**Testing**

**Purpose of Testing**

Testing a system allows the programmer to make sure that the user requirements outlined in the **investigation** are met. This also makes sure that there are no errors in the program that would have not been noticed or disregarded during the development process. This testing would involve the use of various types of data, which will simulate the interaction between a user and the system.

I plan on using 3 types of testing.

* **Functional Testing** – This tests **Single methods** within the system, such as Logging in or Marking a Swimmer.
* **System Testing** – This test simulates the managers interaction with the system, who will interact with all aspects of the system. This includes Multi-method functions within the program, such as if a Swimmer has passed their current level or Changing class information.
* **End user Testing** – This test simulates a member of Staff at the Swimming club using the system in their job. This would include Marking a swimmers presence, Marking their performance, checking the SOW (Scheme of Work).

Throughout all these tests I will be using a **variety of test data**.

This includes:

* **Normal Data** – Valid, Sensible data
* **Erroneous Data** – Invalid data that cannot be processed and should be rejected by the system
* **Extreme Data** – Valid data on the edge of being invalid (this can only be implemented into few functions within my system)
* **Null Data** – No data is inputted.

**Test 1 – Logging In**

**Test 1.1 – Valid Manager PIN**

**Expected Outcome**: The manager should be brought to the Class Select screen

**Input:** (entering the PIN shown below in the Manager Login, then clicking ENTER)

A screenshot of a computer

Description automatically generated

**Result: Pass**

A screenshot of a computer

Description automatically generated

When a valid Manager PIN is entered and the ENTER button is clicked, the frame changes to the Class Select displaying all classes for Monday. For testing purposes, the entry widget is showing the input values but since this is a major security risk, the final system will replace each input character with a **\*** symbol. Upon originally testing this, I was trying to find a way to convert \* into the input digits. However, what I didn’t realise was that Tkinter has a built-in

**Test 1.2 – Invalid Manager PIN**

Expected Outcome: The system should reject the PIN and throw up an error message. The Entry widget will also clear itself.

Input: A PIN which is not valid for any Manager within the system.

A screenshot of a computer

Description automatically generated

**Result: Pass**

A screenshot of a computer

Description automatically generated

The system recognises that the PIN is not within the system for Managers, and returns the above error message, afterwards, the Entry widget is cleared for a new PIN to be entered.

**Test 1.3 – No PIN input**

Expected Outcome: The system should reject the input as a Null value is not a valid PIN within the system. It would also throw up an error message which asks for an input.

Input: No PIN

A screenshot of a computer

Description automatically generated

**Result: Partial Pass**

A screenshot of a computer

Description automatically generated

The system rejects the Null value and throws up the following error message. This is not the expected error message however. To fix this, I would have to first run a presence check within my system before internally checking the database with the entered PIN. I will not implement this, as the system still works as intended.

**Test 1.4 – Entering a PIN less than 4 digits**

Expected Outcome: The system will reject the entered PIN and throw up an error message.

Input: 777

A screenshot of a computer

Description automatically generated

**Result: Partial Pass**

A screenshot of a computer

Description automatically generated

The system rejects the entered PIN as intended, however, the error message should really inform the user that the PIN must be 4 digits.

**Test 1.5 – Entering a PIN more than 5 digits**

Expected Outcome: The system will throw up an error message that will prevent the user from entering another digit. The cap will be 4 digits.

Input: 12345

A computer screen shot of a computer error

Description automatically generated

**Result: Pass**

The system stops me when I try to enter the fifth digit “5”, and gives me the following error message.

The same results will be had with both Teachers and Assistants within the system, as they share the same Login Screen and Validation within the system. For this purpose, I will not be showing the other login screens as these tests will be redundant.

**Test 2 – Class Select**

**Test 2.1 – Switching from Monday-Sunday**

Expected Outcome: The current classes shown inside of the frame for Monday will clear and will be populated with Sunday classes. The user will be notified of what day they are viewing.

Input: Currently on Monday (which is highlighted) and about to select Sunday.

