**Development Testing**

This chapter will contain the problems that I faced during the software development stage of this program and the actions I took to fix them. Some of these were minor solutions to complete redesigns of certain aspects of my system.

**Displaying multiple class registers**

On a given day there are at least 20 different classes going on. Because of this, up to 20 classes need to be displayed on a small screen. I originally had no fix for this problem and this is what my display looked like with only 14/20 classes.

**Initial View**

As more registers get added to the system, the more it pushes down the screen. As you can see from here, the **Other Functions** box is being slightly cut out from the frame.

After some research I found that a scrollable Tkinter canvas would be best suited for this problem, as it would allow for multiple widgets to be stored and not move the **Other Functions** frame.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated**Finished View**

As shown to the left, I have created a Tkinter canvas which contains a frame that stores each row of **Class Registers.** Using previous programmers’ experiences with this problem online, I was able to incorporate code into my system and adapt it to fit my style of OOP.

After creating the ScrollableFrame, I simply run an iterative loop which grids each register and it’s respective class information.

A computer screen with text and images

Description automatically generated**Creating the frame and canvas in a separate file:**

**Gridding registers and class info inside the frame: (gridding info is highlighted)**

A screen shot of a computer program

Description automatically generated

**Moving between days**

Initially, as test data I only created 2 classes per day to see if my frames would get populated with the respective class info. However, when switching to a different day, the classes would generate outside of the frame, which would lead to multiple days being displayed at the same time.

A screenshot of a computer

Description automatically generated**Initial View (Selecting Monday twice):**

When selecting any day, I would call a function that grids the registers for that selected day. After looking around for a fix for this problem I found that you could use ‘.destroy()’ to delete the frame entirely, so that when I would call my function it would simply replace the previous register.

To be able to reuse my code I created a separate function that I would call which destroys my frames.

A screen shot of a computer screen

Description automatically generated**Destroy frames function:**

Any frames that I would display in **Class Select** would go in here to be destroyed. The frame which stores the Registers is called ‘**reg\_frame’**.

A screenshot of a computer

Description automatically generated**Finished View**

The system automatically loads up **Monday.** So it is quite difficult to show that there is a change but. Instead of gridding the registers for **Tuesday** below **Mondays** registers, it deletes them and re-grids **Tuesday** as shown.

**Keeping track of views (updating header)**

A simple functionality in my system to allow for a more friendly user experience was to keep track of the view you are in. Since most views in the system look quite similar and there are a lot of ways to navigate through the system, it made sense to add a header which detailed the location of the user within the system. This was more challenging than originally anticipated, and it required me to create a file which stored the **Frame** of the view that was being accessed at that moment by the user.

**Header code (final version):**

A computer screen with text and images

Description automatically generated

How this header class works is that in every **view class** a ‘**view\_name**’ is created which contains the chosen name of the view i.e. register class will have **view\_name** = “Registers”.

Each view class is then given access to the Header class and whenever the class is accessed, a function is called which passes the **view\_name** into ‘**update\_header’** and changes the name by setting the text of a Label widget inside the header frame to the chosen name.

Looking back on how I approached this problem, I didn’t do much research when trying to learn about how Tkinter Label widgets worked, specifically how to change the text of existing labels, which led to many errors i.e. overlapping header names

A computer screen shot of text

Description automatically generated

Originally I just gridded the header again thinking it would change the text, however this was not the case. After going through every view in my system and back to the staff selection this is what my header looked like.

**Old header display**

Now the header is fixed and correctly shows what view I am in.

A screenshot of a computer

Description automatically generated

**Updating header on exiting a view**

My header function was not able to update itself when going back to a previous view, only going into a new one, and so I had to implement additional functions into my BACK button code.

A screen shot of a computer program

Description automatically generated

On pressing the BACK button, it would now call the **on\_exit()** function within the Header class.

**On exit function:**

By keeping track of all my view\_names and appending it to my array head\_list, I could simply .pop() the last value in the array and display the newest one.

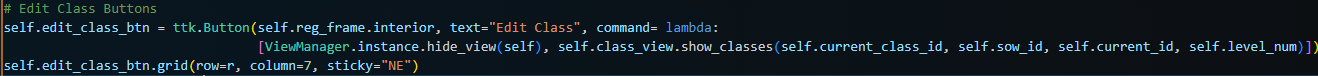
A screenshot of a computer program

Description automatically generated

**Updating SOW**

There is the option to **Edit Class** for each class in **Class Select** view. For me to do this, I would have to pass through every bit of information on that class into my **edit\_class\_btn.** Using the built in function **lambda** made this process easier, as it allows me to call multiple **methods** in my program and **pass in many parameters** into said **methods.** At first, I passed in every value as shown below.

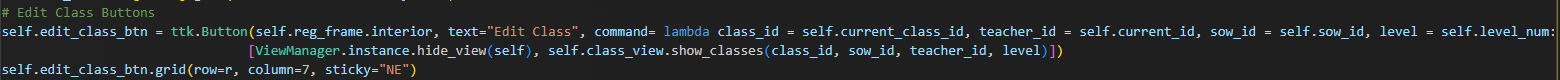
**Original Code**



The problem with this, was that since I was using a loop to generate each button, only the last fetched values from the database where assigned to every button, which resulted in every **edit\_class\_btn** containing the same information.

To work around this, I looked more into the abilities of lambda, and learned that you can assign values to each button through lambda as shown below.

**Adapted Code**



This allowed for each **edit\_class\_btn** to contain information unique to each class and to display the proper class information. I have since implemented this method in each of my classes.

**All Classes View**

Creating the view which displayed all classes was challenging, as I had to get data from different tables within my database to display in a treeview. Dealing with raw data like this can get quite confusing, as I had to store **all data** from the **Classes table** into a variable and reference this data through indexing to get the appropriate information needed to display into my treeview.

After getting all the correct data, I was encountered with a problem where an additional column was being created, which caused the order in which my class info was displayed to be incorrect

**Old class view**

A screenshot of a computer

Description automatically generated

After some thorough research, I found that there was nothing wrong with my code, but that Tkinter treeviews had a specified column name for the 1st column, that being “#0”.

I wasn’t referencing this, and so it made sense now that my columns weren’t aligned with the correct data.

A screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generated**Old class view code New class view code**

From that point on, I had to reference my **class\_id** column as **#0** instead, to adhere to the syntax of Tkinter treeviews.

A screenshot of a computer

Description automatically generated**New class view**

If I could do this again, I would have spent more time researching about Tkinter treeviews and their syntax, instead of just going in and trying to use treeviews.

This hindsight costed me time that could have been spent on developing my system further.