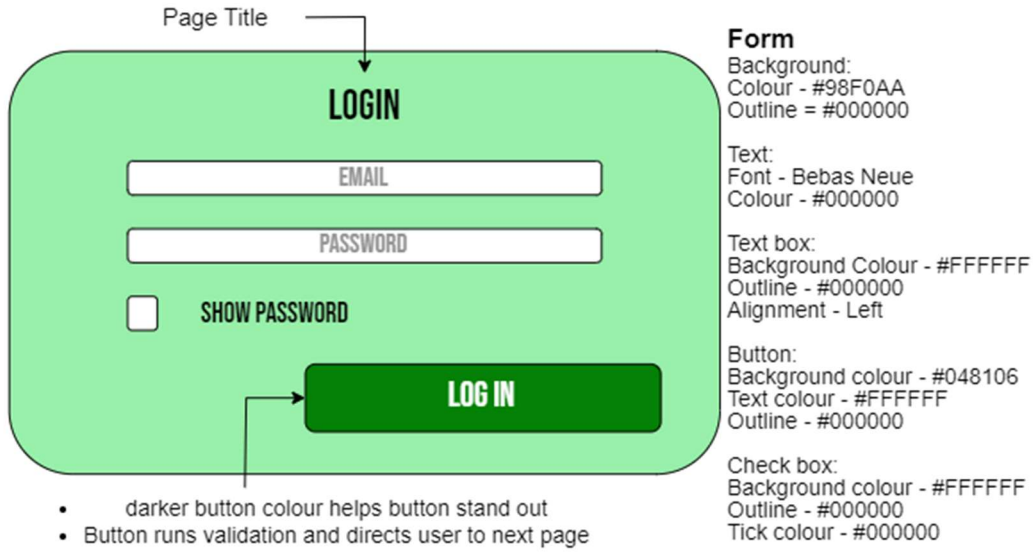
IPSO CHART

INPUTS	PROCESSES	STORAGE	OUTPUT
Email Password Forgot Password	Decrypt password Are email and password the same?	Retrieve data (from SQL server) Login session	Take user to settings page

Login button			

GUI DESIGN

DATA ITEM	DATA TYPE	VALIDATION/RESTRICTIONS
Email Address	String	-Required -Must follow common format
Password	String	-Required



PSEUDOCODE

```
SELECT VolunteerID FROM Volunteers WHERE Email={entered email} AND
Password={entered password}
If length(returned) == 0
      Login
Else
      Print("Wrong username or password")
```

This will log the user in if there is a volunteer on the database with the entered email and entered password.

SETTINGS PAGE

The settings page will be a page where users can change details relating to their account or their preferred participation.

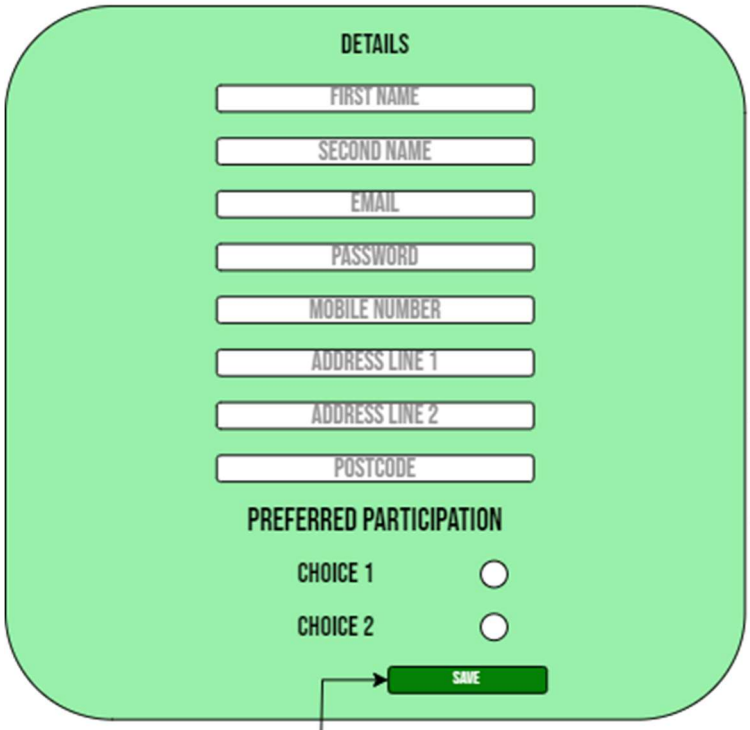
IPSO CHART

INPUTS	PROCESSES	STORAGE	OUTPUT
Alter details. Log out.	Write changed data to DB. Encrypt/ decrypt data	Retrieve details (from SQL server)	Data from DB Setting changed notification.

		End login session (if log out)	Error notification if error.

GUI DESIGN

DATA ITEM	DATA TYPE	VALIDATION/RESTRICTIONS
Forename	String	-Required -Only letters
Surname	String	-Required -Letters and spaces only
Email	String	-Required -Not already been used -Must follow common format
Password	String	-Required -Must be at least 8 characters -Must contain capital letter and number
Mobile Number	Integer	-Required -Must be 11 numbers
Full Address	String	-Required -2 address lines and postcode
Preferred Participation	Boolean	-Required -Checkboxes



Form
Background:
Colour - #98F0AA
Outline = #000000

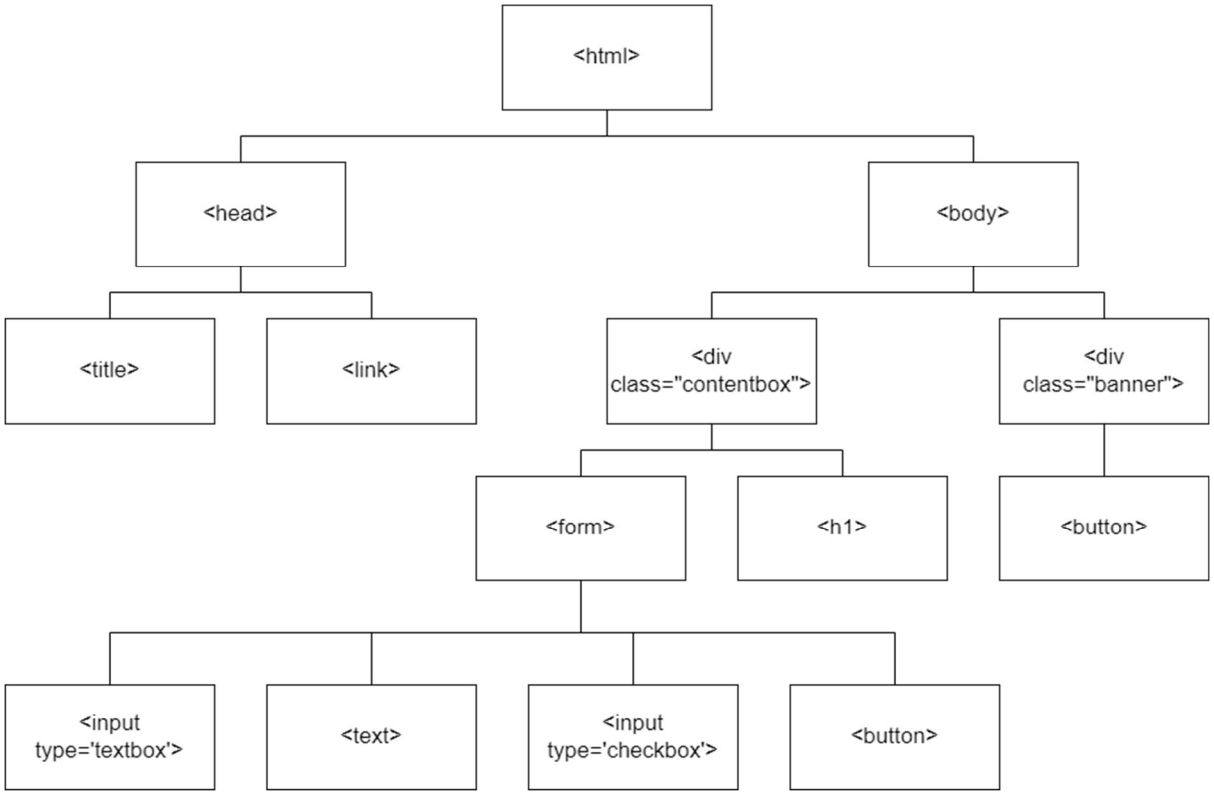
Text:
Font - Bebas Neue
Colour - #000000

Text box:
Colour - #FFFFFF
Outline - #000000
Alignment - Centre

Button:
Background colour - #048106
Text colour - #FFFFFF
Outline - #000000

Check box:
Background Colour - #FFFFFF
Outline - #000000

HTML HEIARCHY CHART

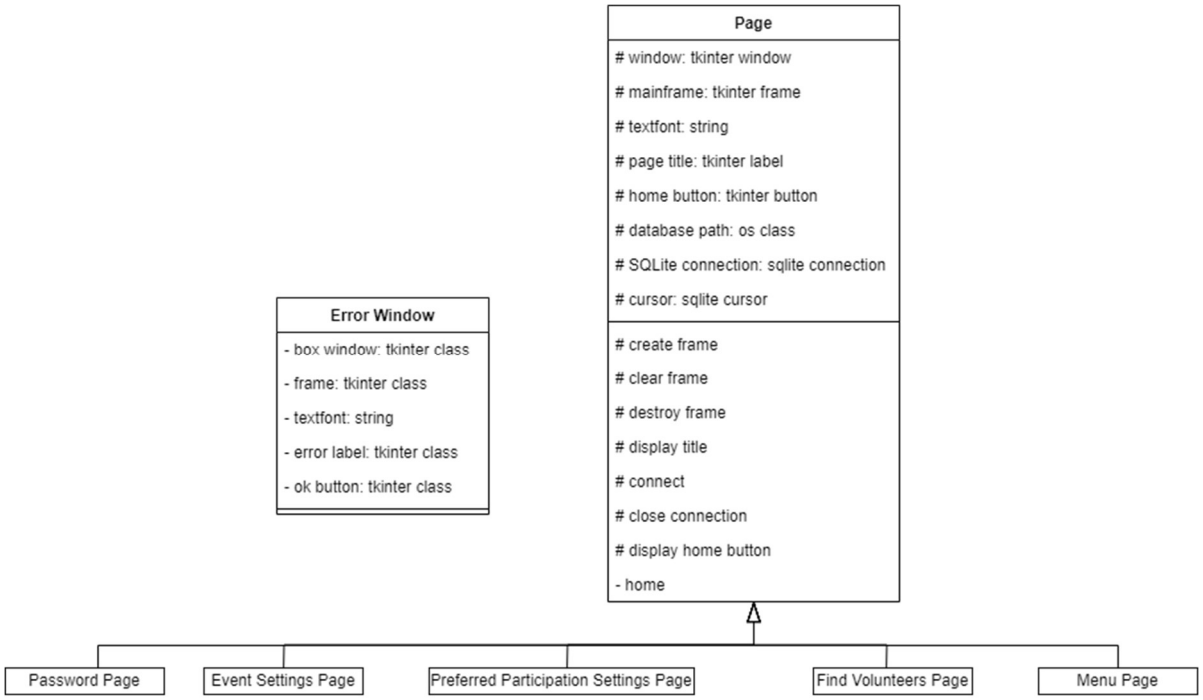


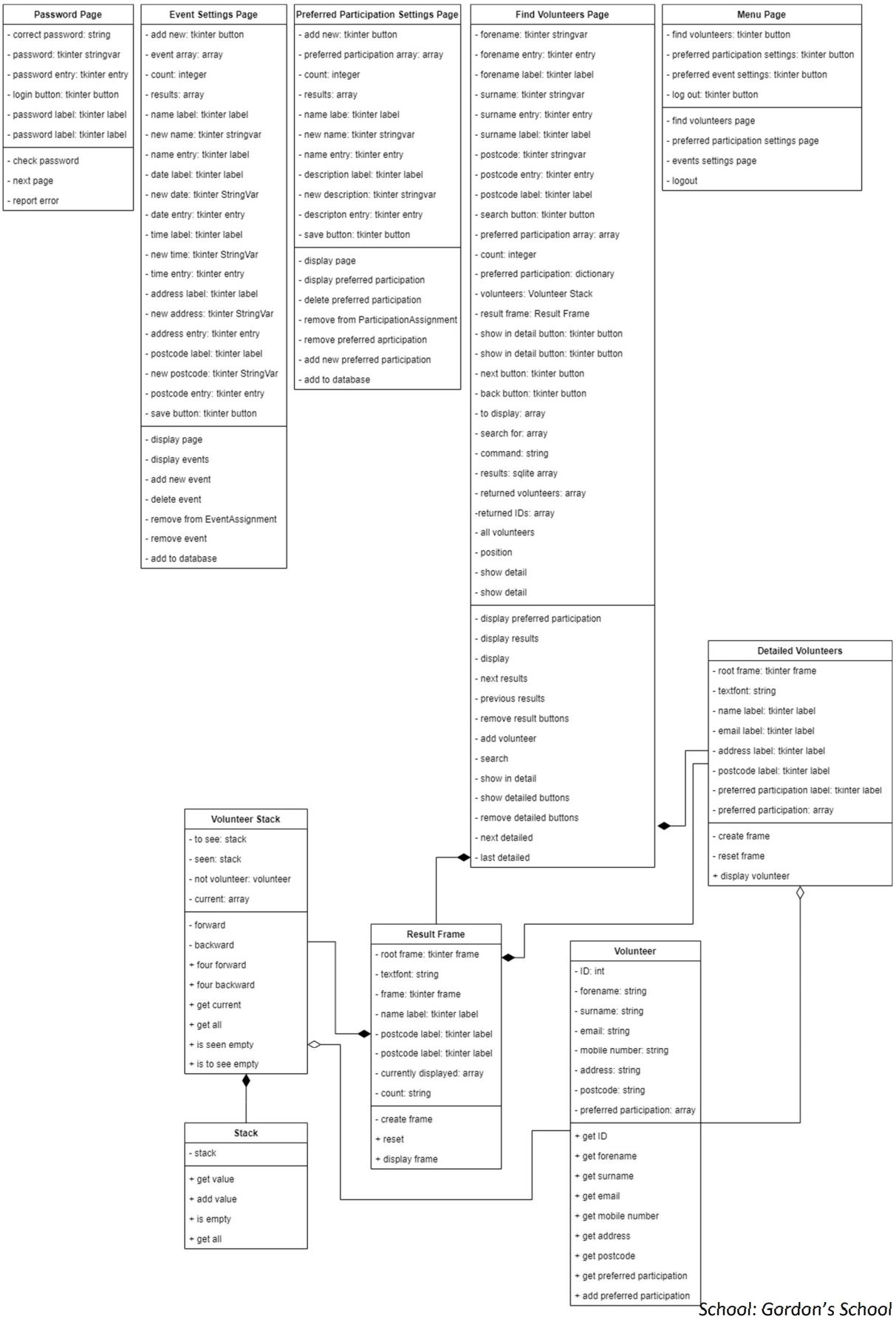
ADMIN PROGRAM

The admin program will be built using python. The libraries used to build this are:

- Tkinter – GUI toolkit
- Psycopg2 – PSQL interaction toolkit

CLASS DIAGRAM




Password Page
- correct password: string
- password: tkinter stringvar
- password entry: tkinter entry
- login button: tkinter button
- password label: tkinter label
- password label: tkinter label
- check password
- next page
- report error
Event Settings Page
- add new: tkinter button
- event array: array
- count: integer
- results: array
- name label: tkinter label
- new name: tkinter stringvar
- name entry: tkinter label
- date label: tkinter label
- new date: tkinter StringVar
- date entry: tkinter entry
- time label: tkinter label
- new time: tkinter StringVar
- time entry: tkinter entry
- address label: tkinter label
- new address: tkinter StringVar
- address entry: tkinter entry
- postcode label: tkinter label
- new postcode: tkinter StringVar
- postcode entry: tkinter entry
- save button: tkinter button
- display page
- display events
- add new event
- delete event
- remove from EventAssignment
- remove event
- add to database
Preferred Participation Settings Page
- add new: tkinter button
- preferred participation array: array
- count: integer
- results: array
- name labe: tkinter label
- new name: tkinter stringvar
- name entry: tkinter entry
- description label: tkinter label
- new description: tkinter stringvar
- descripton entry: tkinter entry
- save button: tkinter button
- display page
- display preferred participation
- delete preferred participation
- remove from ParticipationAssignment
- remove preferred aprticipation
- add new preferred participation
- add to database
Find Volunteers Page
- forename: tkinter stringvar
- forename entry: tkinter entry
- forename label: tkinter label
- surname: tkinter stringvar
- surname entry: tkinter entry
- surname label: tkinter label
- postcode: tkinter stringvar
- postcode entry: tkinter entry
- postcode label: tkinter label
- search button: tkinter button
- preferred participation array: array
- count: integer
- preferred participation: dictionary
- volunteers: Volunteer Stack
- result frame: Result Frame
- show in detail button: tkinter button
- show in detail button: tkinter button
- next button: tkinter button
- back button: tkinter button
- to display: array
- search for: array
- command: string
- results: sqlite array
- returned volunteers: array
-returned IDs: array
- all volunteers
- position
- show detail
- show detail
- display preferred participation
- display results
- display
- next results
- previous results
- remove result buttons
- add volunteer
- search
- show in detail
- show detailed buttons
- remove detailed buttons
- next detailed
- last detailed
Menu Page
- find volunteers: tkinter button
- preferred participation settings: tkinter button
- preferred event settings: tkinter button
- log out: tkinter button
- find volunteers page
- preferred participation settings page
- events settings page
- logout
Detailed Volunteers
- root frame: tkinter frame
- textfont: string
- name label: tkinter label
- email label: tkinter label
- address label: tkinter label
- postcode label: tkinter label
- preferred participation label: tkinter label
- preferred participation: array
- create frame
- reset frame
+ display volunteer
Volunteer Stack
- to see: stack
- seen: stack
- not volunteer: volunteer
- current: array
- forward
- backward
+ four forward
+ four backward
+ get current
+ get all
+ is seen empty
+ is to see empty
Result Frame
- root frame: tkinter frame
- textfont: string
- frame: tkinter frame
- name label: tkinter label
- postcode label: tkinter label
- postcode label: tkinter label
- currently displayed: array
- count: string
- create frame
+ reset
+ display frame
Volunteer
- ID: int
- forename: string
- surname: string
- email: string
- mobile number: string
- address: string
- postcode: string
- preferred participation: array
+ get ID
+ get forename
+ get surname
+ get email
+ get mobile number
+ get address
+ get postcode
+ get preferred participation
+ add preferred participation
Stack
- stack
+ get value
+ add value
+ is empty
+ get all

PASSWORD PAGE

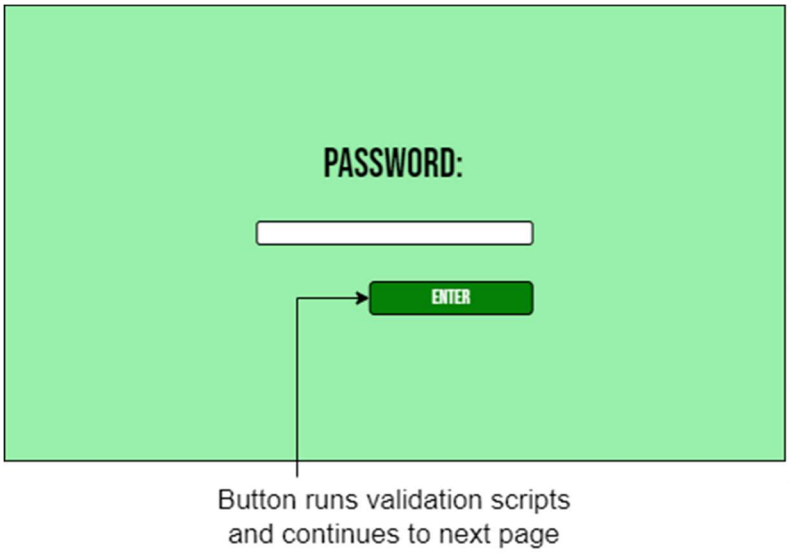
This will be the front page of the admin program where the user must enter a password to access the database.