A screenshot of a computer

Description automatically generated

**Result: Pass**

A screenshot of a computer

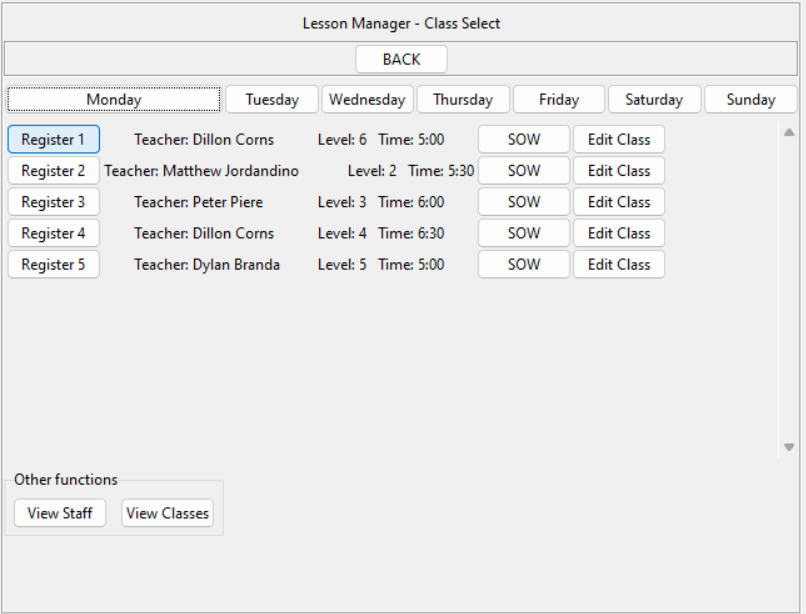
Description automatically generated

The classes for Monday are removed from the frame and the classes for Sunday are shown. The day selected is highlighted in blue as shown, however, a more obvious way of knowing what day is being displayed could be implemented.

**Test 2.2 – Opening a register inside of Monday classes**

Expected Outcome: The view will instantly change to the Register View and update the header to “Lesson Manager – Register”. This view will contain swimmers assigned to that class for that day and have an option to mark them present and to assess them.

Input: Register 1 will be selected in Monday



**Result: Pass**

A screen shot of a computer

Description automatically generated

The Header is updated and the respective swimmers are displayed for the given class as well as the Attendance and Assessment buttons are assigned to each swimmer.

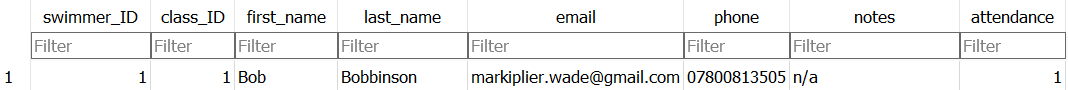
**Test 2.3 – Changing the Attendance of “Bob Bobbinson” to Absent**

Expected Outcome: The “Present” button will change colour and it’s text to red and “Absent” respectively, as well as update the attendance in the database to “0”.

A screen shot of a computer

Description automatically generated

Result: Partial Pass



Even though the button has changed state, the attendance for this swimmer has not been updated, and stays the same value of “1”.

**Test 2.4 – Assess “Bob Bobbinson” and pass the Level**

Expected Outcome: The “MARK” button will change the view to the Assessment view and display the relevant Swimmer information

**Test 2.5 – Opening the SOW for Register 1 on Monday**

Expected Outcome: The view will change to the SOW view and display the 4 boxes containing the information for the class. The header will update to “Lesson Manager – Scheme of Work”.

Input: SOW will be selected in Register 1 in Monday.

A screenshot of a computer

Description automatically generated

**Result: Pass**

A screenshot of a computer

Description automatically generated

The respective SOW information for that level of class is displayed in 4 listboxes. And will open a messagebox showing all the information when selected.

A screenshot of a computer

Description automatically generated

**Test 3 – View Classes**

**Test 3.1 – Removing Classes**

Expected Outcome: The class will be removed from the database and it will update the treeview and Class Select View.

Input: (class details shown below)

A screenshot of a computer

Description automatically generated

**Result: Fail**

A screenshot of a computer

Description automatically generated

The Class Select View containing the removed class has completely deleted itself and has created an error

A computer screen with white text

Description automatically generated

When trying to move to another Day aside from the one where the class has been deleted the program completely crashes and gives this message.

A computer screen shot of a black background

Description automatically generated

What is happening is that the Class itself is being deleted from the database, but not the lesson that contains it. This results in an empty array being passed into self.class\_info which is why an IndexError is being thrown here, since the index self.class\_info[0][1] doesn’t exist.

**Original Code**



To fix this, I plan on deleting the Lesson which contains the respective class\_ID as well as the Class itself as if I don’t, there will be a lesson in the database which is referencing a class\_ID which doesn’t exist and this will result in my program being permanently locked.

**Fixed Code**



**Test 3.2 – Adding Classes**

Expected Outcome: Add the class to the database and show a success message.

