**Post-Prototype Refinements**

**Third party information**

I enquired my classmates and Teachers at my local swim club to give their input on the system, based on their user experience. This included a **questionnaire** which was sent to both that was focused on each area and interface of the system, as well as a copy of the program. My classmates are familiar with my program and the Tkinter framework, so they will be able to give more technical insight on how to program my system, whereas the teachers will have no Tkinter experience so they will purely assess the user experience of the system. They have both given feedback on their thoughts to improve the system and I will implement their ideas accordingly.

**Feedback**

**Teachers**

Whilst not all teachers within the club viewed the system, they shared the same views. The system at it’s current state was giving a bad user experience and needed to be more catered to their needs.

Some of the suggestions included creating boxes/highlights around certain functions to be able to tell different functions apart from one another, as before the widgets were quite cluttered together which resulted in teachers having a hard time selecting things on the system and reading class information.

They also felt that **the buttons on the login screen could be made bigger** so that their **Pin** isn’t incorrectly put it. This would be easy to fix as all I would need to do is resize those buttons and centre them in the screen for easier access.

They liked the simplicity of the Register function within the system as it’s design was intuitive and easy to use.

Being able to see lessons in the future was a big improvement in their eyes and was the most approved feature on the system.

They would like to see a NOTES function implemented in the future for each swimmer, as it makes talking about swimmers easier, rather than passing on notes by word of mouth. To implement this I would use SQL queries to fetch all swimmers from the database and upon selection of a swimmer, it will populate their respective information into a series of entry and text widgets. This would also make editing swimmers easier.

Overall, the teachers were impressed by the system and gave mostly positive feedback about the system.

**Matthew Jordan**

Matthew has a lot of experience with the Tkinter framework, and so most of his suggestions will be towards the code of the system and additional features to change/add.

Setting up the program was no problem for Matthew and even suggested that I incorporate a built-in command which can quickly run my system through the Terminal in my IDLE. This proved very useful in the long run as it was directly accessing the program through using file directories instead of manually running the program through the IDLE.

Upon entering the Staff Select View (the 1st view of the system), Matthew pressed the BACK button which would normally go back to the previous view. This completely removed the UI from the system and locked him out of doing anything in the program. A simple fix for this was to add a length check to my system, so that if there was only 1 view in the program, it would show an error message.

Matthew thought the security of the login screen was sound and there were no major flaws. He did make a comment about the size of the buttons being impractical, which aligned with the teacher’s views on the Login Screen also.

In the **Class Select** screen Matthew noticed that a scrollbar was being created despite there not being enough classes, which added a useless widget to the system. I acknowledged this feedback but decided not to change the design as it would add to much complexity to the program than was necessary.

Matthew also made several comments on how the widgets were quite tightly packed on the screen and recommended padding them to make them easier to select.

In my database, Matthew pointed out that a table wasn’t being used fully and was being redundant in my system, so to change this I simply relocated certain attributes of that table into other tables in my system, to create a more efficient database.

When viewing staff in the system, Matthew suggested to rework the view, as I was creating a lot of buttons that would simply lead to another screen that would carry out the same functionality. To fix this, simply avoided the use of another screen and made editing teachers a lot easier by using a treeview from the Tkinter library that the manager could select teachers from.

Overall, Matthew felt like the system just needed improvements in terms of it’s design, but overall the security of the system was sound. Some user experiences could be improved.

**Alex McAvoy**

Alex found the program ran smoothly and couldn’t find any errors within the system. His user experience was described as “quite a good experience but I don’t understand the functionality of certain aspects of the system.”. Alex felt that certain parts of the system should explain what needs to be done and how certain button works. I have partly implemented his advice and changed the wording of certain functions to make it more intuitive. I will not be adding descriptions of how to carry out certain tasks on the system as teachers would be trained to use the system, and this would add more clutter on the screen, leading to a bad user experience.

Alex agreed that the layout Viewing Staff in the system could be better implemented, and also suggested to store the Staff Details in a treeview. He also suggested to change how the system adds staff. In the process of Adding Staff to the database, they are not assigned to any class, thus they are just stored in the database. To work around this, Alex wants the creating process of staff to include assigning them to a class, this way Staff would be used as soon as they are added to the system. With careful consideration, I have decided not to implement this into my system as it would involve changing the layout of my database which would add too much complexity to the system. For now, adding staff to a class will be implemented in a separate function rather than through the Staff View.