IPSO CHART

INPUTS	PROCESSES	STORAGE	OUTPUT
Password	Ensure password is correct.	Retrieve password. Create login session	Next page

GUI DESIGN

DATA ITEM	DATA TYPE	VALIDATION/RESTRICTIONS
Password	String	-Same as externally stored and protected password



The password for this page will be hardcoded into the program.

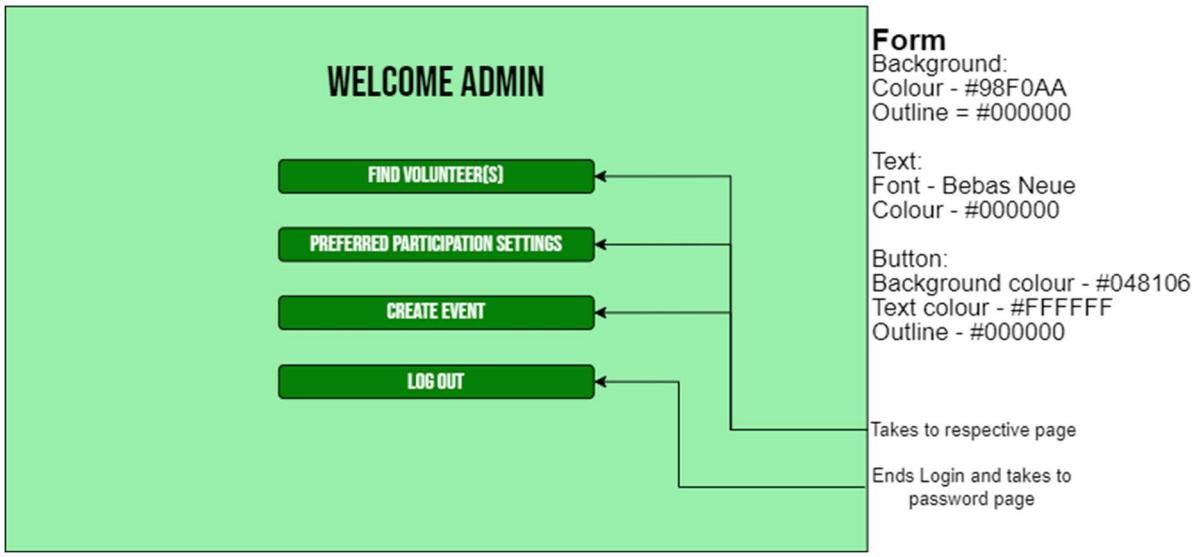
MENU PAGE

This page will contain the way to access the other pages within

IPSO CHART

INPUTS	PROCESSES	STORAGE	OUTPUT
Choice of page Log out	Which page has been clicked	End login session (if log out)	Next page

GUI DESIGN



FIND VOLUNTEER(S) PAGE

This page will allow the admin to search for specific members or groups of members and supply the option to email all members returned.

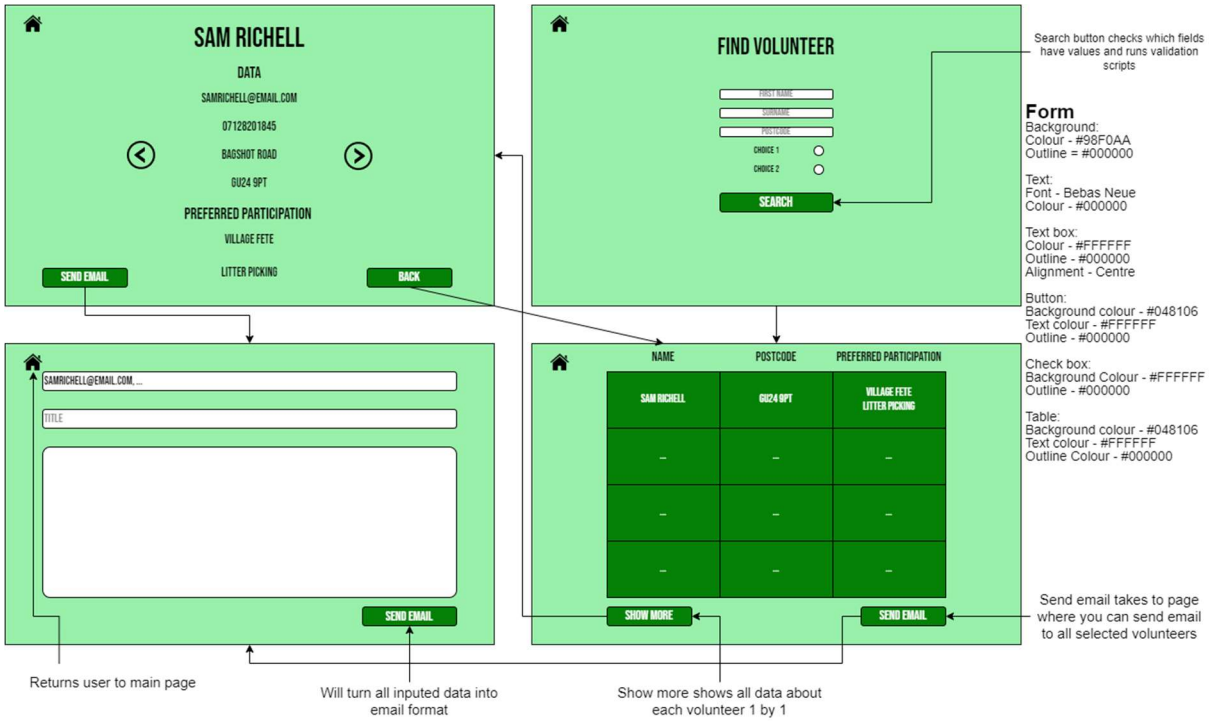
IPSO CHART

INPUTS	PROCESSES	STORAGE	OUTPUT
Inputs for all data types Search button Send email button. Go back a page	Which fields have inputs. Create SQL query. Connect with database. Query database. Arrange returned data.		List of returned values If no values are returned

GUI DESIGN

DATA ITEM	DATA TYPE	VALIDATION/RESTRICTIONS
FIND VOLUNTEER		
First Name	String	-Only letters
Surname	String	-Letters and spaces only
Postcode	String	-Outward code and inward code
Preferred Participation	Boolean	-Checkboxes
SEND EMAIL		
Email	String	-Follow common format
Title	String	-Required -<100 characters

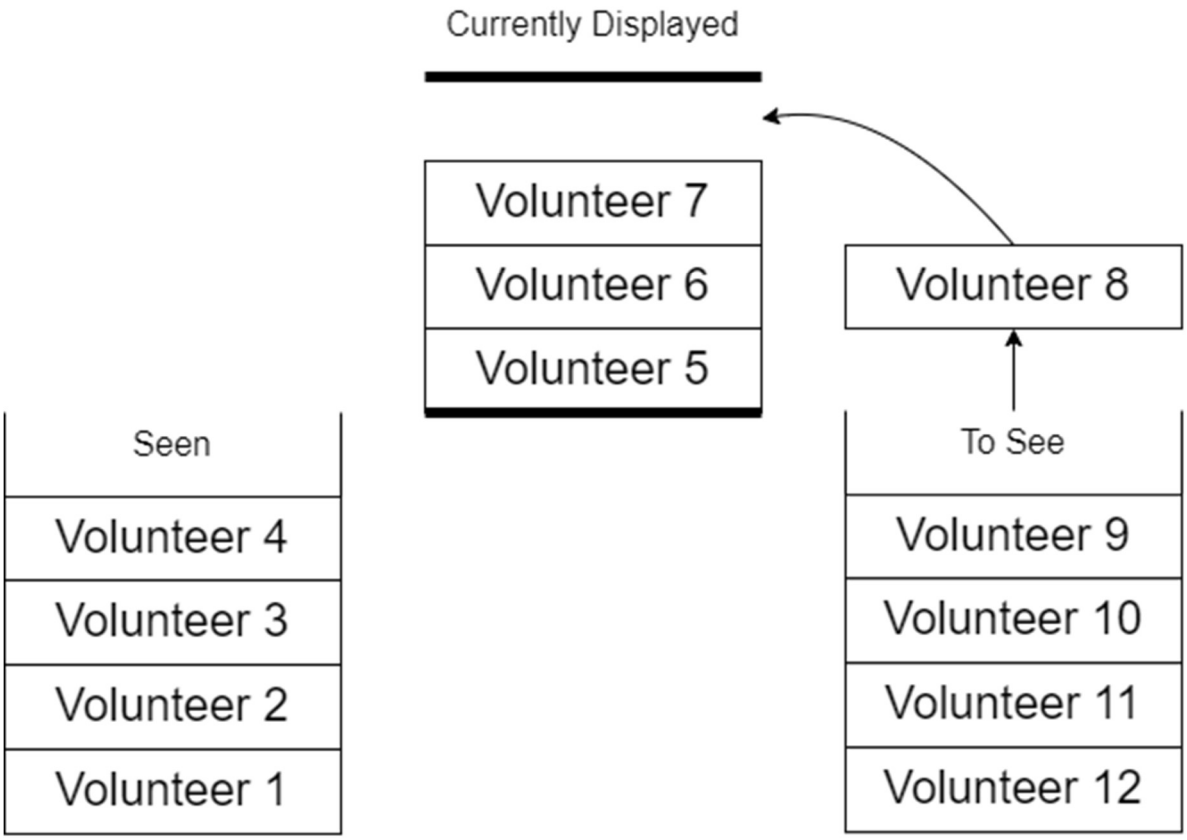
Message	String	-Required



DISPLAY FRAME

STACKS

This is a visual of how the display frame will work.



It will be two stacks (seen and to see). Each returned volunteer will start in too see and when displayed will be kept in a separate array called current. When the next 4 volunteers are to be displayed, The current 4 will be appended to the seen stack, however before being appended, they must be reversed before being added to the stack

PSEUDOCODE

```
Class Stack
	Initialisation
		Private Stack = Array
		Private Length = 0
	Public Pop
		ToReturn = Stack[Length]
		Length = Length – 1
		Return ToReturn
	Public Add (value)
		Append Stack (value)
		Length = Length + 1

Class Volunteer Stack
	Initialisation
		Private Seen = Stack
		Private To See = Stack
		Private Current = Array
	Public GetNext
		For value in reverse(Current)
			Seen Add (value)
		EndFor
		For i in range 4
			Append Current[i] (To See)
		EndFor
	Public GetLast
		For value in reverse(Current)
			To See Add (value)
		EndFor
		For i in range 4
			Append Current[i] (Seen)
		EndFor
```

PAGE SPECIFIC PSEUDOCODE

VOLUNTEER CLASS

```
Class Volunteer
	Initiation (Forename, Surname, Email, MobileNumber, Address, Postcode)
		Private Forename = Forename
		Private Surname = Surname
		Private Email = Email
		Private MobileNumber = MobileNumber
		Private Address = Address
		Private Postcode = Postcode
		Private Preferred Participation = Array
	Public AddPreferredParticipation (value)
		Append Preferred Participation (value)
	Public GetForename
		Return Forename
	Public GetSurname
		Return Surname
	Public GetEmail
		Return Email
	Public GetMobileNumber
		Return MobileNumber
	Public GetAddress
		Return Address
	Public GetPostcode
		Return Postcode
	Public GetPreferredParticipation
		Return Preferred Participation
```

SHOW PREFERRED PARTICIPATION OPTIONS

This pseudocode will execute a query which returns all the types of preferred participation and then adds a check button in the tkinter window which allows the admin to search for any type of preferred participation.

```
query = "SELECT ParticipationName
FROM PreferredParticipation;"

PP_array = []
Count = 0
for row in cursor.execute(query):

	PP_label = ttk.Label(mainframe, text=row[0], font=(textfont, 14))
	PP_label.grid(column=0, row=count, columnspan=2, pady=15)

	PP_choice = IntVar()
	PP_checkbutton = ttk.Checkbutton(mainframe, variable=PP_choice)
	PP_checkbutton.grid(column=2, row=count)

	PP = {"label": PP_label,
	      "text": row[0],
	      "variable": PP_choice,
	      "button": PP_checkbutton}

	PP_array.append(PP)

	Count += 1
```

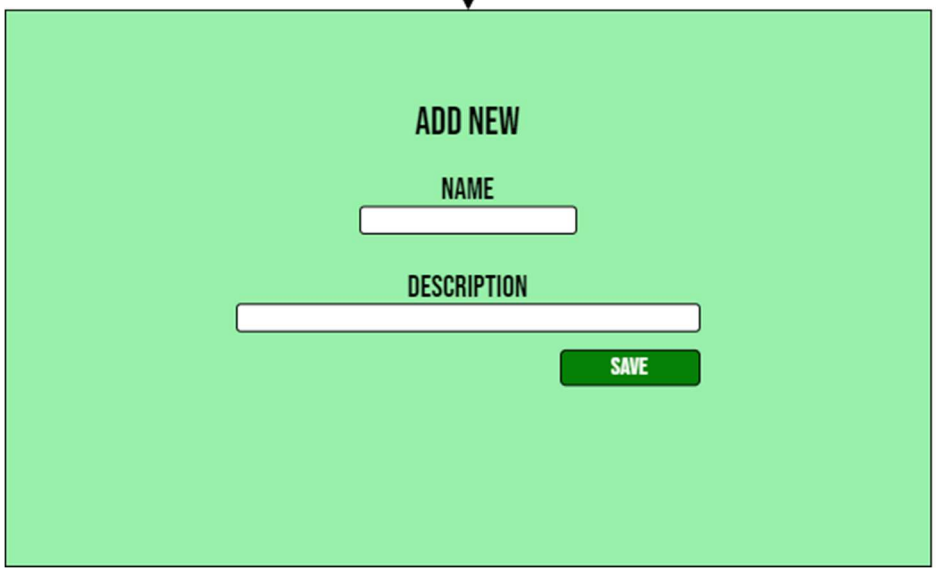
The pseudocode will save the values of each button as part of a dictionary and then append it to an array, allowing it all to be accessed simply for later use. This code can be somewhat reused for the preferred participation setting page.

PREFERRED PARTICIPATION SETTINGS PAGE

IPSO CHART

INPUTS	PROCESSES	STORAGE	OUTPUTS
Add new preferred participation categories. Remove preferred participation categories.	Retrieve current preferred participation categories. Create new column in DB.		Current Preferred Participation. Add new button.

GUI DESIGN



Form
Background:
Colour - #98F0AA

Text:
Font - Bebas Neue
Colour - #000000

Button:
Background colour - #048106
Text colour - #FFFFFF

PSEUDOCODE

DELETING PREFERRED PARTICIPATION CHOICE

Before deleting the preferred participation from the database, any link to the specific preferred participation must first be deleted to ensure that no foreign key constraints are violated.

```
DELETE FROM ParticipationAssignment WHERE ParticipationID={selected participation ID}
```

Now the preferred participation choice can be deleted from the PreferredParticipation table

```
DELETE FROM PreferredParticipation WHERE ParticipationID={selected participation ID}
```

ADDING PREFERRED PARTICIPATION CHOICE

First the new ID must be selected. My idea of good practice will be finding the first empty slot of a preferred participation (if preferred participation with ID 4 had recently been deleted then adding a new one will take its place) This code iterates through all current participation IDs and identifies the gap when the last checked ID is not

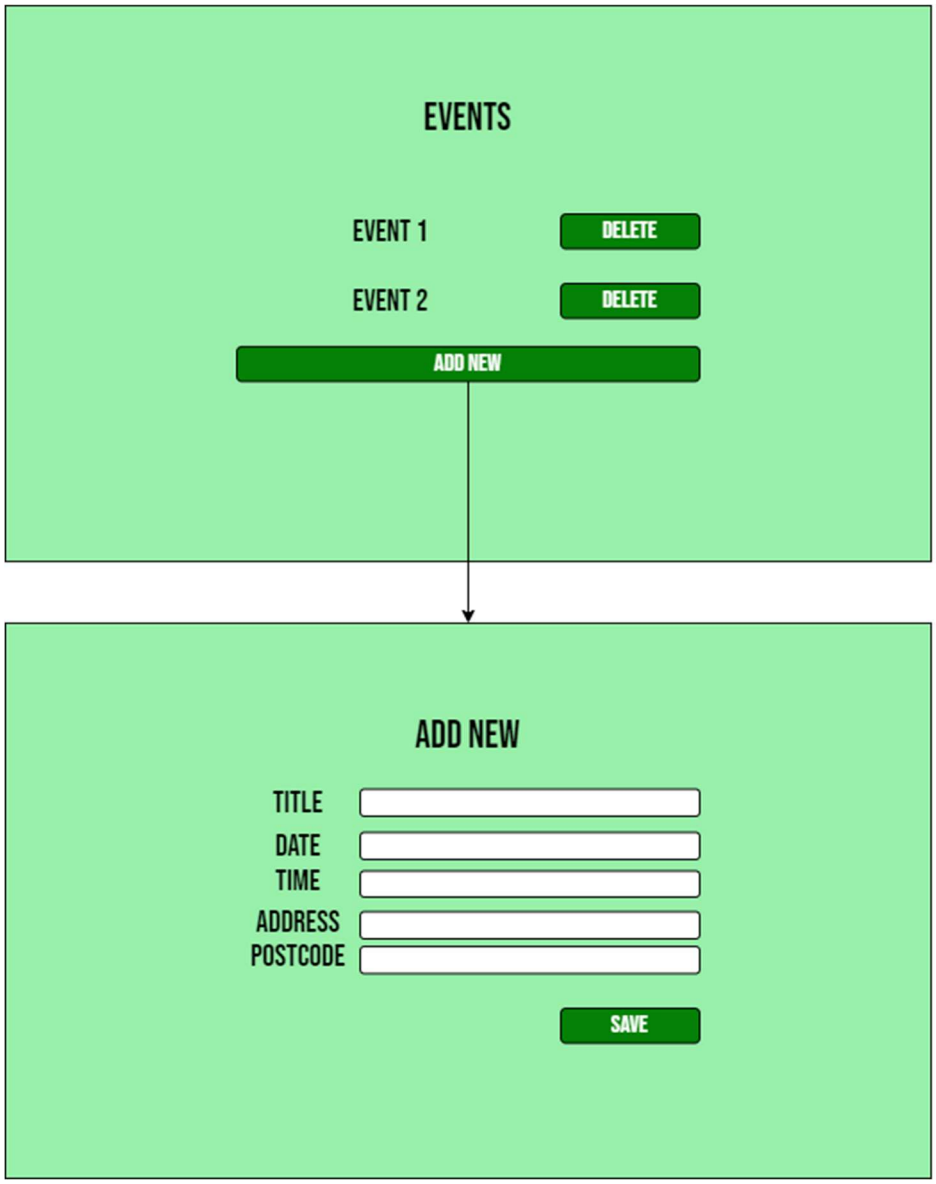
```
SELECT ParticipationID FROM PreferredParticipation
lastID = 0
found = False
FOR ID in results
	IF ID != lastID + 1 AND found = False
		newID = lastID + 1
		found = True
	ENDIF
	lastID = ID
ENDFOR
IF found == False
	newID = lastID + 1
INSERT INTO PreferredParticipation (ParticipationID, ParticipationName, ParticipationDescription) VALUES ({newID}, {entered name}, {entered description});
```

EVENT SETTINGS PAGE

IPSO CHART

INPUTS	PROCESSES	STORAGE	OUTPUTS
Add new events. Remove events.	Retrieve current events. Create new event in DB		Current events. Add new button.