Input: Normal Data

A screenshot of a computer

Description automatically generated

**Result: Pass**

A screenshot of a computer error

Description automatically generatedA screenshot of a computer screen

Description automatically generated

Confirmation message is correctly shown, and the new class is added to the database assigned to the lesson as intended. A way I could improve upon this design would be implementing a **SORT** functionality into this treeview, so that I could sort data based on time, day, or level. Due to time constraints this wasn’t possible, however, this would majorly improve the user experience when adding classes to the system.

Another problem I was met with when testing this view was **deselecting** classes in the treeview. Currently, if the user 1st clicked on a class and then CLEARED the entry widgets, the selected class would still be referenced. This means that this class could be UPDATED or ADDED to the system. A simple approach for this problem would be to simply not allow any NULL data to be accepted into the system. This way if a class is selected, the entry widgets CLEARED, and then the UPDATE button is pressed, the last selected class wouldn’t change. This would be a significant improvement to the system that would overall create a better user experience.

Upon further research, I found a way to deselect items when an action was being performed i.e. **ADD**, **CLEAR**, **REMOVE**. I made a function which would be called at the end of each of these methods for reusability.

**New Code**

This iterates through every **selected** item in the treeview ‘**self.class\_list**’ and deselects them.

A screen shot of a computer code

Description automatically generated

**New class in database**

A screenshot of a table

Description automatically generated

**Test 3.3 – Adding Class with Invalid data**

**Expected Outcome:** An error message will be thrown, and the class will not be added to the database

**Input:** Erroneous Data

A screenshot of a computer

Description automatically generated

**Result: Fail**

The data gets successfully added to the class which causes the All Classes View screen gets completely wiped and thrown this error message.

A screenshot of a computer

Description automatically generated

**Error Message**A black background with white text

Description automatically generated

A white grid with numbers

Description automatically generatedThe class data is still added to the database so now the program will always crash when it runs unless the data is manually removed. This is a problem as none of the staff at the swimming club would have access to the SQL database and would result in downtime for the system.

The fix for this would be to implement a type of validation for the **All Classes View**, to prevent any sort of Erroneous data being accepted into the database. However, I realised that implementing all these methods of validation would not be possible given my time frame and as such I needed a simpler solution. Upon further research, I was able to find the syntax for ComboBoxes in Tkinter. By changing the state of all ComboBoxes to ‘readonly’ I could bypass all forms of validation, as the selected data would already be validated.

**Improved Code**

A screen shot of a computer program

Description automatically generated

Now, there is no way that a user can enter anything into the system asides from the given data.

**Test 4 – Viewing Staff**

**Test 4.1 – Adding Staff**

**Expected Outcome:** Staff will be added to the database and a success message will pop up

**Input:** Normal Data

A screenshot of a computer

Description automatically generated

**Result: Partial Pass**

A screenshot of a computer

Description automatically generated

Data has been added to database but no messagebox had been shown. I simply had forgotten to implement the messagebox function, so this is an easy fix.

**Test 4.2 – Adding Invalid Staff**

**Expected Outcome:** For the system to reject the data and output an error message

**Input:** Erroneous Data

A screenshot of a computer

Description automatically generated

**Result: Fail**

A screenshot of a computer

Description automatically generated

The data was still added to the database and did not show any form of error message. However, the **access\_level** was given a value of -1 instead of “pigeon” which is an unexpected outcome. Due to my system design this unintentionally protects the system as it only recognises staff members with **access\_level** values of **0, 1 OR 2**. Thismeant that the **All Staff** treeview isn’t updated, since the access\_level isn’t recognised by the system. This outcome made me realise that if I was dealing with staff, I should refer to them based on their **access\_level** to increase the security of my system.

I used the same approach for fixing the **All Staff View**, as I did with **All Classes View**, since their functionalities were so similar and made the state of the **Role** ComboBox ‘**readonly’.**

To add extra security to the system, I could validate each individual piece of data. For instance, I would implement a format check for the email, to make sure that there is at least one @ symbol in the email, however an incorrect email would be a great disadvantage to those who provided an incorrect email i.e. they wouldn’t get updates from the Manager, any urgent messages etc. So at this point, the validation is entrusted to the Manager.

Once change I would like to make is a length/presence check on the PIN of the teacher. At the current moment, a new teacher added to the system could have any length of a PIN and it would be accepted into the database. This causes several complications, such as:

* The user cannot enter the system, as the login screen only accepts a maximum of 4 digits.
* The user can have **NO PIN**, which would mean that anyone could accidentally access the system through their account.
* An invalid PIN could be entered, such as a string, which means that the user could not enter the system.

Additionally, the ability to remove staff from the system is not available, but this could be easily implemented given the project had more time for completion.