Alex also noticed that there is no Validation in the **CRUD** system (create, retrieve, update, delete), as he was able to add invalid information to the database e.g. An email with the @ symbol. This is a simple fix, I can make sure that before the SQL query is ran, that a series of validation checks are ran to ensure the data entered is valid.

After Alex ran through the rest of the system, he felt that overall is was structurally sound with few errors. His input and criticism was helpful to the development of the system.

**Implemented Feedback**

**Staff View**

A screenshot of a computer

Description automatically generatedAs suggested, I have added the staff details to a treeview, where the details of the teacher are displayed in entry widgets that can be edited and saved. The Pseudocode will be shown below.

This creates an easier user experience and is quite intuitive to use. I have also created boxes surrounding certain parts of the view which have a header that describes the contents of the box.

At its current state, the manager can **ADD** and **SAVE** staff to the database. The **CLEAR** function just allows for all the entry widgets to be empty so a new staff can be added.

**Pseudocode:**

connect to myDatabase as myDB

all\_staff = "SELECT ALL\_STAFF FROM Staff"

treeview(values = all\_staff)

treeview.grid()

WHEN item IN treeview IS SELECTED:

    populate all Entry widgets

    return

WHEN SAVE button IS PRESSED:

    new\_info = get all info FROM Entry widgets

    "UPDATE Staff SET {ALL\_INFO} AS {new\_info}"

    return

WHEN ADD button IS PRESSED:

    new\_staff\_info = get all info FROM Entry widgets

    "INSERT INTO Staff {new\_staff\_info}"

    return

WHEN CLEAR button IS PRESSED:

    empty all info FROM Entry widgets

    return

**Class View**

I have added the View Swimmer function into the system, following the prototype. This followed the same design plans as the View Staff function and so all that I needed to change was the SQL query that fetched details from the database.

How this system works is that it uses the TeacherID to get the teacher’s name and then the rest of the info is inserted into the treeview.

For adding classes to the treeview/database the code is more complex. The Pseudocode will be shown below for the basic process of how data is added/collected.

Saving Classes is just for updating any info of current Classes in the database. For example, I could change the **TeacherID to be 1** which would show **Dylan Branda** in the teacher box.

A screenshot of a computer

Description automatically generated

The **Pseudocode** for **Class View** is very **similar** to **Staff View**, which is why I am **reusing** certain aspects from the **Staff View** **Pseudocode**.

**Pseudocode:**

connect to myDatabase as myDB

all\_classes = "SELECT ALL\_STAFF FROM Staff"

treeview(values = all\_classes)

treeview.grid()

WHEN item IN treeview IS SELECTED:

    populate all ComboBoxes

    return

WHEN SAVE button IS PRESSED:

    new\_info = get all info FROM ComboBoxes

    "UPDATE Class SET {ALL\_INFO} AS {new\_info}"

    "UPDATE Lessons SET class\_id={class\_id FROM new\_info}"

    return

WHEN ADD button IS PRESSED:

    new\_class\_info = get all info FROM ComboBoxes

    "INSERT INTO Class {new\_class\_info}"

    new\_lesson\_info = get class\_id FROM the new class

    "INSERT INTO Lessons {new\_lesson\_info}"

    return

WHEN CLEAR button IS PRESSED:

    empty all info FROM ComboBoxes

    return

**Swimmer View**

I have not yet created the code for Swimmer view, however, I would use the same process for Class and Staff View to create a treeview which would display all the relevant swimmer info from the database. I would additionally add more functionalities, such as, NOTES function and a REPORT function into this view, so that teachers could write notes and reports about respective swimmers. This can be easily implemented into the system, and the Pseudocode for this will look the same as previously shown. I will show how I plan to design the NOTES and REPORTS functions.

**NOTES/REPORTS Pseudocode:**

connect to myDatabase as myDB

get swimmer\_id FROM treeview

swimmer\_notes = "SELECT Notes FROM Swimmers WHERE SwimmerID = {swimmer\_id}"

notes\_box = Text\_widget()

notes\_box.grid()

notes\_box.insert(swimmer\_notes)

swimmer\_report = "SELECT Report FROM Swimmers WHERE SwimmerID = {swimmer\_id}"

report\_button = Button\_widget(command = report\_view(swimmer\_report))

report\_button.grid()

**Staff Selection**

An error message has been implemented if the user tries to use the BACK button in the **Staff Select Screen.** This is a minor improvement, but prevents the user from being fully locked out of the program.

A screenshot of a computer error

Description automatically generated

**Pseudocode:**

back\_function():

    IF length of views == 1:

        messagebox.showerror("No view to go back to!")

    ELSE:

        continue