GUI DESIGN



Form
Background:
Colour - #98F0AA

Text:
Font - Bebas Neue
Colour - #000000

Button:
Background colour - #048106
Text colour - #FFFFFF

PSEUDOCODE

DELETING EVENT CHOICE

Before deleting the preferred participation from the database, any link to the specific preferred participation must first be deleted to ensure that no foreign key constraints are violated.

```
DELETE FROM EventAssignment WHERE EventID={selected participation ID}
```

Now the preferred participation choice can be deleted from the PreferredParticipation table

```
DELETE FROM Events WHERE EventID={selected participation ID}
```

ADDING EVENT CHOICE

First the new ID must be selected. My idea of good practice will be finding the first empty slot of a preferred participation (if preferred participation with ID 4 had recently been deleted then adding a new one will take its place) This code iterates through all current participation IDs and identifies the gap when the last checked ID is not

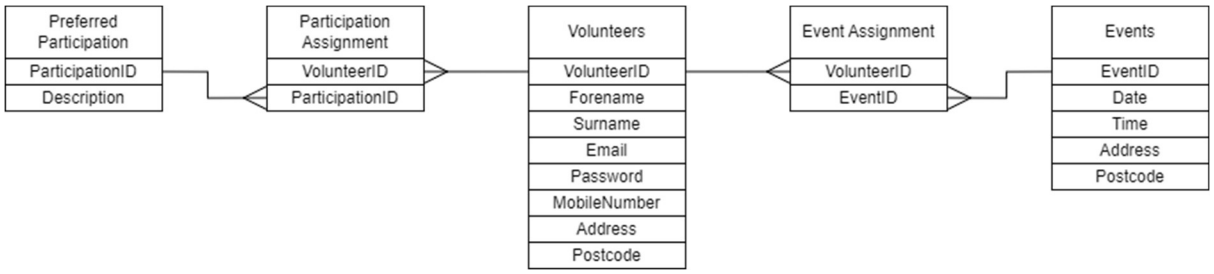
```
SELECT EventID FROM Events
lastID = 0
found = False
FOR ID in results
	IF ID != lastID + 1 AND found = False
		newID = lastID + 1
		found = True
	ENDIF
	lastID = ID
ENDFOR
IF found == False
	newID = lastID + 1
INSERT INTO Events (EventID, Name, Date, Time, Address, Postcode) VALUES ({newID}, {entered name}, {entered date}, {entered time}, {entered address}, {entered postcode});
```

BACKEND

The backend of this system will be an SQL file that can be accessed by the admin program and data written to when new users sign up.

SQL FILE

ERD DIAGRAM



DDL SCRIPT

This is all the code needed to create the SQL Tables that will be required to store the data about the volunteers based on the ERD diagram above.

```
CREATE TABLE Volunteers(
  VolunteerID INTEGER NOT NULL,
  Forename VARCHAR(30) NOT NULL,
  Surname VARCHAR(30) NOT NULL,
  Email VARCHAR(150) NOT NULL,
  Password VARCHAR(100) NOT NULL,
  MobileNumber INTEGER NOT NULL,
  Address VARCHAR(150) NOT NULL,
  Postcode VARCHAR(7) NOT NULL,
  PRIMARY KEY (VolunteerID));

CREATE TABLE EventAssignment(
  VolunteerID INTEGER NOT NULL,
  EventID INTEGER NOT NULL,
  FOREIGN KEY (VolunteerID) REFERENCES Volunteers(VolunteerID)
  FOREIGN KEY (EventID) REFERENCES Events(EventID)
  PRIMARY KEY (VolunteerID, EventID));

CREATE TABLE Events(
  EventID INTEGER NOT NULL,
  EventName VARCHAR(50) NOT NULL,
  Date DATE NOT NULL,
  Time TIME NOT NULL,
  Address VARCHAR(150) NOT NULL,
  Postcode VARCHAR(7) NOT NULL,
  PRIMARY KEY (EventID));

CREATE TABLE ParticipationAssignment(
  VolunteerID INTEGER NOT NULL,
  ParticipationID INTEGER NOT NULL,
  FOREIGN KEY (VolunteerID) REFERENCES Volunteers(VolunteerID)
  FOREIGN KEY (ParticipationID) REFERENCES PreferredParticipation(ParticipationID)
  PRIMARY KEY (VolunteerID, ParticipationID));

CREATE TABLE PreferredParticipation(
  ParticipationID INTEGER NOT NULL,
  Description VARCHAR(300) NOT NULL,
  PRIMARY KEY (ParticipationID));
```

TECHNICAL SOLUTION

DATABASE INITIALISATION

Function: CreateTables (Appendix 9 – Line 7-63)

This code has one purpose: to create the database with its various tables and elements. This is achieved by first creating a connection to the database, running an execute with the cursor and at the end of the program all execute statements are committed.

```
def CreateTables(cursor):

    VolunteersCreation = """
CREATE TABLE IF NOT EXISTS Volunteers(
VolunteerID INTEGER NOT NULL,
Forename VARCHAR(30) NOT NULL,
Surname VARCHAR(30) NOT NULL,
Email VARCHAR(150) NOT NULL,
Password VARCHAR(100) NOT NULL,
MobileNumber INTEGER NOT NULL,
Address VARCHAR(150) NOT NULL,
Postcode VARCHAR(7) NOT NULL,
PRIMARY KEY (VolunteerID));
"""

    EventsCreation = """
CREATE TABLE IF NOT EXISTS Events(
EventID INTEGER NOT NULL,
EventName VARCHAR(50) NOT NULL,
Date DATE NOT NULL,
Time TIME NOT NULL,
Address VARCHAR(150) NOT NULL,
Postcode VARCHAR(7) NOT NULL,
PRIMARY KEY (EventID));
"""

    PrefferedParticipationCreation = """
CREATE TABLE IF NOT EXISTS PreferredParticipation(
ParticipationID INTEGER NOT NULL,
ParticipationName VARCHAR(150) NOT NULL,
ParticipationDescription VARCHAR(300),
PRIMARY KEY (ParticipationID));
"""

    EventAssignmentCreation = """
CREATE TABLE IF NOT EXISTS EventAssignment(
VolunteerID INTEGER NOT NULL,
EventID INTEGER NOT NULL,
```

```
FOREIGN KEY (VolunteerID) REFERENCES Volunteers(VolunteerID)
FOREIGN KEY (EventID) REFERENCES Events(EventID)
PRIMARY KEY (VolunteerID, EventID));
"""

    ParticipationAssignment = """
CREATE TABLE IF NOT EXISTS ParticipationAssignment(
VolunteerID INTEGER NOT NULL,
ParticipationID INTEGER NOT NULL,
FOREIGN KEY (VolunteerID) REFERENCES Volunteers(VolunteerID)
FOREIGN KEY (ParticipationID) REFERENCES
PreferredParticipation(ParticipationID)
PRIMARY KEY (VolunteerID, ParticipationID));
"""

    cursor.execute(VolunteersCreation) #Executing every command to create
all the tables
    cursor.execute(EventsCreation)
    cursor.execute(PrefferedParticipationCreation)
    cursor.execute(EventAssignmentCreation)
    cursor.execute(ParticipationAssignment)
```

ADMIN PROGRAM.PY

Class: Volunteer (Appendix 1 – Line 34-70)

This class will store all the required information from a volunteer returned by the database from a query. As each volunteer can have multiple preferred participation, it will be stored as an array. Each of the values of the volunteer are stored as private and can only be accessed by the defined getters.

```
class Volunteer:
    def __init__(self, ID, forename, surname, email, mobilenumber, address,
postcode):
        self.__ID = ID
        self.__forename = forename
        self.__surname =surname
        self.__email = email
        self.__mobilenumber = mobilenumber
        self.__address = address
        self.__postcode = postcode
        self.__PreferredParticipation = []

    def add_PP(self, PP):
        self.__PreferredParticipation.append(PP)

    def get_ID(self):
        return self.__ID

    def get_forename(self):
        return self.__forename

    def get_surname(self):
        return self.__surname

    def get_email(self):
        return self.__email

    def get_mobilenumber(self):
        return self.__mobilenumber

    def get_address(self):
        return self.__address

    def get_postcode(self):
        return self.__postcode

    def get_PP(self):
        return self.__PreferredParticipation
```

Class: Stack (Appendix 1 – Line 519-541)

This class is very simply to act as a stack. The original state of the stack is passed in as an array and the rear pointer is used to return items at the back of the list. As pop is already a pre-defined function in python, I instead had to name the pop function "get value".

```
class Stack: # Stack allowing easy viewing of
    def __init__(self, array):
        self.__stack = array
        self.__rear_pointer = len(self.__stack)-1

    def get_value(self):
        to_return = self.__stack.pop(self.__rear_pointer)
        self.__rear_pointer -= 1
        return to_return

    def add_value(self, value):
        self.__stack.append(value)
        self.__rear_pointer += 1

    def is_empty(self):
        if len(self.__stack) == 0:
            empty = True
        else:
            empty = False
        return empty

    def get_all(self):
        return self.__stack
```

Class: Volunteer Stack (Appendix 1 – Line 594-597)

This class utilises the stack class as two separate stacks to store the "to see" and "seen" lists. As at each time only 4 volunteers are displayed, the program will only need to move 4 volunteers from stack to stack at a time. To fill up all 4 slots, the "not volunteer" is defined with a dash as it's postcode so that the result frame will show a dash.

```
class Volunteer_Stack:
    def __init__(self, volunteers):
        self.__to_see = Stack(volunteers)
        self.__seen = Stack([]) # No volunteers have been seen yet so stack
starts empty
        self.__not_volunteer = Volunteer(0, "", "", "", "", "", "-")
        self.__not_volunteer.add_PP("")
        self.__current = []

    def __forward(self):
        try:
```

```
            value = self.__to_see.get_value()
        except: # This will fill in any empty slots as a dash
            value = self.__not_volunteer
        self.__current.append(value)

    def __backward(self):
        value = self.__seen.get_value()
        self.__current.append(value)

    def four_forward(self):
        for value in self.__current[::-1]:    # Reversing the appended
current array means the volunteers will be added back to the stack in the
correct order
            self.__seen.add_value(value)
        self.__current = []
        for i in range(4):
            self.__forward()

    def four_backward(self):
        for value in self.__current[::-1]:   # Reversing the appended
current array means the volunteers will be added back to the stack in the
correct order
            self.__to_see.add_value(value)
        self.__current = []
        for i in range (4):
            self.__backward()

    def get_current(self):
        return self.__current

    def get_all(self):
        seen = self.__seen.get_all()
        to_see = self.__to_see.get_all()
        position = len(seen)
        volunteer_array = []
        for volunteer in seen[::-1]:
            volunteer_array.append(volunteer)
        for volunteer in self.__current:
            volunteer_array.append(volunteer)
        for volunteer in to_see[::-1]:
            volunteer_array.append(volunteer)
        return volunteer_array, position

    def is_seen_empty(self):
        return self.__seen.is_empty()
```

```
    def is_to_see_empty(self):
        return self.__to_see.is_empty()
```

Class: Result Frame (Appendix 1 – Line 462-516)

This class is used to display the volunteers returned from the volunteer stack no matter how many volunteers are passed in (in case it changes from 4).

```
class Result_Frame: # Result frame to display results from volunteer search
    def __init__(self, root_frame, textfont):

        self.__root_frame = root_frame
        self.__textfont = textfont

    def __create_frame(self, root_frame):
        self.__frame = create_frame(root_frame)
        self.__frame.grid(row=0, column=1)

    def reset(self):
        for widget in self.__frame.winfo_children():
            widget.destroy()
        self.__frame.destroy()

    def display_frame(self, volunteers):

        self.__create_frame(self.__root_frame)

        self.__currently_displayed = []

        self.__name_label = ttk.Label(self.__frame, text="Name",
font=(self.__textfont, 12,  "bold"))
        self.__name_label.grid(column=0, row=0, pady=40, padx=40)

        self.__postcode_label = ttk.Label(self.__frame, text="Postcode",
font=(self.__textfont, 12, "bold"))
        self.__postcode_label.grid(column=1, row=0, pady=40, padx=40)

        self.__PP_label = ttk.Label(self.__frame, text="Preferred
Participation", font=(self.__textfont, 12, "bold"))
        self.__PP_label.grid(column=2, row=0, pady=40, padx=20)

        self.__count = 1
        for value in volunteers:
```

```
            name_label = ttk.Label(self.__frame,
text=f"{value.get_forename()} {value.get_surname()}", font=self.__textfont)
            name_label.grid(column=0, row=self.__count, pady=40)

            postcode_label = ttk.Label(self.__frame,
text=value.get_postcode(), font=self.__textfont)
            postcode_label.grid(column=1, row=self.__count, pady=40)

            PP_string = ""

            for PP in value.get_PP():
                PP_string += f"{PP}\n"

            PP_string = PP_string[0:-1]   # Removes the final \n from the
string

            PP_label = ttk.Label(self.__frame, text=PP_string,
font=self.__textfont)
            PP_label.grid(column=2, row=self.__count, sticky=W)

            volunteer = {"Name Label": name_label,
                         "Postcode Label": postcode_label,
                         "PP Label": PP_label}

            self.__currently_displayed.append(volunteer)
            self.__count += 1
```

Function: Display Results (Appendix 1 – Line 305-314 – Taken from find volunteers page class)

This function defines the volunteer stack and result frame initially and initialises the results section of the search for volunteer page.

```
def __display_results(self):

        self.__volunteers = Volunteer_Stack(self.__returned_volunteers)

        self.__result_frame = Result_Frame(self._mainframe, self._textfont)

        self.__next_results()

        self.__show_in_detail_button = ttk.Button(self._mainframe,
text="Show More Detail", command=self.__show_in_detail)
        self.__show_in_detail_button.grid(column=1, row=1)
```

Function: Next Results (Appendix 1 – Line 333-337 – Taken from find volunteers page class)

This function moves the volunteer stack forward and returns the next 4 volunteers before calling another function to display them.

```
def __next_results(self):
        self.__volunteers.four_forward()
        self.__to_display = self.__volunteers.get_current()
        self.__remove_result_buttons()
        self.__display()
```

Function: Previous Results (Appendix 1 – Line 339-343 – Taken from find volunteers page class)

This function moves the volunteer stack backward and returns the last 4 volunteers before calling another function to display them.

```
def __previous_results(self):
        self.__volunteers.four_backward()
        self.__to_display = self.__volunteers.get_current()
        self.__remove_result_buttons()
        self.__display()
```

Function: Display (Appendix 1 – Line 316-331 – Taken from find volunteers page class)

This Function will reset the frame and use the returned volunteers to display them using the result frame defined before

```
def __display(self):

        try:
            self.__result_frame.reset()
        except:
            pass

        self.__result_frame.display_frame(self.__to_display)

        if self.__volunteers.is_to_see_empty() != True: # Only displays
button if user can go forward
            self.__next_button = ttk.Button(self._mainframe, text="Next",
command=self.__next_results)
            self.__next_button.grid(column=2, row=0)

        if self.__volunteers.is_seen_empty() != True: # Only displays button
if user can go backwards
            self.__back_button = ttk.Button(self._mainframe, text="Back",
command=self.__previous_results)
            self.__back_button.grid(column=0, row=0)
```

Class: Page (Appendix 1 – Line 122-160)

This class acts as a parent class to all the page classes and provides all the information that each class needs as a protected variable or class. One notable function in this class is the home function. This function is private as daughter classes should not be able to access it, it should only be called when the home button is pressed.

```
class Page: # Parent class to all other pages
    def __init__(self, window, textfont, title):
        self._textfont = textfont
        self._window = window
        self._window.title(title)

    def _create_frame(self):
        self._mainframe = create_frame(self._window)
        self._mainframe.pack(padx=250, pady=100)

    def _display_title(self, page_title):
        self._page_title = ttk.Label(self._mainframe, text=page_title,
font=(self._textfont, 30, "bold"))

    def _clear_frame(self):
        for widget in self._mainframe.winfo_children():
            widget.destroy()

    def _destroy_frame(self):
        self._mainframe.destroy()

    def _connect(self):
        self._DBPath = os.path.join(os.path.dirname(__file__),
"VolunteerDB.sqlite3") #Defining the pathway to the sqlite3 file

        self._SQLiteConnection = sqlite3.connect(self._DBPath) #Connecting
to the sql file
        self._cursor = self._SQLiteConnection.cursor() #creating the cursor

    def _close_connection(self):
        self._cursor.close()
        self._SQLiteConnection.close()

    def _display_home_button(self):
        self.__home_button = ttk.Button(self._window, text="Home",
command=self.__home)
        self.__home_button.place(x=5, y=5)

    def __home(self): # Subclasses should not be able to access
        self.__home_button.destroy()
```

```
        self._clear_frame()
        self._destroy_frame()
        menu_page(self._window, self._textfont)
```

Function: Display PP (Appendix 1 – Line 275-303 – Taken from find volunteers page class)

This function is taken from the find volunteers page class and it finds all the current preferred participation options from the database, iterates through them and displays them on the page next to a delete button. Each row of preferred participation and delete button is formed as a dictionary which is then appended to a list in case any further updates require altering these buttons.

```
def __display_PP(self):

        self._connect()

        query = """SELECT ParticipationName
        FROM PreferredParticipation;"""

        self.__PP_array = []
        self.__count = 1

        self._cursor.execute(query)
        self.__results = self._cursor.fetchall()

        for row in self.__results:

            PP_label = ttk.Label(self._mainframe, text=row[0],
font=(self._textfont, 14))
            PP_label.grid(column=0, row=self.__count, columnspan=2, pady=15)

            PP_delete_button = ttk.Button(self._mainframe, text="Delete",
command=lambda to_delete=row[0]: self.__delete_PP(to_delete))
            PP_delete_button.grid(column=2, row=self.__count)

            PP = {"label": PP_label,
                  "text": row[0],
                  "button": PP_delete_button}     # All the values for each
of the Preferred Participation buttons are stored as a dictionary

            self.__PP_array.append(PP)

            self.__count += 1

        self._close_connection()
```

Function: Search (Appendix 1 – Line 368-416 – Taken from find volunteers page class)