**Test 5 – Assessing Swimmers**

**Test 5.1 – Changing the mark**

**Expected Outcome:** The **Passed** column in the treeview will change from FAIL to PASS and vice versa on the selected Skill. This data will be saved in the treeview.

A screenshot of a computer

Description automatically generated**Input:** CHANGE MARK button is pressed for the 1st skill.

**Result: Partial Pass**

The mark updated for the selected skill in the treeview as expected, however, this was not saved upon leaving and entering the Assessment View again.

This is because all of the **Passed** values in the treeview are determined based on the state of the ‘**passed’** parameter in the database. The ‘**passed**’ field in the **Swimmers** table in the database is automatically set to 0 when a new swimmer is added. The only way the ‘**passed**’ field is updated is when the swimmer has passed in ALL skills.

A screen shot of a computer

Description automatically generated

To fix this issue I would have to create a new table within the database which would contain the swimmer\_ID and a field for every skill in the syllabus. The number of skills being assessed will always be 5 so this simplifies the process. When a skill is updated, the corresponding skill in this new table will be updated and saved. Unfortunately, due to time constraints, this won’t be implemented into the finished system. However, given more time, I would be able to implement this.

To work around this, I have put in place a temporary button called PASS ALL, which will allow the teacher to change all of the states of the **Passed** column in the treeview to PASS. Alongside this change, the PASS button will only work if ALL the values in the **Passed** column are set to PASS, instead of referencing the database for the updated ‘**pased**’ value. This will enable staff to use this functionality, allowing swimmers to progress to higher levels in the swimming club.

**Updated Assessment view**

A screenshot of a computer

Description automatically generated

**Test 5.2 – Passing a swimmer that PASSES in all skills**

**Expected outcome:** The swimmer will move on to a different screen where the user will select what class the swimmer will move to.

**Input:** **PASS** button is pressed when all skills have been set to **PASS**

A screenshot of a computer

Description automatically generated

**Result: Pass**

The user is successfully brought to a new screen where they can select a new class to move the swimmer to.

**New View**

A screenshot of a computer

Description automatically generated

A way to improve this would be providing the user with a messagebox that signals the success of the swimmer passing the level.

When working on this I came up with a new functionality to the system that I would implement if I had the given time. Upon passing the swimmer, the system would fetch the swimmer’s email and send an automated message to them, which would say that they passed and to what class they are moving to. This would further enhance the user experience within the system.

**Test 5.3 – Passing a swimmer that FAILS in all skills**

**Expected outcome:** The swimmer will no be able to move on and the system will give an error message.

**Input:** Erroneous data

A screenshot of a computer

Description automatically generated

**Result: Pass**

This was a successful test, the user is now informed on how to PASS the swimmer and can gain information about the system.

**Test 6 – Scheme of Work (SOW)**

**Test 6.1 – Checking SOW**

**Expected outcome:** The user will get a popup message which contains all the information about the selected part of the SOW.

**Input:** Selecting the intro

A screenshot of a computer

Description automatically generated

**Result: Pass**

This was an insightful test as it made me question the design of how the user views the SOW. Creating this, I made use of a messagebox to store the information without knowing of any alternative. Upon further research, I realised I could have used a Text widget created in another view, that would have more than enough space to display the SOW. However, due to time constraints I decided not to add this to the system. A plus side of using a message box, is that I saves me having to create another view to display the information and setting up links in my system. Additionally, on the off chance that the information in the SOW if bigger than the Text widget, the messagebox easily adapts and changes size to fit the contents of the SOW. My research did not go in vain however, as I used my knowledge of Tkinter Text widgets and implemented them into another part of my system.

**Test 6.2 – Checking a longer SOW**

**Expected outcome:** The **SOW** displayedin the listboxes willnot be able to contain the entirety of the information, but upon clicking the listbox, the messagebox will contain all the information needed.

**Input:** Selecting a part of the SOW that is cut off from view in the listbox.

A screenshot of a computer

Description automatically generated

**Result: Pass**

The messagebox easily displays the rest of the information that isn’t shown in the listbox which gives the user the rest of the information needed to plan for the class.

When originally designing this view, I had decided to show that information from the SOW was truncated by using an ellipsis (…). This proved to be more awkward to implement, as I had made the system cut off any data that was more than 35 characters long and replace it with an ellipsis to be inserted into the listbox. Unfortunately, due to the size of certain characters, it would not always display the ellipsis which resulted in me removing the feature.

In the future, I would have done more research about Tkinter listboxes before designing this view and made changes accordingly.

**Test 7 – Editing Classes**

**Test 7.1 – Editing a class and displaying the correct class information**

**Expected outcome:** The system will fetch the respective data for the class and display them in a new view i.e. **SOW**, **Swimmers**, **Level** and **Teacher**.