This function is taken from the find volunteers page class and it forms the search query based on the inputted name and/or preferred participation choices are. A dictionary has been used to store what its searching for and its value. This dictionary is then appended to a list so it can be iterated through to form the search query. To form the query, a string of "['type']=['value']" is added onto the preformed start of the query and then an extra line is added to link all the tables together. The returned list is iterated through by each row and the add volunteer function is used.

```
def __search(self):
        self._connect()

        self.__search_for = []   # List for all that

        if self.__forename.get() != "":
            self.__search_for.append({"type": "Forename", "value":
self.__forename.get(), "PP": False})

        if self.__surname.get() != "":
            self.__search_for.append({"type": "Surname", "value":
self.__surname.get(), "PP": False})

        if self.__postcode.get() != "":
            self.__search_for.append({"type": "Postcode", "value":
self.__postcode.get(), "PP": False})

        for PP in self.__PP_array:
            if PP["variable"].get() == 1:
                self.__search_for.append({"type": PP["text"], "value":
PP["variable"].get(), "PP": True})

        self.__command = """SELECT Volunteers.VolunteerID,
Volunteers.Forename, Volunteers.Surname, Volunteers.Email,
Volunteers.MobileNumber, Volunteers.Address, Volunteers.Postcode,
PreferredParticipation.ParticipationName
        FROM Volunteers, PreferredParticipation, ParticipationAssignment
        WHERE""" # start to the SQL query to find volunteers

        for PP in self.__search_for:
            if PP["PP"] == True:
                self.__command += f"
PreferredParticipation.ParticipationName = '{PP['type']}' AND"
            else:
                self.__command += f" Volunteers.{PP['type']} =
'{PP['value']}' AND"
```

```
        self.__command += f" Volunteers.VolunteerID =
ParticipationAssignment.VolunteerID AND
ParticipationAssignment.ParticipationID =
PreferredParticipation.ParticipationID;"

        self._cursor.execute(self.__command)
        self.__results = self._cursor.fetchall()

        if len(self.__results) == 0:          # Results in an error when no
volunteers are returned
            error_window("No Volunteers Found", "Arial")
        else:
            self.__returned_volunteers = []
            self.__returned_IDs = []

            for row in self.__results:
                self.__add_volunteer(row)

            self._close_connection()

            self._clear_frame()

            self.__display_results()
```

Function: Delete PP (Appendix 1 – Line 697-708 – Taken from the preferred participation settings class)

This function will delete the preferred participation option next to the delete button clicked using a select statement to find the participation ID and then calling remove from participation assignment to remove all the data from the link table "ParticipationAssignment" and finally deleting the preferred participation choice from the "PreferredParticipation" Table by calling the Remove PP function.

```
def __delete_PP(self, to_delete):
        self._connect()
        for PP in self.__PP_array:
            if PP["text"] == to_delete:
                query = f"SELECT ParticipationID FROM PreferredParticipation
WHERE ParticipationName = '{PP['text']}'"
                self._cursor.execute(query)
                to_remove_ID, = self._cursor.fetchone()
                self.__remove_from_ParticipationAssignment(to_remove_ID) #
Delete any users connected to PP before deleting PP
                self.__remove_PP(to_remove_ID)
        self._SQLiteConnection.commit()
        self._clear_frame()
        self.__display_page()
```

Function: Remove From ParticipationAssignment (Appendix 1 – Line 710-712 – Taken from the preferred participation settings class)

This function removes all objects from the link table with a specific participation ID.

```
def __remove_from_ParticipationAssignment(self, PP_ID):
        delete_command = f"DELETE FROM ParticipationAssignment WHERE
ParticipationID={PP_ID}"
        self._cursor.execute(delete_command)
```

Function: Remove PP (Appendix 1 – Line 714-716 – Taken from the preferred participation settings class)

This function removes a preferred participation choice with a specific ID from the preferred participation table.

```
def __remove_PP(self, PP_ID):
        remove_command = f"DELETE FROM PreferredParticipation WHERE
ParticipationID={PP_ID}"
        self._cursor.execute(remove_command)
```

Function: Add to DB (Appendix 1 – Line 740-765 – Taken from the preferred participation settings class)

This function adds a new preferred participation choice to the database based on the inputted data into the program.

```
def __add_to_db(self):

        self._connect()

        query = "SELECT ParticipationID FROM PreferredParticipation"
        self._cursor.execute(query)
        results = self._cursor.fetchall()
        last_ID = 0
        found = False
        for row in results:
            to_check, = row  # Splits the returned tuple
            if to_check != last_ID + 1 and found == False:
                new_ID = last_ID + 1
                found = True
            last_ID = to_check
        if found == False:
            new_ID = last_ID + 1

        command = f"INSERT INTO PreferredParticipation (ParticipationID,
ParticipationName, ParticipationDescription) VALUES ({new_ID},
'{self.__new_name.get()}', '{self.__new_description.get()}')"
        self._cursor.execute(command)
        self._SQLiteConnection.commit()
```

```
        self._close_connection()

        self._clear_frame()
        self.__display_page()
```

The functions taken from the preferred participation settings page class have all been reused with minor tweaks such as changing the name of the functions to a more fitting one, so it is not necessary for them to be pasted just to explain the same thing.

UPDATE DETAILS.PHP

UPDATING DETAILS

Updating Details (Appendix 5 – Line 70-111)

This code functions to update the user's details that they have entered into the web form. The code will first remove the user from the database before ensuring that the email and phone number are not already in use. The code will then add the user back to the database with their new details and if the email or phone number are already in use, those will not be updated.

```
// Delete current Details
$delete_PP_statement = $DB_con->prepare("DELETE FROM ParticipationAssignment
WHERE VolunteerID=?");
$delete_PP_statement->execute([$ID]);
$delete_volunteer_statement = $DB_con->prepare("DELETE FROM Volunteers WHERE
VolunteerID=?");
$delete_volunteer_statement->execute([$ID]);

// Check if email or phone number already in use
$check_statement = $DB_con->query("SELECT * FROM Volunteers WHERE
Email='$email' OR MobileNumber='$phone_number'");

$check_results = $check_statement->fetchAll(PDO::FETCH_ASSOC);

// Ensures a result wasn't returned
if (count($check_results) == 0) {

    // Insert statement prepared
    $prepared_statement = $DB_con->prepare("INSERT INTO Volunteers
(VolunteerID, Forename, Surname, Email, Password, MobileNumber, Address,
Postcode) VALUES (?, ?, ?, ?, ?, ?, ?, ?)");

    // Values added into insert statement
    // Insert statement executed returns true if completed and false if
error
    $completed = $prepared_statement->execute([$ID, $forename, $surname,
$email, $password, $phone_number, $address, $postcode]);

    for ($count=0; $count<count($selected_PP); $count++) {
        // Preparing the insert statement for the selected Preferred
Participation
        $PP_prepared_statement = $DB_con->prepare("INSERT INTO
ParticipationAssignment (VolunteerID, ParticipationID) VALUES (?, ?)");
        //Executing the statement
        $PP_prepared_statement->execute([$ID, $selected_PP[$count]]);
    };
```

```
} else {
    // Resetting the email and mobile number
    $email = $user_results[0]["Email"];
    $phone_number = $user_results[0]["MobileNumber"];

    // Insert statement prepared
    $prepared_statement = $DB_con->prepare("INSERT INTO Volunteers
(VolunteerID, Forename, Surname, Email, Password, MobileNumber, Address,
Postcode) VALUES (?, ?, ?, ?, ?, ?, ?, ?)");

    // Values added into insert statement
    // Insert statement executed returns true if completed and false if
error
    $completed = $prepared_statement->execute([$ID, $forename, $surname,
$email, $password, $phone_number, $address, $postcode]);

    for ($count=0; $count<count($selected_PP); $count++) {
        // Preparing the insert statement for the selected Preferred
Participation
        $PP_prepared_statement = $DB_con->prepare("INSERT INTO
ParticipationAssignment (VolunteerID, ParticipationID) VALUES (?, ?)");
        //Executing the statement
        $PP_prepared_statement->execute([$ID, $selected_PP[$count]]);
    };
};
```

TESTING

All passwords in testing will be set as “Password123”

Testing Link: https://tinyurl.com/yt4ys85p

Objective Number	Test Number	Test Description	Test type	Input	Expected Output	Actual Output	Pass/ Fail	Changes	Evidence
1.3.1	1.1.1	Enter name into sign up page	Normal	Sam Richell	added to database	Added to database	Pass		Test 1
1.3.1	1.1.2	Enter numbers into sign up page	Erroneous	@N0T 4N4M3:	Form doesn’t allow submission	Form doesn’t allow submission	Pass		Test 1
1.3.2	1.2.1	Enter email into sign up page	Normal	samrichell17@outlook.com	added to database	Added to database	Pass		Test 1
1.3.2	1.2.2	Enter non email into sign up page	Erroneous	notmyemail	Form doesn’t allow submission	Form doesn’t allow submission	Pass		Test 1
1.3.3	1.3.1	Enter password into sign up page	Normal	Password123	added to database	Added to database	Pass		Test 1
1.3.3	1.3.2	Enter 8 character password into sign up page	Boundary	Thisis8c	Added to database	Added to database	Pass		Test 1
1.3.3	1.3.4	Enter 7 character password into sign up page	Boundary	Thisis7	Form doesn’t allow submission	Form doesn’t allow submission	Pass		Test 1
1.3.3	1.3.5	Enter 100 character	Boundary	12345689 12345689 12345689 12345689 12345689 12345689	Added to database	Added to database	Pass		Test 1

		password into sign up page		12345689 12345689 12345689 12345689 1234568900					
1.3.3	1.3.5	Enter 101 character password into sign up page	Boundary	12345689 12345689 12345689 12345689 12345689 12345689 12345689 12345689 12345689 12345689 12345689000	Form doesn't allow submission	Form doesn't allow submission	Pass		Test 1
1.3.4	1.4.1	Enter mobile number into sign up page	Normal	07123456789	added to database	Added to database	Pass		Test 1
1.3.4	1.4.2	Enter not a phone number into sign up page	Erroneous	This is not a phone number	Form doesn't allow submission	Form doesn't allow submission	Pass		Test 1
1.3.5	1.5.1	Enter full address into sign up page	Normal	Bagshot Road	added to database	Added to database	Pass		Test 1
1.3.5	1.5.2	Enter postcode into sign up page	Normal	GU24 9PT	Added to database	Added to database	Pass		Test 1
1.3.5	1.5.3	Enter invalid postcode into sign up page	Erroneous	GU24 9PT	Form doesn't allow submission	Form doesn't allow submission	Pass		Test 1
1.3.6	1.6.1	Select preferred participation	Normal	2/3 preferred participation selected	added to database	Added to database	Pass		Test 1
1.3.6	1.6.2	Select no preferred participation	Erroneous	0/3 preferred participation selected	Error telling user to select at least 1 preferred	Error telling user to select at least 1 preferred	Pass		Test 1

					participation choice shows	participation choice shows			
1.3.6.1	1.7	Add new preferred participation and see how we page is affected	Normal	New preferred participation added to database	Form shows new preferred participation type	Form shows new preferred participation type	Pass		Test 1
2.1	2.1	Ensure that correct password must be entered.	Normal	Password entered	Main page opens	Main page opens	Pass		Test 2
2.1	2.2	Ensure that correct password must be entered	Erroneous	Wrong password entered	Page shows error	Page shows error	Pass		Test 2
2.3.1	3.1.1	Ensure all volunteers with a certain name are returned	Normal	Sam Richell	12 volunteer names returned	12 volunteer names returned	Pass		Test 3
2.3.1	3.1.2	Check what happens if a name not in the database is returned	Normal	Thisisnot Inthedatabase	Page shows error saying no volunteers returned	Page shows error saying no volunteers returned	Pass		Test 3
2.3	3.2.1	Ensure all volunteers with a specific postcode are returned	Normal	LM51 2FD	Shows 1 volunteer	Shows 1 volunteer	Pass		Test 3
2.3	3.2.2	Check what happens if a specific postcode	Normal	AA11 1AA	Page shows error saying no volunteers returned	Page shows error saying no volunteers returned	Pass		Test 3

		isn't in the database							
2.3	3.3.1	Check what happens if no values are entered	Normal	nothing	All volunteers returned	All volunteers returned	Pass		Test 3
2.3	3.3.2	Ensure only certain volunteers are returned	Normal	Sam Richell, DK09 6SF	One volunteer returned	One volunteer returned	Pass		Test 3
2.3	3.4	Show further details of two volunteers	Normal	Sam Richell, Show more detail	Further details of two volunteers	Further details of two volunteers	Pass		Test 3
2.4	4.1.1	Add new preferred participation choice	Normal	Add New, 1	Shows new preferred participation "1"	Shows new preferred participation "1"	Pass		Test 4
2.5	4.1.2	Delete preferred participation choice	Normal	Delete	Deletes new preferred participation "1"	Deletes new preferred participation "1"	Pass		Test 4
2.6	4.2.1	Add new event	Normal	Add New, 1	Shows new event "1"	Shows new event "1"	Pass		Test 4
2.7	4.2.2	Delete event	Normal	Delete	Deletes new event "1"	Deletes new event "1"	Pass		Test 4
3.3/ 3.4	5.1.1	Attempt to login with wrong email	Erroneous	samrichell17@notemail.com Password123	Doesn't log in	Doesn't log in	Pass		Test 5
3.3/ 3.4	5.1.2	Attempt to login with wrong password	Erroneous	samrichell17@outlook.com Password1234	Doesn't log in	Doesn't log in	Pass		Test 5

3.3/ 3.4	5.1.4	Attempt to login with existing user details	Normal	samrichell17@outlook.com Password123	User logged in	User logged in	Pass		Test 5
3.6	5.2	Is user directed to next page after valid details entered	Normal	samrichell17@outlook.com Password123	User directed to next page	User directed to next page	Pass		Test 5
4.3.1	6.1.1	Changing volunteer's name	Normal	Samuel	Details changed	Details don't update	Fail	Appendix 5: Update Details.php – User deleted before checking whether email/ phone number already in use	Test 6
4.3.1	6.1.2	Changing volunteer's name	Erroneous	^NOTQ^	Form submission fails	Form submission fails	Pass		Test 6
4.3.2	6.2.1	Changing volunteer's email	Normal	Samuelrichell@outlook.com	Details changed	Details don't update	Fail	Appendix 5: Update Details.php – User deleted before checking whether email/ phone number already in use	Test 6
4.3.2	6.2.2	Changing volunteer's email	Erroneous	Thisisnotanemail	Form submission fails	Form submission fails	Pass		Test 6

4.3.3	6.3.1	Changing volunteer's password	Normal	Password1234	Details changed	Details don't update	Fail	Appendix 5: Update Details.php – User deleted before checking whether email/ phone number already in use	Test 6
4.3.3	6.3.2	Changing volunteer's password	Boundary	Thisis7	Form submission fails	Form submission fails	Pass		Test 6
4.3.4	6.4.1	Changing volunteer's mobile number	Normal	07712345678	Details changed	Details don't update	Fail	Appendix 5: Update Details.php – User deleted before checking whether email/ phone number already in use	Test 6
4.3.4	6.4.2	Changing volunteer's mobile number	Erroneous	notaphonenumber	Form submission fails	Form submission fails	Pass		Test 6
4.3.5	6.5	Changing volunteer's address	Normal	New address road	Details changed	Details don't update	Fail	Appendix 5: Update Details.php – User deleted before checking whether email/ phone number already in use	Test 6

4.3.5	6.6.1	Changing volunteer's postcode	Normal	GU14 9PA	Details changed	Details don't update	Fail	Appendix 5: Update Details.php – User deleted before checking whether email/ phone number already in use	Test 6
4.3.5	6.6.2	Changing volunteer's postcode	Erroneous	GU158 145PT	Form submission fails	Form submission fails	Pass		Test 6
4.3.6	6.7.1	Changing volunteer's preferred participation	Normal	1/3 Preferred Participation Selected	Details changed	Details don't update	Fail	Appendix 5: Update Details.php – User deleted before checking whether email/ phone number already in use	Test 6
4.3.6	6.7.2	Changing volunteer's preferred participation	Erroneous	0/3 Preferred participation Selected	Error telling user to select at least 1 preferred participation choice shows	Error telling user to select at least 1 preferred participation choice shows	Pass		Test 6
4.4	6.8	Sign out button signs user out	Normal	Sign out button pressed	User signed out	User signed out	Pass		Test 6
4.3.1	6.1.1 (retest)	Changing volunteer's name	Normal	Samuel	Details changed	Details changed	Pass		Test 6 (retest)

4.3.2	6.2.1 (retest)	Changing volunteer's email	Normal	Samuelrichell@outlook.com	Details changed	Details changed	Pass		Test 6 (retest)
4.3.3	6.3.1 (retest)	Changing volunteer's password	Normal	Password1234	Details changed	Details changed	Pass		Test 6 (retest)
4.3.4	6.4.1 (retest)	Changing volunteer's mobile number	Normal	07712345678	Details changed	Details changed	Pass		Test 6 (retest)
4.3.5	6.5 (retest)	Changing volunteer's address	Normal	New address road	Details changed	Details changed	Pass		Test 6 (retest)
4.3.5	6.6.1 (retest)	Changing volunteer's postcode	Normal	GU14 9PA	Details changed	Details changed	Pass		Test 6 (retest)
4.3.6	6.7.1 (retest)	Changing volunteer's preferred participation	Normal	1/3 Preferred Participation Selected	Details changed	Details changed	Pass		Test 6 (retest)

EVALUATION

OBJECTIVE ANALYSIS

Objective	Met?	Comment
1.1	Yes	The sign-up page is a web page however for now it is only accessible by knowing the IP address of the host computer and only when a certain program is being used to locally host the web site. A fix to this could be paying to use a web hosting service and uploading the files to the hosting service. Another benefit of this approach would be that the hosting service would provide a security certificate.
1.2	Yes	I have met both 1.2.1 and 1.2.2. Clear labels show the users where to input information and clear buttons contrast the background to minimise any confusion when using the solution. A possible extension to this could be adding a high contrast mode or larger text allowing easier access and navigation to those with sight problems.
1.3	Yes	The sign-up page very clearly allows users to enter all the information they need to and doesn't allow them to enter anything which is either in the wrong format or the database won't allow. A future addition to this could be only allowing users to enter real addresses and postcodes however this may require complicated code or an API.
2.1	Yes	The password page has a predefined password which will only allow leadership at Lightwater Connected to access it. However, this may not be the most secure way to store the password as when the code is compiled it may be possible for someone to extract the password from the .exe file, they may even be able to brute force the password. Two separate fixes to these problems could include: storing the password in the database in an encrypted form, introducing a 3 attempts every minute solution.
2.2	Yes	The program can access the database
2.3	Yes	The program can return lists of volunteers based on forename, surname, postcode and certain preferred participation parameters. An extension on this could be returning volunteers based on events they have worked at.
2.4	Yes	The program can add preferred participation options.

2.5	Yes	The program can delete preferred participation option. An extension on 2.4 and 2.5 could be allowing the program to edit the preferred participation options
2.6	Yes	The program can add events.
2.7	Yes	The program can delete events. Just like the 2.4 and 2.5, an extension to 2.6 and 2.7 could be allowing the program to edit the events. One critique of the events functionality at the moment is it serves almost no purpose, you cannot assign volunteers to events, and you can't access any details of the events, you can only see their name. There is plenty of functionality that these events can add such as a possible new page logging events volunteers have attended and allowing them to sign up to new ones.
2.8	Yes	I have met 2.8.1, 2.8.2 and 2.8.3. Clear labels show the users where to input information and clear buttons contrast the background to minimise any confusion when using the solution and the data produced in the tables is easy to read. Although 2.8.4 has not been met, I still believe that 2.8 (Program must be user friendly) has been met. The idea for a CSV file that can be printed came from the leader of Lightwater Connected claiming that he liked to use the current system to print off certain data. He has since resigned and once a new leader is elected, if they request the same, then the feature can be added.
2.9	No	Due to time constraints, my solution has failed on this objective. Perhaps a simpler solution to this problem could have been producing a list of all returned emails which can be copied and then pasted into another email service such as outlook.
3.1	Yes	The login page is a web page however it suffers the same drawbacks as 1.1.
3.2	Yes	The login page is user friendly, and any additional ideas are written in 1.2.
3.3	Yes	Users can enter an email to login. A future addition to this feature could be allowing login using a phone number or each user having a username unique to each account.
3.4	Yes	The user can enter a password however once the password is entered there is no way the user can check they've entered the right password before the login button is pressed. A show password button could be implemented to fix this.

3.5	Yes	The login page authenticates users when a correct email and password combination is inputted and returns them to the login page if wrong details are entered.
3.6	Yes	When correct details are entered, users are directed to the settings page where they can alter their data. A home page could be added where users can perform other tasks from, however, for now there are no other tasks so there is no need for a home page.
3.7	No	Due to time constraints, an automated password recovery system has not been implemented. This could have been implemented as an email sending a user their password however this may not have been the most secure system.
4.1	Yes	The settings page is a web page however it has the same drawbacks as 1.1 and 3.1
4.2	Yes	The settings page is user friendly and could be updated to be more inclusive using the same ways outlined in 1.2.
4.3	Yes	The volunteers can alter their specific data and change their preferred participation however an additional feature could be allowing volunteers to delete their accounts.
4.4	Yes	The volunteers can sign out of their accounts allowing for extra security.
5.1	Yes	The database can store all the required information for each volunteer and any new information can be easily added
5.2	Yes	Preferred participation data can be stored, and assignment is stored and required when users are entering the data.
5.3	Yes	Although the events do not serve much of a purpose, the details of events can be stored allowing for future updates to add functionality to these events.

USER FEEDBACK

Due to the leader of Lightwater Connected resigning a couple of months ago and a new leader not having been elected yet, I requested that the technical volunteer at Lightwater Connected provide feedback. And I have laid out the main points:

VOLUNTEER USER APPLICATION

LOGIN PAGE

“The login page is nice and easy to use - it’s clear what I should do. It’s also nice and clear how to sign up if I don’t have an account yet. I like that my password isn’t displayed when I enter it. However, it would be nice to show some kind of error message if an incorrect email and password is entered.”

SIGN UP PAGE

“The signup page is also very clear and easy. Nice use of required fields, and there’s good validation when entering the phone number and postcode, although it would be nice to have some guidance shown on the screen about the expected format. Same for the password. Again, nice that this is masked when entering it. Layout on the participation options could be a bit tidier - left-aligned would look much neater. I liked that it would not let me register a second user with the same email or phone number, but ideally it would show an error message explaining this, rather than just clearing the form and showing it again.”

UPDATE DETAILS PAGE

“The page to edit my current details is nice and clear. I like that it displays my current details so I know what I’m changing. It shouldn’t really show my current password though. It’s good that if I try to submit it with no preferred participation options, it tells me that I can’t do this, and I have to select at least one. Nice clear link to sign out.”

ADMIN APPLICATION

PASSWORD PAGE

“Nice use of Python TKinter to make a simple windowed application - this is much easier to use than a console application. Nice simple login screen. Support for multiple administrators would be a nice feature to add in the future.”

MENU PAGE

“Good clear main menu screen showing the functions which are available.”

SEARCH PAGE

“Good search function here. I like that I can search for a specific user or search for volunteers for a specific activity. Good display of the search results. They could be a bit more compact though, so that I can see more results on a single page. A feature to export the results, (e.g. to a CSV file) would be a nice feature to add in the future.”

PREFERRED PARTICIPATION SETTINGS PAGE

"Very easy to add more participation options and to delete old ones. It would be nice to have some sort of "Are you sure?" confirmation before deleting an option though. Allowing editing options (e.g. to fix spelling mistakes) would be a nice future feature."

EVENT SETTINGS PAGE

"The events feature is very basic, but I recognise that it was only included to demonstrate how the application could support additional features - it is not intended to be a fully built feature."

SUMMARY

"Both applications are very clearly laid out and easy to find my way around. By keeping the applications separate there is no danger of service users somehow being able to access administrative functions, which is a good security measure. I like the use of the Lightwater Connected green colour scheme to make it feel familiar to existing users. I didn't encounter any bugs when using either application, so they appear to have been well tested already."

POSSIBLE EXTENSIONS

Had I had more time, I would have included an assortment of features including some that aid accessibility such as a high contrast mode or larger text for the web page. Most of the local volunteers are retired people who volunteer as part of their social life. In general, older people tend to be worse at using tech and more likely to have physical impairments so features aiding accessibility are almost vital to this.

To expand on the events. A possible way they could be implemented is a home web page where volunteers can see which events they have previously attended and allow them to sign up to new events. In addition, the admin program could have a new search function allowing admins to search for volunteers which have attended previous events. A positive to this would be when planning a recurring event, people who have attended in the past can easily be asked to attend future versions of the events.

Another feature that could be added is one allowing the admin to not only do AND searches but OR searches. The current system will create an SQL search query which returns volunteers who fulfil all the selected characteristics instead of those who fill maybe only one or two of them. This could be completed without a time constraint.

There are plenty of different ways that more detailed SQL queries could be implemented such as implementing one which returns volunteers which attended any events within a specific time frame. The biggest challenge of this and the feature before would be implementing it into the GUI however with enough time it could be done.

REFERENCES

BBC. (2023, September 15). *Register with the BBC*. Retrieved from BBC: https://account.bbc.com/register/details?ab=o18&action=register&clientId=Account&context=news&isCasso=false&nonce=OH0xblsR-s9rKdsZnKMedClb77_P4soC78LA&ptrt=https%3A%2F%2Fwww.bbc.co.uk%2Fnews&realm=%2F&redirectUri=https%3A%2F%2Fsession.bbc.co.uk%2Fsession%

Lightwater Connected. (2023, September 15). *Lightwater Connected*. Retrieved from Facebook: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.facebook.com%2FGU18Connected%2F&psig=AOvVaw1PlmKA9jn7-XMG1taJmTE2&ust=1695317312859000&source=images&cd=vfe&opi=89978449&ved=0CA8QjhxqFwoTCNDAnajbuYEDFQAAAAAdAAAAABAI

macrotrends. (2023, September 14). *Oracle Net Worth 2010-2023 | ORCL*. Retrieved from macrotrends: https://www.macrotrends.net/stocks/charts/ORCL/oracle/net-worth

Mailchimp. (2023, September 14). Retrieved from Mailchimp: https://mailchimp.com/?reqhost=mailchimp.co.uk

MainlyWebStuff. (2024, February 23). *Connecting to a SQLite database with PHP*. Retrieved from YouTube: https://www.youtube.com/watch?v=bR3nxnCGqmY&list=PLU70qqWW4frENsWYAm-tAKp2ZJQ_dt3WR&index=2

MainlyWebStuff. (2024, February 26). *Creating an INSERT form to add records - PHP and SQLITE*. Retrieved from YouTube: https://www.youtube.com/watch?v=cyl0Oj3rmmg&list=PLU70qqWW4frENsWYAm-tAKp2ZJQ_dt3WR&index=8

Prakash_d22. (2024, March 1). *How to show an alert box in PHP? [closed]*. Retrieved from Stack Overflow: https://stackoverflow.com/questions/13837375/how-to-show-an-alert-box-in-php

Tom.Slick. (2024, January 25). *Python Tkinter clearing a frame*. Retrieved from Stack Overflow: https://stackoverflow.com/questions/15781802/python-tkinter-clearing-a-frame

TutorialRepublic. (2024, March 3). *How to Strip All Spaces Out of a String in PHP*. Retrieved from TutorialRepublic: https://www.tutorialrepublic.com/faq/how-to-strip-all-spaces-out-of-a-string-in-php.php

UCAS. (2023, September 15). *Register*. Retrieved from UCAS: https://accounts.ucas.com/Account/Login

W3Schools. (2024, March 1). *PHP count() Function*. Retrieved from W3Schools: https://www.w3schools.com/php/func_array_count.asp

W3Schools. (2024, March 1). *PHP for loop*. Retrieved from W3Schools: https://www.w3schools.com/php/php_looping_for.asp

W3Schools. (2024, March 1). *PHP isset() Function*. Retrieved from W3Schools: https://www.w3schools.com/php/func_var_isset.asp#:~:text=The%20isset()%20function%20checks,NULL%2C%20otherwise%20it%20returns%20false.

Wikipedia. (2023, September 14). *Oracle Corporation*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Oracle_Corporation