**Input:** The class from Register 1 is being Edited

A screenshot of a computer program

Description automatically generated

**Result: Pass**

A screenshot of a computer

Description automatically generated

The system has selected the correct class data and displayed it in a new view as expected.

As I had previously stated about this view in previous chapters, the overall design is quite closely packed together and could be implemented better. Some changes I would make to improve this view would include: a Label frame that contains the Swimmer Details to be changed, a Label frame which contains the class details to be changed i.e. Teacher and Level, creating a coloured border around the SOW and swimmer treeview to be able to distinguish between the 2. These changes will not be implemented into the final version of the system due to time constraints and that it isn’t vital for the functionality of the system.

**Test 7.2 – Updating Swimmer info**

**Expected outcome:** The selected swimmer will be updated in the treeview and in the database when the save button is pressed. The user will also be notified by a messagebox.

**Input:** Valid data

A screenshot of a computer

Description automatically generated

**Result: Partial Pass**

The new swimmer details are updated in the database, however the data in the treeview is only updated when the user leaves the view and enters again, which creates a bad user experience. This can be fixed by using previous treeview updated methods like in **All Classes** and **All Teacher** view, however, this isn’t vital for the overall functionality of the system, and so it will not be implemented.

**New swimmer details in database**

A close up of a contact us

Description automatically generated

**Test 7.2**

**Expected outcome:** For the data to not be accepted into the database and to throw up an error message.

**Input:** Erroneous data

A screenshot of a computer

Description automatically generated

**Result: Fail**

The database has accepted the invalid data and saved it in the database. This can be fixed by using a series of validation techniques such as: Format checks, presence checks and type checks, to prevent invalid data.

**Test 7.3 – Updating Teacher and Level**

**Expected outcome:** TheTeacher and Level with both automatically update in the database and provide a message box which informs the user of the changes made

**Input:** Selected data shown below

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

As an extra added layer of security to the system, I have made the Teacher and Level update independently from the SAVE button. The reason for this was to allow the user to be sure that the Teacher and Level were always the current Teacher and Level displayed in the ComboBoxes, which saved me having to implement an update functionality for the view.

In the messageboxes provided, it informs the user exactly what changes were made. To improve upon this, I might have liked to add exactly what current Teacher and Level was being changed, so there is more clarity for the user.

I also prevented the chance of having erroneous data being input for Teachers and Levels, by making the state of both ComboBoxes to be ‘**readonly’**.

Overall, I feel a very pleasant user experience for this functionality, with robust security.

**Result: Pass**

**Test 7.4 – Selecting SOW to edit**

**Expected outcome:** The user will be brought to a new view which will contain a Text widget that contains the current SOW.

**Input:** Selected **Main** section of the SOW

A screenshot of a computer

Description automatically generated

**Result: Pass**

The user is brought immediately to the Edit SOW view and the correct SOW information is being displayed. The Text widget also allows for the user to type anything they want into the SOW. There is no current validation for this, but a presence check would be smart to implement, to prevent Null data from entering the database.

**Test 7.5 – Updating SOW**

**Expected outcome:** The SOW can be edited and saved by pressing the SAVE button. Doing this will automatically bring the user back to the **Edit Class** view. A messagebox will be provided to inform the user of the success.

A screenshot of a computer

Description automatically generated**Input:** Additional SOW info

**Result: Partial Pass**

The SOW is updated in the database and the user is taken back to the Edit Class view as intended upon pressing **OK**.

To confirm the view changes back to **Edit Class** view, we can see the last selected thing in **Edit Class.**

A screenshot of a computer

Description automatically generated

Whilst the data in the database is updated, the data inside of the listbox isn’t updated. This could be fixed by simply refreshing the view i.e. removing it and adding it again.

Alongside this, an interesting result was found in the database. Whenever the SAVE was made for the **Main** section, it also added a new line. **SHOWN BELOW**

A screenshot of a computer

Description automatically generated

New line in the **main** section.

This is especially interesting because I am not creating a new line whenever I update the SOW, so the reason for this is still unknown. However, this presents various issues if left unfixed, namely this could lead to memory leakage in the long run of the program as unnecessary space is being taken up in the database.

To fix this issue I have printed out the data before it was added to the database to see exactly what was being passed. Here is the output:

A screenshot of a computer program

Description automatically generated

It isn’t so easy to see, but every time the data is being changed, a new line is added to the output. To work around this I simply added a .strip() to the end of the new\_data variable to prevent any white space from being added to the database.

**New Code**



**New database output**

A screenshot of a computer

Description automatically generated