APPENDIX

APPENDIX 1: ADMIN PROGRAM.PY

```
1. ############ SAM RICHELL COMPUTER SCIENCE NEA ############
2.
3. from tkinter import *
4. from tkinter import ttk
5. import os
6. import sqlite3
7.
8.
9. background_colour = "#98F0AA"
10.      button_background_colour = "#048106"
11.      active_button_background_colour = "#049006"
12.      button_text_colour = "#FFFFFF"
13.      label_text_colour = "#000000"
14.      text_box_text_colour = "#000000"
15.      tex_box_background_colour = "#FFFFFF"
16.
17.      # textfont for the whole
18.      text_font = "Arial"
19.
20.
21.      def setup(bgc, bbc, btc, ltc, tbtc, tbbc, abgc):
22.          window = Tk()
23.          #window.geometry("960x540")
24.          window.configure(bg=bgc)
25.          style = define_style(window)
26.          style.configure_button(btc, bbc, abgc)
27.          style.configure_label(ltc, bgc)
28.          style.configure_entry(tbtc, tbbc)
29.          style.configure_checkbutton(bgc)
30.          style.configure_frame(bgc)
31.          return window
32.
33.
34.      class Volunteer:
35.          def __init__(self, ID, forename, surname, email,
   mobilenumber, address, postcode):
36.              self.__ID = ID
37.              self.__forename = forename
38.              self.__surname =surname
39.              self.__email = email
40.              self.__mobilenumber = mobilenumber
41.              self.__address = address
42.              self.__postcode = postcode
43.              self.__PreferredParticipation = []
44.
45.          def add_PP(self, PP):
46.              self.__PreferredParticipation.append(PP)
47.
48.          def get_ID(self):
49.              return self.__ID
50.
51.          def get_forename(self):
52.              return self.__forename
```

```
53.
54.         def get_surname(self):
55.             return self.__surname
56.
57.         def get_email(self):
58.             return self.__email
59.
60.         def get_mobilenumber(self):
61.             return self.__mobilenumber
62.
63.         def get_address(self):
64.             return self.__address
65.
66.         def get_postcode(self):
67.             return self.__postcode
68.
69.         def get_PP(self):
70.             return self.__PreferredParticipation
71.
72.
73.     class define_style:
74.         def __init__(self, window):
75.             self.__style = ttk.Style(window)
76.             self.__style.theme_use("alt")
77.
78.         def configure_button(self, fgc, bgc, abgc):
79.             self.__style.configure("TButton", foreground=fgc,
    background=bgc)
80.             self.__style.map("TButton",
    background=[("active", abgc)])
81.
82.         def configure_label(self, fgc, bgc):
83.             self.__style.configure("TLabel", foreground=fgc,
    background=bgc)
84.
85.         def configure_entry(self, fgc, bgc):
86.             self.__style.configure("TEntry", foreground=fgc,
    background=bgc)
87.
88.         def configure_checkbutton(self, bgc):
89.             self.__style.configure("TCheckbutton",
    background=bgc)
90.             self.__style.map("TCheckbutton",
    background=[("active", bgc)])
91.
92.         def configure_frame(self, bgc):
93.             self.__style.configure("TFrame", background=bgc)
94.
95.
96.     def create_frame(root):
97.         frame = ttk.Frame(root)
98.         frame.grid(column=0, row=0, sticky=(N, E, S, W))
99.         return frame
100.
101.
102.     class error_window:
103.         def __init__(self, message, textfont):
104.
105.             self.__textfont = textfont
```

```
106.
107.             self.__box_window = Tk()
108.             self.__box_window.configure(bg="#FFFFFF")
109.             self.__box_window.geometry("150x50+400+300")
110.
111.             self.__frame = create_frame(self.__box_window)
112.
113.             self.__error_label = ttk.Label(self.__frame,
    text=message, font=(self.__textfont, 10))
114.             self.__error_label.grid(column=0, row=0,
    sticky=(W, E))
115.
116.             self.__ok_button = ttk.Button(self.__frame,
    text="ok", command=self.__box_window.destroy)
117.             self.__ok_button.grid(column=0, row=1, sticky=(W,
    E))
118.
119.             self.__frame.pack()
120.
121.
122.     class Page: # Parent class to all other pages
123.         def __init__(self, window, textfont, title):
124.             self._textfont = textfont
125.             self._window = window
126.             self._window.title(title)
127.
128.         def _create_frame(self):
129.             self._mainframe = create_frame(self._window)
130.             self._mainframe.pack(padx=250, pady=100)
131.
132.         def _display_title(self, page_title):
133.             self._page_title = ttk.Label(self._mainframe,
    text=page_title, font=(self._textfont, 30, "bold"))
134.
135.         def _clear_frame(self):
136.             for widget in self._mainframe.winfo_children():
137.                 widget.destroy()
138.
139.         def _destroy_frame(self):
140.             self._mainframe.destroy()
141.
142.         def _connect(self):
143.             self._DBPath =
    os.path.join(os.path.dirname(__file__), "VolunteerDB.sqlite3")
    #Defining the pathway to the sqlite3 file
144.
145.             self._SQLiteConnection =
    sqlite3.connect(self._DBPath) #Connecting to the sql file
146.             self._cursor = self._SQLiteConnection.cursor()
    #creating the cursor
147.
148.         def _close_connection(self):
149.             self._cursor.close()
150.             self._SQLiteConnection.close()
151.
152.         def _display_home_button(self):
153.             self.__home_button = ttk.Button(self._window,
    text="Home", command=self.__home)
154.             self.__home_button.place(x=5, y=5)
```

```
155.
156.         def __home(self): # Subclasses should not be able to
   access
157.             self.__home_button.destroy()
158.             self._clear_frame()
159.             self._destroy_frame()
160.             menu_page(self._window, self._textfont)
161.
162.
163.     class password_page(Page): # Password page class
164.         def __init__(self, window, textfont):
165.             super().__init__(window, textfont, "Login")
166.             self._create_frame()
167.             self._display_title("Welcome")
168.
169.             self._page_title.grid(column=0, row=0,
   columnspan=2, padx=5, pady=20)
170.
171.             self.__correct_password = ""
172.
173.             self.__password = StringVar()
174.             self.__password_entry =
   ttk.Entry(self._mainframe, textvariable=self.__password,
   width=20, show="*")
175.             self.__password_entry.grid(column=1, row=1,
   sticky=(W, E), padx=5, pady=10)
176.             self.__password_entry.focus()
177.
178.             self.__login_button = ttk.Button(self._mainframe,
   text="Login", command=self.__check_password, width=20)
179.             self.__login_button.grid(column=0, row=2,
   columnspan=2, sticky=(W, E), padx=5, pady=5)
180.
181.             self.__password_label =
   ttk.Label(self._mainframe, text="Password",
   font=(self._textfont, 16))
182.             self.__password_label.grid(column=0, row=1,
   sticky=(W, E), padx=5, pady=5)
183.
184.         def __check_password(self):       # Checks to make
   sure the input into the password entry is correct
185.             if self.__password.get() ==
   self.__correct_password:
186.                 self.__next_page()
187.             else:
188.                 self.__report_error("Password Incorrect")
189.
190.         def __next_page(self):            # Take user to menu
   page
191.             self._clear_frame()
192.             self._destroy_frame()
193.             menu_page(self._window, self._textfont)
194.
195.         def __report_error(self, message): # Uses
   error_window class to report an error to the user
196.             error_window(message, self._textfont)
197.
198.
199.     class menu_page(Page): # Menu page class
```

```
200.        def __init__(self, window, textfont):
201.            super().__init__(window, textfont, "Menu")
202.            self._create_frame()
203.            self._display_title("Welcome Admin")
204.            self._page_title.grid(column=0, row=0, pady=20)
205.
206.            self.__find_volunteers =
    ttk.Button(self._mainframe, text="Find Volunteer(s)",
    command=self.__find_volunteers_page)
207.            self.__find_volunteers.grid(column=0, row=1,
    pady=10, sticky=(W, E))
208.
209.            self.__PP_settings = ttk.Button(self._mainframe,
    text="Preferred Participation Settings",
    command=self.__PP_settings_page)
210.            self.__PP_settings.grid(column=0, row=2, pady=10,
    sticky=(W, E))
211.
212.            self.__event_settings =
    ttk.Button(self._mainframe, text="Event Settings",
    command=self.__events_settings_page)
213.            self.__event_settings.grid(column=0, row=3,
    pady=10, sticky=(W, E))
214.
215.            self.__log_out = ttk.Button(self._mainframe,
    text="Log Out", command=self.__logout)
216.            self.__log_out.grid(column=0, row=4, pady=10,
    sticky=(W, E))
217.
218.        def __find_volunteers_page(self):
219.            self._clear_frame()
220.            self._destroy_frame()
221.            find_volunteers_page(self._window,
    self._textfont)
222.
223.        def __PP_settings_page(self):
224.            self._clear_frame()
225.            self._destroy_frame()
226.
    Preferred_Participation_Settings_Page(self._window,
    self._textfont)
227.
228.        def __events_settings_page(self):
229.            self._clear_frame()
230.            self._destroy_frame()
231.            Event_Settings_Page(self._window, self._textfont)
232.
233.        def __logout(self):
234.            self._clear_frame()
235.            self._destroy_frame()
236.            password_page(self._window, self._textfont)
237.
238.
239.    class find_volunteers_page(Page): # Search for volunteers
    page class
240.        def __init__(self, window, textfont):
241.            super().__init__(window, textfont, "Find
    Volunteers")
242.            self._create_frame()
```

```
243.             self._display_title("Find Volunteer(s)")
244.
245.             self._page_title.grid(column=0, row=0, pady=20,
   columnspan=3)
246.
247.             self.__forename = StringVar()
248.             self.__forename_entry =
   ttk.Entry(self._mainframe, textvariable=self.__forename)
249.             self.__forename_entry.grid(column=1, row=1,
   columnspan=2, pady=15, sticky=(W, E))
250.
251.             self.__forename_label =
   ttk.Label(self._mainframe, text="Forename",
   font=(self._textfont, 14))
252.             self.__forename_label.grid(column=0, row=1)
253.
254.             self.__surname = StringVar()
255.             self.__surname_entry = ttk.Entry(self._mainframe,
   textvariable=self.__surname)
256.             self.__surname_entry.grid(column=1, row=2,
   columnspan=2, pady=15, sticky=(W, E))
257.
258.             self.__surname_label = ttk.Label(self._mainframe,
   text="Surname", font=(self._textfont, 14))
259.             self.__surname_label.grid(column=0, row=2)
260.
261.             self.__postcode = StringVar()
262.             self.__postcode_entry =
   ttk.Entry(self._mainframe, textvariable=self.__postcode)
263.             self.__postcode_entry.grid(column=1, row=3,
   columnspan=2, pady=15, sticky=(W, E))
264.
265.             self.__postcode_label =
   ttk.Label(self._mainframe, text="Postcode",
   font=(self._textfont, 14))
266.             self.__postcode_label.grid(column=0, row=3)
267.
268.             self.__display_PP()
269.
270.             self.__search_button =
   ttk.Button(self._mainframe, text="Search",
   command=self.__search)
271.             self.__search_button.grid(column=2,
   row=self.__count, pady=15, sticky=(E))
272.
273.             self._display_home_button()
274.
275.         def __display_PP(self):
276.
277.             self._connect()
278.
279.             query = """SELECT ParticipationName
280.             FROM PreferredParticipation;"""
281.
282.             self.__PP_array = []
283.             self.__count = 4
284.
285.             for row in self._cursor.execute(query):
286.
```

```
287.                 self.__PP_label = ttk.Label(self._mainframe,
    text=row[0], font=(self._textfont, 14))
288.                 self.__PP_label.grid(column=0,
    row=self.__count, columnspan=2, pady=15)
289.
290.                 self.__PP_choice = IntVar()
291.                 self.__PP_checkbutton =
    ttk.Checkbutton(self._mainframe, variable=self.__PP_choice)
292.                 self.__PP_checkbutton.grid(column=2,
    row=self.__count)
293.
294.                 self.__PP = {"label": self.__PP_label,
295.                              "text": row[0],
296.                              "variable": self.__PP_choice,
297.                              "button": self.__PP_checkbutton}
    # All the values for each of the Preferred Participation
    buttons are stored as a dictionary
298.
299.                 self.__PP_array.append(self.__PP)
300.
301.                 self.__count += 1
302.
303.             self._close_connection()
304.
305.         def __display_results(self):
306.
307.             self.__volunteers =
    Volunteer_Stack(self.__returned_volunteers)
308.
309.             self.__result_frame =
    Result_Frame(self._mainframe, self._textfont)
310.
311.             self.__next_results()
312.
313.             self.__show_in_detail_button =
    ttk.Button(self._mainframe, text="Show More Detail",
    command=self.__show_in_detail)
314.             self.__show_in_detail_button.grid(column=1,
    row=1)
315.
316.         def __display(self):
317.
318.             try:
319.                 self.__result_frame.reset()
320.             except:
321.                 pass
322.
323.
    self.__result_frame.display_frame(self.__to_display)
324.
325.             if self.__volunteers.is_to_see_empty() != True: #
    Only displays button if user can go forward
326.                 self.__next_button =
    ttk.Button(self._mainframe, text="Next",
    command=self.__next_results)
327.                 self.__next_button.grid(column=2, row=0)
328.
329.             if self.__volunteers.is_seen_empty() != True: #
    Only displays button if user can go backwards
```

```
330.                 self.__back_button =
    ttk.Button(self._mainframe, text="Back",
    command=self.__previous_results)
331.                 self.__back_button.grid(column=0, row=0)
332.
333.         def __next_results(self):
334.             self.__volunteers.four_forward()
335.             self.__to_display =
    self.__volunteers.get_current()
336.             self.__remove_result_buttons()
337.             self.__display()
338.
339.         def __previous_results(self):
340.             self.__volunteers.four_backward()
341.             self.__to_display =
    self.__volunteers.get_current()
342.             self.__remove_result_buttons()
343.             self.__display()
344.
345.         def __remove_result_buttons(self):   # Will remove
    any buttons that are there
346.             try:
347.                 self.__back_button.destroy()
348.             except:
349.                 pass
350.             try:
351.                 self.__next_button.destroy()
352.             except:
353.                 pass
354.
355.         def __add_volunteer(self, row):
356.             if row[0] in self.__returned_IDs:
357.
358.                 for volunteer in self.__returned_volunteers:
359.                     if volunteer.get_ID() == row[0]:
360.                         volunteer.add_PP(row[7])
361.
362.             else:
363.                 self.__volunteer = Volunteer(row[0], row[1],
    row[2], row[3], row[4], row[5], row[6])
364.                 self.__volunteer.add_PP(row[7])
365.
    self.__returned_volunteers.append(self.__volunteer)
366.                 self.__returned_IDs.append(row[0])
367.
368.         def __search(self):
369.             self._connect()
370.
371.             self.__search_for = []   # List for all that
372.
373.             if self.__forename.get() != "":
374.                 self.__search_for.append({"type": "Forename",
    "value": self.__forename.get(), "PP": False})
375.
376.             if self.__surname.get() != "":
377.                 self.__search_for.append({"type": "Surname",
    "value": self.__surname.get(), "PP": False})
378.
379.             if self.__postcode.get() != "":
```

```
380.                 self.__search_for.append({"type": "Postcode",
    "value": self.__postcode.get(), "PP": False})
381.
382.             for PP in self.__PP_array:
383.                 if PP["variable"].get() == 1:
384.                     self.__search_for.append({"type":
    PP["text"], "value": PP["variable"].get(), "PP": True})
385.
386.             self.__command = """SELECT
    Volunteers.VolunteerID, Volunteers.Forename,
    Volunteers.Surname, Volunteers.Email, Volunteers.MobileNumber,
    Volunteers.Address, Volunteers.Postcode,
    PreferredParticipation.ParticipationName
387.             FROM Volunteers, PreferredParticipation,
    ParticipationAssignment
388.             WHERE""" # start to the SQL query to find
    volunteers
389.
390.             for PP in self.__search_for:
391.                 if PP["PP"] == True:
392.                     self.__command += f"
    PreferredParticipation.ParticipationName = '{PP['type']}' AND"
393.                 else:
394.                     self.__command += f"
    Volunteers.{PP['type']} = '{PP['value']}' AND"
395.
396.             self.__command += f" Volunteers.VolunteerID =
    ParticipationAssignment.VolunteerID AND
    ParticipationAssignment.ParticipationID =
    PreferredParticipation.ParticipationID;"
397.
398.             self._cursor.execute(self.__command)
399.             self.__results = self._cursor.fetchall()
400.
401.             if len(self.__results) == 0:          # Results
    in an error when no volunteers are returned
402.                 error_window("No Volunteers Found", "Arial")
403.             else:
404.                 self.__returned_volunteers = []
405.                 self.__returned_IDs = []
406.
407.                 for row in self.__results:
408.                     self.__add_volunteer(row)
409.
410.                 self._close_connection()
411.
412.                 self._clear_frame()
413.
414.                 self.__display_results()
415.
416.         def __show_in_detail(self):
417.             self.__result_frame.reset()
418.
419.             self.__show_in_detail_button.destroy()
420.             self.__remove_result_buttons()
421.
422.             self.__all_volunteers, self.__position =
    self.__volunteers.get_all()
423.
```

```
424.             self.__show_detail =
   Detailed_Volunteers(self._mainframe, self._textfont)
425.
426.
   self.__show_detail.display_volunteer(self.__all_volunteers[sel
   f.__position])
427.
428.             self.__show_detailed_buttons()
429.
430.         def __show_detailed_buttons(self):
431.             if self.__position != len(self.__all_volunteers)-
   1:
432.                 self.__next_detail_button =
   ttk.Button(self._mainframe, text="Next",
   command=self.__next_detailed)
433.                 self.__next_detail_button.grid(column=2,
   row=0)
434.
435.             if self.__position != 0:
436.                 self.__last_detail_button =
   ttk.Button(self._mainframe, text="Back",
   command=self.__last_detailed)
437.                 self.__last_detail_button.grid(column=0,
   row=0)
438.
439.         def __remove_detailed_buttons(self):
440.             try:
441.                 self.__next_detail_button.destroy()
442.             except:
443.                 pass
444.             try:
445.                 self.__last_detail_button.destroy()
446.             except:
447.                 pass
448.
449.         def __next_detailed(self):
450.             self.__remove_detailed_buttons()
451.             self.__position += 1
452.
   self.__show_detail.display_volunteer(self.__all_volunteers[sel
   f.__position])
453.             self.__show_detailed_buttons()
454.
455.         def __last_detailed(self):
456.             self.__remove_detailed_buttons()
457.             self.__position -= 1
458.
   self.__show_detail.display_volunteer(self.__all_volunteers[sel
   f.__position])
459.             self.__show_detailed_buttons()
460.
461.
462.     class Result_Frame: # Result frame to display results
   from volunteer search
463.         def __init__(self, root_frame, textfont):
464.
465.             self.__root_frame = root_frame
466.             self.__textfont = textfont
467.
```

```
468.        def __create_frame(self, root_frame):
469.            self.__frame = create_frame(root_frame)
470.            self.__frame.grid(row=0, column=1)
471.
472.        def reset(self):
473.            for widget in self.__frame.winfo_children():
474.                widget.destroy()
475.            self.__frame.destroy()
476.
477.        def display_frame(self, volunteers):
478.
479.            self.__create_frame(self.__root_frame)
480.
481.            self.__currently_displayed = []
482.
483.            self.__name_label = ttk.Label(self.__frame,
    text="Name", font=(self.__textfont, 12,  "bold"))
484.            self.__name_label.grid(column=0, row=0, pady=40,
    padx=40)
485.
486.            self.__postcode_label = ttk.Label(self.__frame,
    text="Postcode", font=(self.__textfont, 12, "bold"))
487.            self.__postcode_label.grid(column=1, row=0,
    pady=40, padx=40)
488.
489.            self.__PP_label = ttk.Label(self.__frame,
    text="Preferred Participation", font=(self.__textfont, 12,
    "bold"))
490.            self.__PP_label.grid(column=2, row=0, pady=40,
    padx=20)
491.
492.            self.__count = 1
493.            for value in volunteers:
494.
495.                name_label = ttk.Label(self.__frame,
    text=f"{value.get_forename()} {value.get_surname()}",
    font=self.__textfont)
496.                name_label.grid(column=0, row=self.__count,
    pady=40)
497.
498.                postcode_label = ttk.Label(self.__frame,
    text=value.get_postcode(), font=self.__textfont)
499.                postcode_label.grid(column=1,
    row=self.__count, pady=40)
500.
501.                PP_string = ""
502.
503.                for PP in value.get_PP():
504.                    PP_string += f"{PP}\n"
505.
506.                PP_string = PP_string[0:-1]   # Removes the
    final \n from the string
507.
508.                PP_label = ttk.Label(self.__frame,
    text=PP_string, font=self.__textfont)
509.                PP_label.grid(column=2, row=self.__count,
    sticky=W)
510.
511.                volunteer = {"Name Label": name_label,
```

```
512.                                 "Postcode Label":
    postcode_label,
513.                                 "PP Label": PP_label}
514.
515.                 self.__currently_displayed.append(volunteer)
516.                 self.__count += 1
517.
518.
519.     class Stack: # Stack allowing easy viewing of
520.         def __init__(self, array):
521.             self.__stack = array
522.             self.__rear_pointer = len(self.__stack)-1
523.
524.         def get_value(self):
525.             to_return = self.__stack.pop(self.__rear_pointer)
526.             self.__rear_pointer -= 1
527.             return to_return
528.
529.         def add_value(self, value):
530.             self.__stack.append(value)
531.             self.__rear_pointer += 1
532.
533.         def is_empty(self):
534.             if len(self.__stack) == 0:
535.                 empty = True
536.             else:
537.                 empty = False
538.             return empty
539.
540.         def get_all(self):
541.             return self.__stack
542.
543.
544.     class Volunteer_Stack:
545.         def __init__(self, volunteers):
546.             self.__to_see = Stack(volunteers)
547.             self.__seen = Stack([]) # No volunteers have been
    seen yet so stack starts empty
548.             self.__not_volunteer = Volunteer(0, "", "", "",
    "", "", "-")
549.             self.__not_volunteer.add_PP("")
550.             self.__current = []
551.
552.         def __forward(self):
553.             try:
554.                 value = self.__to_see.get_value()
555.             except: # This will fill in any empty slots as a
    dash
556.                 value = self.__not_volunteer
557.             self.__current.append(value)
558.
559.         def __backward(self):
560.             value = self.__seen.get_value()
561.             self.__current.append(value)
562.
563.         def four_forward(self):
564.             for value in self.__current[::-1]:    # Reversing
    the appended current array means the volunteers will be added
    back to the stack in the correct order
```

```
565.                 self.__seen.add_value(value)
566.             self.__current = []
567.             for i in range(4):
568.                 self.__forward()
569.
570.         def four_backward(self):
571.             for value in self.__current[::-1]:   # Reversing
    the appended current array means the volunteers will be added
    back to the stack in the correct order
572.                 self.__to_see.add_value(value)
573.             self.__current = []
574.             for i in range (4):
575.                 self.__backward()
576.
577.         def get_current(self):
578.             return self.__current
579.
580.         def get_all(self):
581.             seen = self.__seen.get_all()
582.             to_see = self.__to_see.get_all()
583.             position = len(seen)
584.             volunteer_array = []
585.             for volunteer in seen[::-1]:
586.                 volunteer_array.append(volunteer)
587.             for volunteer in self.__current:
588.                 volunteer_array.append(volunteer)
589.             for volunteer in to_see[::-1]:
590.                 volunteer_array.append(volunteer)
591.             return volunteer_array, position
592.
593.         def is_seen_empty(self):
594.             return self.__seen.is_empty()
595.
596.         def is_to_see_empty(self):
597.             return self.__to_see.is_empty()
598.
599.
600.     class Detailed_Volunteers:
601.         def __init__(self, root_frame, textfont):
602.             self.__root_frame = root_frame
603.             self.__textfont = textfont
604.             self.__create_frame()
605.
606.         def __create_frame(self):
607.             self.__frame = create_frame(self.__root_frame)
608.             self.__frame.grid(row=0, column=1)
609.
610.         def __reset_frame(self):
611.             for widget in self.__frame.winfo_children():
612.                 widget.destroy()
613.
614.         def display_volunteer(self, volunteer):
615.
616.             self.__reset_frame()
617.
618.             self.__name_label = ttk.Label(self.__frame,
    text=f"{volunteer.get_forename()} {volunteer.get_surname()}",
    font=(self.__textfont, 16, "bold"))
619.             self.__name_label.grid(column=1, row=0, pady=10)
```

```
620.
621.             self.__email_label = ttk.Label(self.__frame,
   text=f"{volunteer.get_email()}", font=self.__textfont)
622.             self.__email_label.grid(column=1, row=1, pady=5)
623.
624.             self.__number_label = ttk.Label(self.__frame,
   text=f"{volunteer.get_mobilenumber()}", font=self.__textfont)
625.             self.__number_label.grid(column=1, row=2, pady=5)
626.
627.             self.__address_label = ttk.Label(self.__frame,
   text=f"{volunteer.get_address()}", font=self.__textfont)
628.             self.__address_label.grid(column=1, row=3,
   pady=5)
629.
630.             self.__postcode_label = ttk.Label(self.__frame,
   text=f"{volunteer.get_postcode()}", font=self.__textfont)
631.             self.__postcode_label.grid(column=1, row=4,
   pady=5)
632.
633.             self.__PreferredParticipation_label =
   ttk.Label(self.__frame, text="Preferred Participation",
   font=(self.__textfont, 16, "bold"))
634.
   self.__PreferredParticipation_label.grid(column=1, row=5,
   pady=10)
635.
636.             PP_array = volunteer.get_PP()
637.
638.             self.__PP = []
639.
640.             for i in range (0, len(PP_array)):
641.                 PP_label = ttk.Label(self.__frame,
   text=f"{PP_array[i]}", font=self.__textfont)
642.                 PP_label.grid(column=1, row=i+6, pady=2)
643.                 PP = {"Label": PP_label,
644.                       "text": PP_array[i]}
645.                 self.__PP.append(PP)
646.
647.
648.     class Preferred_Participation_Settings_Page(Page):
649.         def __init__(self, window, textfont):
650.             super().__init__(window, textfont, "Preferred
   Participation Settings")
651.             self._create_frame()
652.
653.             self.__display_page()
654.
655.         def __display_page(self):
656.             self._display_title("Preferred Participation")
657.             self._page_title.grid(column=0, row=0,
   columnspan=3, pady=40)
658.
659.             self.__display_PP()
660.
661.             self.__add_new = ttk.Button(self._mainframe,
   text="Add New Preferred Participation",
   command=self.__add_new_PP)
662.             self.__add_new.grid(column=0, row=self.__count,
   columnspan=3, sticky=(W, E))
```

```
663.
664.             self._display_home_button()
665.
666.         def __display_PP(self):
667.
668.             self._connect()
669.
670.             query = """SELECT ParticipationName
671.             FROM PreferredParticipation;"""
672.
673.             self.__PP_array = []
674.             self.__count = 1
675.
676.             self._cursor.execute(query)
677.             self.__results = self._cursor.fetchall()
678.
679.             for row in self.__results:
680.
681.                 PP_label = ttk.Label(self._mainframe,
    text=row[0], font=(self._textfont, 14))
682.                 PP_label.grid(column=0, row=self.__count,
    columnspan=2, pady=15)
683.
684.                 PP_delete_button =
    ttk.Button(self._mainframe, text="Delete", command=lambda
    to_delete=row[0]: self.__delete_PP(to_delete))
685.                 PP_delete_button.grid(column=2,
    row=self.__count)
686.
687.                 PP = {"label": PP_label,
688.                       "text": row[0],
689.                       "button": PP_delete_button}     # All
    the values for each of the Preferred Participation buttons are
    stored as a dictionary
690.
691.                 self.__PP_array.append(PP)
692.
693.                 self.__count += 1
694.
695.             self._close_connection()
696.
697.         def __delete_PP(self, to_delete):
698.             self._connect()
699.             for PP in self.__PP_array:
700.                 if PP["text"] == to_delete:
701.                     query = f"SELECT ParticipationID FROM
    PreferredParticipation WHERE ParticipationName =
    '{PP['text']}'"
702.                     self._cursor.execute(query)
703.                     to_remove_ID, = self._cursor.fetchone()
704.
    self.__remove_from_ParticipationAssignment(to_remove_ID) #
    Delete any users connected to PP before deleting PP
705.                     self.__remove_PP(to_remove_ID)
706.             self._SQLiteConnection.commit()
707.             self._clear_frame()
708.             self.__display_page()
709.
```

```
710.        def __remove_from_ParticipationAssignment(self,
    PP_ID):
711.            delete_command = f"DELETE FROM
    ParticipationAssignment WHERE ParticipationID={PP_ID}"
712.            self._cursor.execute(delete_command)
713.
714.        def __remove_PP(self, PP_ID):
715.            remove_command = f"DELETE FROM
    PreferredParticipation WHERE ParticipationID={PP_ID}"
716.            self._cursor.execute(remove_command)
717.
718.        def __add_new_PP(self):
719.            self._clear_frame()
720.            self._display_title("Add New")
721.            self._page_title.grid(column=0, row=0,
    columnspan=3, pady=40)
722.
723.            self.__name_label = ttk.Label(self._mainframe,
    text="Title", font=(self._textfont, 14))
724.            self.__name_label.grid(column=1, row=1)
725.
726.            self.__new_name = StringVar()
727.            self.__name_entry = ttk.Entry(self._mainframe,
    textvariable=self.__new_name)
728.            self.__name_entry.grid(column=0, row=2,
    columnspan=3)
729.
730.            self.__description_label =
    ttk.Label(self._mainframe, text="Description",
    font=(self._textfont, 14))
731.            self.__description_label.grid(column=1, row=3)
732.
733.            self.__new_description = StringVar()
734.            self.__description_entry =
    ttk.Entry(self._mainframe,
    textvariable=self.__new_description, width=80)
735.            self.__description_entry.grid(column=0, row=4,
    columnspan=3)
736.
737.            self.__save_button = ttk.Button(self._mainframe,
    text="Save", command=self.__add_to_db)
738.            self.__save_button.grid(column=0, row=5,
    columnspan=3, pady=10, sticky=E)
739.
740.        def __add_to_db(self):
741.
742.            self._connect()
743.
744.            query = "SELECT ParticipationID FROM
    PreferredParticipation"
745.            self._cursor.execute(query)
746.            results = self._cursor.fetchall()
747.            last_ID = 0
748.            found = False
749.            for row in results:
750.                to_check, = row  # Splits the returned tuple
751.                if to_check != last_ID + 1 and found ==
    False:
752.                    new_ID = last_ID + 1
```

```
753.                     found = True
754.                 last_ID = to_check
755.             if found == False:
756.                 new_ID = last_ID + 1
757.
758.             command = f"INSERT INTO PreferredParticipation
    (ParticipationID, ParticipationName, ParticipationDescription)
    VALUES ({new_ID}, '{self.__new_name.get()}',
    '{self.__new_description.get()}')"
759.             self._cursor.execute(command)
760.             self._SQLiteConnection.commit()
761.
762.             self._close_connection()
763.
764.             self._clear_frame()
765.             self.__display_page()
766.
767.
768.     class Event_Settings_Page(Page):
769.         def __init__(self, window, textfont):
770.             super().__init__(window, textfont, "Event
    Settings")
771.
772.             self._create_frame()
773.
774.             self.__display_page()
775.
776.         def __display_page(self):
777.
778.             self._display_title("Event")
779.             self._page_title.grid(column=0, row=0,
    columnspan=3, pady=40)
780.
781.             self.__display_events()
782.
783.             self.__add_new = ttk.Button(self._mainframe,
    text="Add New Event", command=self.__add_new_event)
784.             self.__add_new.grid(column=0, row=self.__count,
    columnspan=3, sticky=(W, E))
785.
786.             self._display_home_button()
787.
788.         def __display_events(self):
789.             self._connect()
790.
791.             query = """SELECT EventName
792.             FROM Events;"""
793.
794.             self.__event_array = []
795.             self.__count = 1
796.
797.             for row in self._cursor.execute(query):
798.
799.                 event_label = ttk.Label(self._mainframe,
    text=row[0], font=(self._textfont, 14))
800.                 event_label.grid(column=0, row=self.__count,
    columnspan=2, pady=15)
801.
```

```
802.                 event_delete_button =
   ttk.Button(self._mainframe, text="Delete", command=lambda
   to_delete=row[0]: self.__delete_event(to_delete))
803.                 event_delete_button.grid(column=2,
   row=self.__count)
804.
805.                 event = {"label": event_label,
806.                          "text": row[0],
807.                          "delete button":
   event_delete_button}     # All the values for each of the
   Preferred Participation buttons are stored as a dictionary
808.
809.                 self.__event_array.append(event)
810.
811.                 self.__count += 1
812.
813.             self._close_connection()
814.
815.         def __add_new_event(self):
816.             self._clear_frame()
817.             self._display_title("Add New")
818.             self._page_title.grid(column=0, row=0,
   columnspan=2, pady=40, padx=70)
819.
820.             self.__name_label = ttk.Label(self._mainframe,
   text="Title", font=(self._textfont, 14))
821.             self.__name_label.grid(column=0, row=1)
822.
823.             self.__new_name = StringVar()
824.             self.__name_entry = ttk.Entry(self._mainframe,
   textvariable=self.__new_name)
825.             self.__name_entry.grid(column=1, row=1,
   sticky=(W, E))
826.
827.             self.__date_label = ttk.Label(self._mainframe,
   text="Date", font=(self._textfont, 14))
828.             self.__date_label.grid(column=0, row=2)
829.
830.             self.__new_date = StringVar()
831.             self.__date_entry = ttk.Entry(self._mainframe,
   textvariable=self.__new_date)
832.             self.__date_entry.grid(column=1, row=2,
   sticky=(W, E))
833.
834.             self.__time_label = ttk.Label(self._mainframe,
   text="Time", font=(self._textfont, 14))
835.             self.__time_label.grid(column=0, row=3)
836.
837.             self.__new_time = StringVar()
838.             self.__time_entry = ttk.Entry(self._mainframe,
   textvariable=self.__new_time)
839.             self.__time_entry.grid(column=1, row=3,
   sticky=(W, E))
840.
841.             self.__address_label = ttk.Label(self._mainframe,
   text="Address", font=(self._textfont, 14))
842.             self.__address_label.grid(column=0, row=4)
843.
844.             self.__new_address = StringVar()
```

```
845.             self.__address_entry = ttk.Entry(self._mainframe,
   textvariable=self.__new_address)
846.             self.__address_entry.grid(column=1, row=4,
   sticky=(W, E))
847.
848.             self.__postcode_label =
   ttk.Label(self._mainframe, text="Postcode",
   font=(self._textfont, 14))
849.             self.__postcode_label.grid(column=0, row=5)
850.
851.             self.__new_postcode = StringVar()
852.             self.__postcode_entry =
   ttk.Entry(self._mainframe, textvariable=self.__new_postcode)
853.             self.__postcode_entry.grid(column=1, row=5,
   sticky=(W, E))
854.
855.             self.__save_button = ttk.Button(self._mainframe,
   text="Save", command=self.__add_to_db)
856.             self.__save_button.grid(column=0, row=6,
   columnspan=2, pady=10, sticky=E)
857.
858.         def __delete_event(self, to_delete):
859.             self._connect()
860.             for PP in self.__event_array:
861.                 if PP["text"] == to_delete:
862.                     query = f"SELECT EventID FROM Events
   WHERE EventName = '{PP['text']}'"
863.                     self._cursor.execute(query)
864.                     to_remove_ID, = self._cursor.fetchone()
865.
   self.__remove_from_EventAssignment(to_remove_ID) # Delete any
   users connected to PP before deleting PP
866.                     self.__remove_event(to_remove_ID)
867.             self._SQLiteConnection.commit()
868.             self._clear_frame()
869.             self.__display_page()
870.
871.         def __remove_from_EventAssignment(self, event_ID):
872.             delete_command = f"DELETE FROM EventAssignment
   WHERE EventID={event_ID}"
873.             self._cursor.execute(delete_command)
874.
875.         def __remove_event(self, event_ID):
876.             remove_command = f"DELETE FROM Events WHERE
   EventID={event_ID}"
877.             self._cursor.execute(remove_command)
878.
879.         def __add_to_db(self):
880.
881.             self._connect()
882.
883.             query = "SELECT EventID FROM Events"
884.             self._cursor.execute(query)
885.             results = self._cursor.fetchall()
886.             greatest_ID = 0
887.             for row in results:
888.                 to_check, = row  # Splits the returned tuple
889.                 if to_check > greatest_ID:
890.                     greatest_ID = to_check
```

```
891.
892.             command = f"INSERT INTO Events (EventID,
   EventName, Date, Time, Address, Postcode) VALUES
   ({greatest_ID+1}, '{self.__new_name.get()}',
   '{self.__new_date.get()}', '{self.__new_time.get()}',
   '{self.__new_address.get()}', '{self.__new_postcode.get()}')"
893.             self._cursor.execute(command)
894.             self._SQLiteConnection.commit()
895.
896.             self._close_connection()
897.
898.             self._clear_frame()
899.             self.__display_page()
900.
901.     window = setup(background_colour,
   button_background_colour, button_text_colour,
   label_text_colour, text_box_text_colour,
   tex_box_background_colour, active_button_background_colour)
902.     password_page(window, text_font)
903.     window.mainloop()
```

Line 135-137/ 473-474/ 611-612: (Tom.Slick, 2024)

APPENDIX 2: LOGIN PAGE.PHP

```
1. <?php
2.
3. session_start();
4.
5. if ($_SESSION["logged_in"] == TRUE) {
6.     header("Location: Settings Page.php");
7. };
8.
9. ?>
10.
11.
12.     <html>
13.         <head>
14.             <title>Login Page</title>
15.             <link rel="stylesheet" type="text/css"
   href="style.css">
16.         </head>
17.         <body>
18.             <div class="banner">
19.                 <a href="Sign Up Page.php">Sign Up</a>
20.                 <span>Lightwater Connected</span>
21.             </div>
22.             <div class="content_box">
23.                 <h1>Login</h1>
24.                 <form action="Login.php" method="POST">
25.                     <label for="email">Email: <input
   type="text" id="email" name="email"></label><br><br>
26.                     <label for="password">Password: <input
   type="password" id="password" name="password"></label><br><br>
27.                     <input type="submit" value="Submit">
28.                 </form>
29.             </div>
```

```
30.         </body>
31.     </html>
```

APPENDIX 3: LOGIN.PHP

```
1. <?php
2.
3. // Starting a session allows variables to be used in other
   programs
4. session_start();
5.
6. $DB_con = new PDO("sqlite:VolunteerDB.sqlite3");
7.
8. // Collect inputed email and password from login page
9. $email = $_POST["email"];
10.     $password = $_POST["password"];
11.
12.     // SQL statment to find user
13.     $select_statement = $DB_con->query("SELECT VolunteerID,
   Email FROM Volunteers WHERE Email='$email' AND
   Password='$password'");
14.
15.     // Results returned as array
16.     $results = $select_statement->fetchAll(PDO::FETCH_ASSOC);
17.
18.     if ($results[0]["Email"] == $email) {
19.
20.         $_SESSION["logged_in"] = TRUE;
21.         $_SESSION["VolunteerID"] =
   $results[0]["VolunteerID"];
22.
23.         header("Location: Settings Page.php");
24.     } else {
25.         header("Location: Login Page.php");
26.     };
27.
28.     ?>
```

Line 6: (MainlyWebStuff, Connecting to a SQLite database with PHP, 2024)

APPENDIX 4: SETTINGS PAGE.PHP

```
1. <?php
2.
3. session_start();
4.
5. // Ensure user has already logged in
6. if ($_SESSION["logged_in"] == TRUE) {
7.     $DB_con = new PDO("sqlite:VolunteerDB.sqlite3");
8.
9.     $ID = $_SESSION["VolunteerID"];
10.
11.         $find_user_statement = $DB_con->query("SELECT
   Volunteers.Forename, Volunteers.Surname, Volunteers.Email,
   Volunteers.Password, Volunteers.MobileNumber,
   Volunteers.Address, Volunteers.Postcode,
```

```
    PreferredParticipation.ParticipationName FROM Volunteers,
    PreferredParticipation, ParticipationAssignment WHERE
    Volunteers.VolunteerID=ParticipationAssignment.VolunteerID AND
    ParticipationAssignment.ParticipationID=PreferredParticipation
    .ParticipationID AND Volunteers.VolunteerID=$ID");
12.
13.         $volunteer_results =
    $find_user_statement->fetchAll(PDO::FETCH_ASSOC);
14.
15.         $volunteer_details = $volunteer_results[0];
16.
17.         $forename = $volunteer_details["Forename"];
18.         $surname = $volunteer_details["Surname"];
19.         $email = $volunteer_details["Email"];
20.         $password = $volunteer_details["Password"];
21.         $phone_number = $volunteer_details["MobileNumber"];
22.         $address = $volunteer_details["Address"];
23.         $postcode = $volunteer_details["Postcode"];
24.
25.         $num_of_participation = count($volunteer_results);
26.         $selected_PP =
    array($volunteer_details["ParticipationName"]);
27.
28.         for ($count=1; $count<$num_of_participation;
    $count++) {
29.             array_push($selected_PP,
    $volunteer_results[$count]["ParticipationName"]);
30.         };
31.
32.         $find_PP_statement = $DB_con->query("SELECT
    ParticipationName FROM PreferredParticipation");
33.
34.         $PP_results =
    $find_PP_statement->fetchAll(PDO::FETCH_ASSOC);
35.
36.         $PP = array();
37.
38.         for ($count=0; $count<count($PP_results); $count++) {
39.             array_push($PP,
    $PP_results[$count]["ParticipationName"]);
40.         };
41.
42.     } else {
43.         header("Location: Login Page.php");
44.     };
45.
46.     ?>
47.     <html>
48.         <head>
49.             <title>Settings</title>
50.             <link rel="stylesheet" type="text/css"
    href="style.css">
51.         </head>
52.         <body>
53.             <div class="banner">
54.                 <a href="Sign Out.php">Sign Out</a>
55.                 <span>Lightwater Connected</span>
56.             </div>
57.             <div class="content_box">
```

```
58.                     <h1>Welcome <?php echo "$forename"?></h1>
59.                     <form action="Update Details.php"
   method="POST">
60.                         <label for="forename">Forename: <?php
   echo "$forename" ?></label>
61.                         <input type="text" id="forename"
   name="forename" pattern="([a-z]|[A-Z])+"><br><br>
62.                         <label for="surname">Surname: <?php echo
   "$surname" ?></label>
63.                         <input type="text" id="surname"
   name="surname" pattern="([a-z]|[A-Z])+"><br><br>
64.                         <label for="email">Email: <?php echo
   "$email" ?></label>
65.                         <input type="email" id="email"
   name="email"><br><br>
66.                         <label for="password">Password: <?php
   echo "$password" ?></label>
67.                         <input type="password" id="password"
   name="password" pattern=".{8,100}" title="Must be between 8
   and 100 characters"><br><br>
68.                         <label for="phone_number">Phone Number:
   0<?php echo "$phone_number" ?></label>
69.                         <input type="text" id="phone_number"
   name="phone_number" pattern="07[0-9].{8}" title="Enter a valid
   UK mobile number"><br><br>
70.                         <label for="address">Address: <?php echo
   "$address" ?></label>
71.                         <input type="text" id="address"
   name="address"><br><br>
72.                         <label for="postcode">Postcode: <?php
   echo "$postcode" ?></label>
73.                         <input type="text" id="postcode"
   name="postcode" pattern="([A-Z]|[A-Z][A-Z])([1-9][A-Z]?|[1-
   9][1-9]) [0-9][A-Z][A-Z]" title="Enter a valid postcode
   (including a space)"><br><br>
74.                         <?php
75.                         for ($count=0; $count<count($PP);
   $count++) {
76.                             $to_output = $PP[$count];
77.                             if (in_array($to_output,
   $selected_PP)) {
78.                                 echo "<label
   for='$to_output'><input type='checkbox' id='$to_output'
   name='$to_output' checked> $to_output</label>";
79.                             } else {
80.                                 echo "<label
   for='$to_output'><input type='checkbox' id='$to_output'
   name='$to_output'> $to_output</label>";
81.                             };
82.                         };
83.                         ?>
84.                         <input type="submit" value="Submit">
85.         </body>
86.     </html>
87.
88.     <?php
89.
90.     function phpAlert($msg) {
```

```
91.          echo '<script type="text/javascript">alert("' .
   $msg . '")</script>';
92.      };
93.
94.      if (isset($_SESSION["entered_PP"])) {
95.          if ($_SESSION["entered_PP"] == FALSE) {
96.              phpAlert("Please select atleast one preferred
   participation option");
97.          };
98.      };
99.
100.     $_SESSION["entered_PP"] = TRUE;
101.
102.     ?>
```

Line 7: (MainlyWebStuff, Connecting to a SQLite database with PHP, 2024)

Line 25/ 38/ 75: (W3Schools, 2024)

Line 28/ 38/ 75: (W3Schools, 2024)

Line 94: (W3Schools, 2024)

APPENDIX 5: UPDATE DETAILS.PHP

```
1. <?php
2.
3. session_start();
4.
5. // Ensure user has entered preferred participation
6. if (count($_POST) < 8) {
7.      $_SESSION["entered_PP"] = FALSE;
8.      header("Location: Settings Page.php");
9.      exit;
10.       };
11.
12.       // Creates PDO object telling it it's and sqlite db at
   such location
13.       $DB_con = new PDO("sqlite:VolunteerDB.sqlite3");
14.
15.       // Retrieving values entered by user on
16.       $ID = $_SESSION["VolunteerID"];
17.       $forename = $_POST["forename"];
18.       $surname = $_POST["surname"];
19.       $email = $_POST["email"];
20.       $password = $_POST["password"];
21.       $phone_number = $_POST["phone_number"];
22.       $address = $_POST["address"];
23.       $postcode = $_POST["postcode"];
24.
25.       // Finding users current details
26.       $find_user_statement = $DB_con->query("SELECT Forename,
   Surname, Email, Password, MobileNumber, Address, Postcode FROM
   Volunteers WHERE VolunteerID=$ID");
27.
28.       $user_results =
   $find_user_statement->fetchAll(PDO::FETCH_ASSOC);
```

```
29.
30.      // Ensuring no values are left empty
31.      if ($forename == "") {
32.          $forename = $user_results[0]["Forename"];
33.      };
34.      if ($surname == "") {
35.          $surname = $user_results[0]["Surname"];
36.      };
37.      if ($email == "") {
38.          $email = $user_results[0]["Email"];
39.      };
40.      if ($password == "") {
41.          $password = $user_results[0]["Password"];
42.      };
43.      if ($phone_number == "") {
44.          $phone_number = $user_results[0]["MobileNumber"];
45.      };
46.      if ($address == "") {
47.          $address = $user_results[0]["Address"];
48.      };
49.      if ($postcode == "") {
50.          $postcode = $user_results[0]["Postcode"];
51.      };
52.
53.      // Append all sellected preferred participation to an
   array
54.      $selected_PP = array();
55.
56.      // Select all preferred participation from DB
57.      $PP_results = $DB_con->query("SELECT ParticipationName,
   ParticipationID FROM
   PreferredParticipation")->fetchAll(PDO::FETCH_ASSOC);
58.
59.      // Append all selected preferred participation into list
60.      for ($count=0; $count<count($PP_results); $count++) {
61.          $PP = $PP_results[$count]["ParticipationName"];
62.          $PP_ID = $PP_results[$count]["ParticipationID"];
63.          $PP_underscore = str_replace(" ", "_", $PP);
64.          if (array_key_exists($PP_underscore, $_POST)) {
65.              array_push($selected_PP, $PP_ID);
66.          };
67.      };
68.
69.
70.      // Delete current Details
71.      $delete_PP_statement = $DB_con->prepare("DELETE FROM
   ParticipationAssignment WHERE VolunteerID=?");
72.      $delete_PP_statement->execute([$ID]);
73.      $delete_volunteer_statement = $DB_con->prepare("DELETE
   FROM Volunteers WHERE VolunteerID=?");
74.      $delete_volunteer_statement->execute([$ID]);
75.
76.      // Check if email or phone number already in use
77.      $check_statement = $DB_con->query("SELECT * FROM
   Volunteers WHERE Email='$email' OR
   MobileNumber='$phone_number'");
78.
79.      $check_results =
   $check_statement->fetchAll(PDO::FETCH_ASSOC);
```

```
80.
81.     // Ensures a result wasn't returned
82.     if (count($check_results) == 0) {
83.
84.         // Insert statement prepared
85.         $prepared_statement = $DB_con->prepare("INSERT INTO
    Volunteers (VolunteerID, Forename, Surname, Email, Password,
    MobileNumber, Address, Postcode) VALUES
    (?, ?, ?, ?, ?, ?, ?, ?)");
86.
87.         // Values added into insert statement
88.         // Insert statement executed returns true if
    completed and false if error
89.         $completed = $prepared_statement->execute([$ID,
    $forename, $surname, $email, $password, $phone_number,
    $address, $postcode]);
90.
91.         for ($count=0; $count<count($selected_PP); $count++)
    {
92.             // Preparing the insert statement for the
    selected Preferred Participation
93.             $PP_prepared_statement = $DB_con->prepare("INSERT
    INTO ParticipationAssignment (VolunteerID, ParticipationID)
    VALUES (?, ?)");
94.             //Executing the statement
95.             $PP_prepared_statement->execute([$ID,
    $selected_PP[$count]]);
96.         };
97.     } else {
98.         // Resetting the email and mobile number
99.         $email = $user_results[0]["Email"];
100.        $phone_number = $user_results[0]["MobileNumber"];
101.
102.        // Insert statement prepared
103.        $prepared_statement = $DB_con->prepare("INSERT INTO
    Volunteers (VolunteerID, Forename, Surname, Email, Password,
    MobileNumber, Address, Postcode) VALUES
    (?, ?, ?, ?, ?, ?, ?, ?)");
104.
105.        // Values added into insert statement
106.        // Insert statement executed returns true if
    completed and false if error
107.        $completed = $prepared_statement->execute([$ID,
    $forename, $surname, $email, $password, $phone_number,
    $address, $postcode]);
108.
109.        for ($count=0; $count<count($selected_PP); $count++)
    {
110.            // Preparing the insert statement for the
    selected Preferred Participation
111.            $PP_prepared_statement = $DB_con->prepare("INSERT
    INTO ParticipationAssignment (VolunteerID, ParticipationID)
    VALUES (?, ?)");
112.            //Executing the statement
113.            $PP_prepared_statement->execute([$ID,
    $selected_PP[$count]]);
114.        };
115.    };
116.
```

```
    header("Location: Settings Page.php");

    ?>
```

Line 13: (MainlyWebStuff, Connecting to a SQLite database with PHP, 2024)

Line 85-89/ 93-95: (MainlyWebStuff, Creating an INSERT form to add records - PHP and SQLITE, 2024)

Line 6/ 60/ 82/ 91/ 109: (W3Schools, 2024)

Line 60/ 91/ 109: (W3Schools, 2024)

Line 63: (TutorialRepublic, 2024)

APPENDIX 6: SIGN UP PAGE.PHP

```
<?php

session_start();

$_SESSION["logged_in"] = FALSE;

// Creates PDO object telling it it's and sqlite db at such location
$DB_con = new PDO("sqlite:VolunteerDB.sqlite3");

    // Query the database
    $statement = $DB_con->query("SELECT ParticipationName FROM PreferredParticipation");

    // Retrieve what database returns
    $results = $statement->fetchAll(PDO::FETCH_ASSOC);

    // define preferred participation array
    $PP = array();

    for ($count=0; $count<count($results); $count++) {
        array_push($PP, $results[$count]["ParticipationName"]);
    };

    ?>

    <html>
        <head>
            <title>Sign Up</title>
            <link rel="stylesheet" type="text/css" href="style.css">
        </head>
        <body>
            <div class="banner">
                <a href="Login Page.php">Log In</a>
                <span>Lightwater Connected</span>
            </div>
            <div class="content_box">
                <h1>Sign Up</h1>
```

```
37.                     <form action="Sign Up.php" method="POST">
38.                         <label for="forename">Forename:
39.                         <input type="text" id="forename"
   name="forename" required pattern="([a-z]|[A-Z])+"></label><br>
40.                         <label for="surname">Surname:
41.                         <input type="text" id="surname"
   name="surname" required pattern="([a-z]|[A-Z])+"></label><br>
42.                         <label for="email">Email:
43.                         <input type="email" id="email"
   name="email" required></label><br>
44.                         <label for="password">Password:
45.                         <input type="password" id="password"
   name="password" required pattern=".{8,100}" title="Must be
   between 8 and 100 characters"></label><br>
46.                         <label for="phone_number">Phone Number:
47.                         <input type="text" id="phone_number"
   name="phone_number" required pattern="(07)[0-9]{9}"
   title="Enter a valid UK mobile number"></label><br>
48.                         <label for="address">Address:
49.                         <input type="text" id="address"
   name="address" required></label><br>
50.                         <label for="postcode">Postcode:
51.                         <input type="text" id="postcode"
   name="postcode" required pattern="([A-Z]|[A-Z][A-Z])([1-9][A-
   Z]?|[1-9][1-9]) [0-9][A-Z][A-Z]" title="Enter a valid postcode
   (including a space)"></label><br>
52.                         <h2>Preferred Participation</h2>
53.                         <label for="subtext">Please Select
   atleast one</label>
54.                         <?php
55.                         for ($count=0; $count<count($PP);
   $count++) {
56.                             $to_output = $PP[$count];
57.                             echo "<label for='$to_output'><input
   type='checkbox' id='$to_output' name='$to_output'>
   $to_output</label>";
58.                         };
59.                         ?>
60.                         <input type="submit" value="Submit">
61.                     </form>
62.                 </div>
63.             </body>
64.         </html>
65.
66.         <?php
67.
68.         function phpAlert($msg) {
69.             echo '<script type="text/javascript">alert("' .
   $msg . '")</script>';
70.         };
71.
72.         if (isset($_SESSION["entered_PP"])) {
73.             if ($_SESSION["entered_PP"] == FALSE) {
74.                 phpAlert("Please select atleast one preferred
   participation option");
75.             };
76.         };
77.
78.         $_SESSION["entered_PP"] = TRUE;
```

```
79.
80.     ?>
```

Line 68-74: (Prakash_d22, 2024)

Line 19/ 55: (W3Schools, 2024)

Line 55: (W3Schools, 2024)

Line 72: (W3Schools, 2024)

APPENDIX 7: SIGN UP.PHP

```
1. <?php
2.
3. session_start();
4.
5. // Ensure user has entered preferred participation
6. if (count($_POST) < 8) {
7.     $_SESSION["entered_PP"] = FALSE;
8.     header("Location: Sign Up Page.php");
9.     exit;
10.      };
11.
12.      // Creates PDO object telling it it's and sqlite db at
   such location
13.      $DB_con = new PDO("sqlite:VolunteerDB.sqlite3");
14.
15.      // Retrieving values entered by user on
16.      $forename = $_POST["forename"];
17.      $surname = $_POST["surname"];
18.      $email = $_POST["email"];
19.      $password = $_POST["password"];
20.      $phone_number = $_POST["phone_number"];
21.      $address = $_POST["address"];
22.      $postcode = $_POST["postcode"];
23.
24.      // Append all sellected preferred participation to an
   array
25.      $selected_PP = array();
26.
27.      // Select all preferred participation from DB
28.      $PP_results = $DB_con->query("SELECT ParticipationName,
   ParticipationID FROM
   PreferredParticipation")->fetchAll(PDO::FETCH_ASSOC);
29.
30.      // Append all selected preferred participation into list
31.      for ($count=0; $count<count($PP_results); $count++) {
32.          $PP = $PP_results[$count]["ParticipationName"];
33.          $PP_ID = $PP_results[$count]["ParticipationID"];
34.          $PP_underscore = str_replace(" ", "_", $PP);
35.          if (array_key_exists($PP_underscore, $_POST)) {
36.              array_push($selected_PP, $PP_ID);
37.          };
38.      };
39.
40.      // Check if email or phone number already in use
```

```
41.     $check_statement = $DB_con->query("SELECT * FROM
    Volunteers WHERE Email='$email' OR
    MobileNumber='$phone_number'");
42.
43.     $check_results =
    $check_statement->fetchAll(PDO::FETCH_ASSOC);
44.
45.     // Ensures a result wasn't returned
46.     if (count($check_results) == 0) {
47.
48.         // SQL statement to find max ID
49.         $find_max_statement = $DB_con->query("SELECT
    MAX(VolunteerID) FROM Volunteers");
50.
51.         // Store results as a 2D array
52.         $results =
    $find_max_statement->fetchAll(PDO::FETCH_ASSOC);
53.
54.         // Finds needed value in 2D array
55.         $max_ID = $results[0]["MAX(VolunteerID)"];
56.
57.         $new_ID = $max_ID + 1;
58.
59.         // Insert statement prepared
60.         $prepared_statement = $DB_con->prepare("INSERT INTO
    Volunteers (VolunteerID, Forename, Surname, Email, Password,
    MobileNumber, Address, Postcode) VALUES
    (?, ?, ?, ?, ?, ?, ?, ?)");
61.
62.         // Values added into insert statement
63.         // Insert statement executed returns true if
    completed and false if error
64.         $completed = $prepared_statement->execute([$new_ID,
    $forename, $surname, $email, $password, $phone_number,
    $address, $postcode]);
65.
66.         for ($count=0; $count<count($selected_PP); $count++)
    {
67.             // Preparing the insert statement for the
    selected Preferred Participation
68.             $PP_prepared_statement = $DB_con->prepare("INSERT
    INTO ParticipationAssignment (VolunteerID, ParticipationID)
    VALUES (?, ?)");
69.             //Executing the statement
70.             $PP_prepared_statement->execute([$new_ID,
    $selected_PP[$count]]);
71.         };
72.
73.         // Sends user to login page if completed and returns
    user to sign up page if not
74.         if($completed){
75.             header("Location: Login Page.php");
76.         } else{
77.             header("Location: Sign Up Page.php");
78.         };
79.         exit;
80.     } else {
81.         header("Location: Sign Up Page.php");
82.         exit;
```

```
83.     };
84.     ?>
```

Line 8: (MainlyWebStuff, Connecting to a SQLite database with PHP, 2024)

Line 6/ 31/ 46/ 66: (W3Schools, 2024)

Line 31/ 66: (W3Schools, 2024)

Line 34: (TutorialRepublic, 2024)

APPENDIX 8: SIGN OUT.PHP

```
1. <?php
2.
3. session_start();
4.
5. $_SESSION["logged_in"] = FALSE;
6.
7. header("Location: Login Page.php");
8.
9. ?>
```

APPENDIX 9: DATABASE INITIALISATION.PY

```
1. import os
2. import sqlite3
3. import random
4.
5. DBPath = os.path.join(os.path.dirname(__file__),
   "VolunteerDB.sqlite3") #Defining the pathway to the sqlite3
   file
6.
7. def CreateTables(cursor):
8.
9.     VolunteersCreation = """
10.      CREATE TABLE IF NOT EXISTS Volunteers(
11.      VolunteerID INTEGER NOT NULL,
12.      Forename VARCHAR(30) NOT NULL,
13.      Surname VARCHAR(30) NOT NULL,
14.      Email VARCHAR(150) NOT NULL,
15.      Password VARCHAR(100) NOT NULL,
16.      MobileNumber INTEGER NOT NULL,
17.      Address VARCHAR(150) NOT NULL,
18.      Postcode VARCHAR(7) NOT NULL,
19.      PRIMARY KEY (VolunteerID));
20.      """
21.
22.          EventsCreation = """
23.      CREATE TABLE IF NOT EXISTS Events(
24.      EventID INTEGER NOT NULL,
25.      EventName VARCHAR(50) NOT NULL,
26.      Date DATE NOT NULL,
27.      Time TIME NOT NULL,
28.      Address VARCHAR(150) NOT NULL,
```

```
29.      Postcode VARCHAR(7) NOT NULL,
30.      PRIMARY KEY (EventID));
31.      """
32.
33.          PrefferedParticipationCreation = """
34.      CREATE TABLE IF NOT EXISTS PreferredParticipation(
35.      ParticipationID INTEGER NOT NULL,
36.      ParticipationName VARCHAR(150) NOT NULL,
37.      ParticipationDescription VARCHAR(300),
38.      PRIMARY KEY (ParticipationID));
39.      """
40.
41.          EventAssignmentCreation = """
42.      CREATE TABLE IF NOT EXISTS EventAssignment(
43.      VolunteerID INTEGER NOT NULL,
44.      EventID INTEGER NOT NULL,
45.      FOREIGN KEY (VolunteerID) REFERENCES
   Volunteers(VolunteerID)
46.      FOREIGN KEY (EventID) REFERENCES Events(EventID)
47.      PRIMARY KEY (VolunteerID, EventID));
48.      """
49.
50.          ParticipationAssignment = """
51.      CREATE TABLE IF NOT EXISTS ParticipationAssignment(
52.      VolunteerID INTEGER NOT NULL,
53.      ParticipationID INTEGER NOT NULL,
54.      FOREIGN KEY (VolunteerID) REFERENCES
   Volunteers(VolunteerID)
55.      FOREIGN KEY (ParticipationID) REFERENCES
   PreferredParticipation(ParticipationID)
56.      PRIMARY KEY (VolunteerID, ParticipationID));
57.      """
58.
59.          cursor.execute(VolunteersCreation) #Executing every
   command to create all the tables
60.          cursor.execute(EventsCreation)
61.          cursor.execute(PrefferedParticipationCreation)
62.          cursor.execute(EventAssignmentCreation)
63.          cursor.execute(ParticipationAssignment)
64.
65.
66.      def add_people(n, cursor):
67.
68.          forenames = ["Sam", "Jack", "James", "Max", "Ben",
   "Graham", "Hugo", "Anthony", "Zorawar", "Karan", "Archie",
   "Noah", "George", "Harry", "Ollie", "Toby", "Mark", "Joe",
   "Caleb", "Ruth", "Charlotte", "Isey", "Issy", "Marco", "Will",
   "John", "Justin", "Sarah", "Caitlyn", "Owen", "Cole", "Lewis",
   "Hayden", "Billie", "Lauren", "Alex"]
69.          surnames = ["Brown", "Davies", "Evans", "Green",
   "Wilson", "Roberts", "Anderson", "Harrison", "Johnson",
   "Smith", "Thomas", "Hughes", "Robinson", "Armstrong",
   "Wright", "Gibson", "Brinkman", "Richell", "Aulakh", "Lock",
   "Perret", "Toon", "Deighton", "Naumov", "Parry", "Moffat",
   "Light", "Baker", "Travers", "Ward"]
70.          letters = list(map(chr, range(ord('A'), ord('Z')+1)))
71.
72.          for i in range (1, n+1):
73.
```

```
74.                 forename = forenames[random.randint(0,
    len(forenames)-1)]
75.                 surname = surnames[random.randint(0,
    len(surnames)-1)]
76.                 postcode = f"{letters[random.randint(0,
    len(letters)-1)]}{letters[random.randint(0, len(letters)-
    1)]}{random.randint(0, 9)}{random.randint(0, 9)}
    {random.randint(0, 9)}{letters[random.randint(0, len(letters)-
    1)]}{letters[random.randint(0, len(letters)-1)]}"
77.                 email =
    f"{forename[0:4]}{surname}@outlook.com"
78.                 password = f"{surname}{random.randint(0,
    9)}{random.randint(0, 9)}{random.randint(0, 9)}"
79.                 mobile_number = int(f"07{random.randint(0,
    9)}{random.randint(0, 9)}{random.randint(0,
    9)}{random.randint(0, 9)}{random.randint(0,
    9)}{random.randint(0, 9)}{random.randint(0,
    9)}{random.randint(0, 9)}")
80.
81.                 statement = f"""INSERT INTO Volunteers
    (VolunteerID, Forename, Surname, Email, Password,
    MobileNumber, Postcode, Address)
82.                 VALUES (
83.                 {i}, '{forename}', '{surname}', '{email}',
    '{password}', {mobile_number}, '{postcode}', 'Gordons School'
84.                 );"""
85.
86.                 cursor.execute(statement)
87.
88.     def add_PP(cursor):
89.
90.         pp_statement1 = """INSERT INTO PreferredParticipation
    VALUES (
91.         1,
92.         'Restoring Park Benches',
93.         'Volunteers will fix benches, ensure they are smooth,
    and apply finishing coats'
94.         )"""
95.
96.         pp_statement2 = """INSERT INTO PreferredParticipation
    VALUES (
97.         2,
98.         'Sweeping Leaves',
99.         'Volunteers will sweep leaves in certain streets and
    help make the local area look cleaner'
100.        )"""
101.
102.        pp_statement3 = """INSERT INTO PreferredParticipation
    VALUES (
103.        3,
104.        'Cleaning Graffiti',
105.        'Volunteers will scrub illegal graffiti off walls'
106.        )"""
107.
108.        cursor.execute(pp_statement1)
109.        cursor.execute(pp_statement2)
110.        cursor.execute(pp_statement3)
111.
112.    def AssignPP(cursor):
```

```
113.
114.         search_statement = "SELECT VolunteerID FROM
     Volunteers"
115.         cursor.execute(search_statement)
116.         for row in cursor.fetchall():
117.             PP = random.randint(1, 3)
118.             statement = f"INSERT INTO ParticipationAssignment
     VALUES ({row[0]}, {PP})"
119.             if random.randint(1, 5) == 5 and PP == 2:
120.                 statement2 = f"INSERT INTO
     ParticipationAssignment VALUES ({row[0]}, {PP-1})"
121.                 cursor.execute(statement2)
122.                 if random.randint(1, 3) == 3:
123.                     statement3 = f"INSERT INTO
     ParticipationAssignment VALUES ({row[0]}, 3)"
124.                     cursor.execute(statement3)
125.             if random.randint(1, 5) == 5 and PP == 3:
126.                 statement2 = f"INSERT INTO
     ParticipationAssignment VALUES ({row[0]}, {random.randint(1,
     2)})"
127.                 cursor.execute(statement2)
128.             cursor.execute(statement)
129.
130.     def add_events(cursor):
131.
132.         event_statement1 = """INSERT INTO Events VALUES (
133.         1,
134.         'Lightwater Village Fete 2022',
135.         '11/20/2022',
136.         '10:30',
137.         'All Saints Church',
138.         'GU185SJ'
139.         )"""
140.
141.         event_statement2 = """INSERT INTO Events VALUES (
142.         2,
143.         'Lightwater Village Fete 2023',
144.         '14/20/2023',
145.         '10:30',
146.         'All Saints Church',
147.         'GU185SJ'
148.         )"""
149.
150.         cursor.execute(event_statement1)
151.         cursor.execute(event_statement2)
152.
153.     def assign_events(cursor):
154.
155.         search_statement = "SELECT VolunteerID FROM
     Volunteers"
156.         cursor.execute(search_statement)
157.         for row in cursor.fetchall():
158.             if random.randint(0,3) > 1:
159.                 event = random.randint(1, 2)
160.                 statement = f"INSERT INTO EventAssignment
     VALUES ({row[0]}, {event})"
161.                 if random.randint(1, 5) > 2 and event == 2:
162.                     statement2 = f"INSERT INTO
     EventAssignment VALUES ({row[0]}, {event-1})"
```

```
163.                     cursor.execute(statement2)
164.                 cursor.execute(statement)
165.
166.     def destroy_db(cursor):
167.         cursor.execute("DROP TABLE ParticipationAssignment;")
168.         cursor.execute("DROP TABLE EventAssignment;")
169.         cursor.execute("DROP TABLE Events;")
170.         cursor.execute("DROP TABLE Volunteers;")
171.         cursor.execute("DROP TABLE PreferredParticipation;")
172.
173.     def PopulateDatabase(DBPath):
174.
175.         SQLiteConnection = sqlite3.connect(DBPath) #
   Connecting to the sql file
176.         cursor = SQLiteConnection.cursor() # Creating the
   cursor
177.
178.         try:
179.             destroy_db(cursor)
180.         except:
181.             pass
182.
183.         CreateTables(cursor)
184.
185.         add_people(10000, cursor)
186.
187.         add_PP(cursor)
188.
189.         AssignPP(cursor)
190.
191.         add_events(cursor)
192.
193.         assign_events(cursor)
194.
195.         SQLiteConnection.commit()
196.
197.         cursor.close()
198.         SQLiteConnection.close()
199.
200.     PopulateDatabase(DBPath)
```