Testing Document

# Manual Testing

# Graphical user interface, application Description automatically generatedTest Plan 1

|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 25/02/21 | Tested By: | Jack Davies |
| System: | Login Page | Environment: | Python |
| Objective: | Check if the system will login without entering any details or accepting the terms and conditions box. | Test ID: | 1 |
| Version: | 1 | Test Type: | Absent |
| Status: | In Process |  |  |
|  | | | |
| Type of Tests | Before | After | Result |
| * Invalid * Boundary * Valid * Wrong * Absent | The type of test that will occur will be an absent test. This test will include no input to see if the appropriate error will occur. | The type of test in the above screenshot is an absent test. As you can see, no data has been entered into the textboxes and the terms and conditions box has not been accepted. | As there was no data being entered and tick boxes have not been ticked, an error message has occurred to tell the user to check their input. This test was successful because we do not want our program to allow the user to login without the correct login details and without accepting the terms and conditions. We also do not want our program to crash either if an error occurs. |

# A computer screen with a program on it Description automatically generated with low confidenceTest Plan 2

|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 25/02/21 | Tested By: | Jack Davies |
| System: | Login Page | Environment: | Python |
| Objective: | Try to login without entering the correct login details | Test ID: | 2 |
| Version: | 1 | Test Type: | Invalid |
| Status: | In Process |  |  |
|  | | | |
| Type of Tests | Before | After | Result |
| * Invalid * Boundary * Valid * Wrong * Absent | The type of test that will occur will be an invalid test. This test will include inputting incorrect details to see if the system will login in or if the appropriate error will occur. | The type of test in the above screenshot is an invalid test. As you can see, the wrong details have been entered into the textboxes. However, the terms and conditions have been accepted which means the only error that should occur would be about incorrect login details. | An incorrect username and password have been entered, which created an error. This test was successful because the user should not be allowed to login with incorrect details. The correct error message appeared too, and the program did not crash. |

# Graphical user interface, application Description automatically generatedTest Plan 3

|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 25/02/21 | Tested By: | Jack Davies |
| System: | Login Page | Environment: | Python |
| Objective: | Try to login without accepting the terms and conditions. | Test ID: | 3 |
| Version: | 1 | Test Type: | Absent |
| Status: | In Process |  |  |
|  | | | |
| Type of Tests | Before | After | Result |
| * Invalid * Boundary * Valid * Wrong * Absent | The type of test that will occur will be an absent test. This test will try to login without accepting the terms and conditions. | The type of test in the above screenshot is an absent test. As you can see, the terms and conditions box has not been ticked which means the user should be able to login without accepting it. | This test was successful because the user did not accept the conditions and an error message appeared informing the user to accept the conditions if they wish to continue. |

# A picture containing text, screenshot, monitor, indoor Description automatically generatedTest Plan 4

|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 25/02/21 | Tested By: | Jack Davies |
| System: | Login Page | Environment: | Python |
| Objective: | Login by using correct usernames and passwords and by clicking the accept button for the terms and conditions. | Test ID: | 4 |
| Version: | 1 | Test Type: | Valid |
| Status: | In Process |  |  |
|  | | | |
| Type of Tests | Before | After | Result |
| * Invalid * Boundary * Valid * Wrong * Absent | The type of test that will occur will be a valid test. This test will try to login by using correct usernames and passwords and by clicking the accept button for the terms and conditions. | The type of test in the above screenshot is a valid test. As you can see, the terms and conditions box has been accepted and the data has been inputted for the username and password. | This test was successful because the user entered correct login details and they also accepted the conditions. No error message appeared, on a message to say welcome to the user. |

# Test Plan 5

A picture containing text, monitor, screenshot, indoor

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 25/02/21 | Tested By: | Jack Davies |
| System: | Login Page | Environment: | Python |
| Objective: | Login by inputting incorrect data in the usernames and passwords textboxes. But also accepting the button for the terms and conditions. | Test ID: | 5 |
| Version: | 1 | Test Type: | Invalid |
| Status: | In Process |  |  |
|  | | | |
| Type of Tests | Before | After | Result |
| * Invalid * Boundary * Valid * Wrong * Absent | The type of test that will occur will be an Invalid test. This test will try to login by incorrect usernames and passwords and by clicking the accept button for the terms and conditions. | The type of test in the above screenshot is an invalid test. As you can see, the terms and conditions box has been accepted and the data has been inputted for the username and password. However, they are incorrect details. | This test was successful because an error message has appeared informing the user that the only error that has occurred is an input error. Meaning that either the username or password is incorrect. |

Graphical user interface, text, application

Description automatically generated

# Test Plan 6

A computer screen capture

Description automatically generated with low confidence

|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 25/02/21 | Tested By: | Jack Davies |
| System: | Login Page | Environment: | Python |
| Objective: | No SQL injection security was implemented in our code. This caused a vulnerability which could cause a lot of damage if not corrected. | Test ID: | 6 |
| Version: | 1 | Test Type: | Boundary |
| Status: | In Process |  |  |
|  | | | |
| Type of Tests | Before | After | Result |
| * Invalid * Boundary * Valid * Wrong * Absent | The type of test that will occur will be a boundary test. No SQL injection was implemented. | The type of test in the above screenshot is a boundary test. The line that the error occurs is SELECT \* FROM Password WHERE ID = ? AND Password = ? % id, passw) | This test was unsuccessful. If the user enters a legitimate search value, for example, 1478, in the above statement then all fine. But if they try something untoward, for example, 1478'; DROP TABLE Password; then the results are fatal.  The statement passes ID and Password from the client directly to the database, without performing any sort of check or validation. This sort of code is perfect for inviting SQL injection.  Test plan 7 will show how we will correct this vulnerability. |

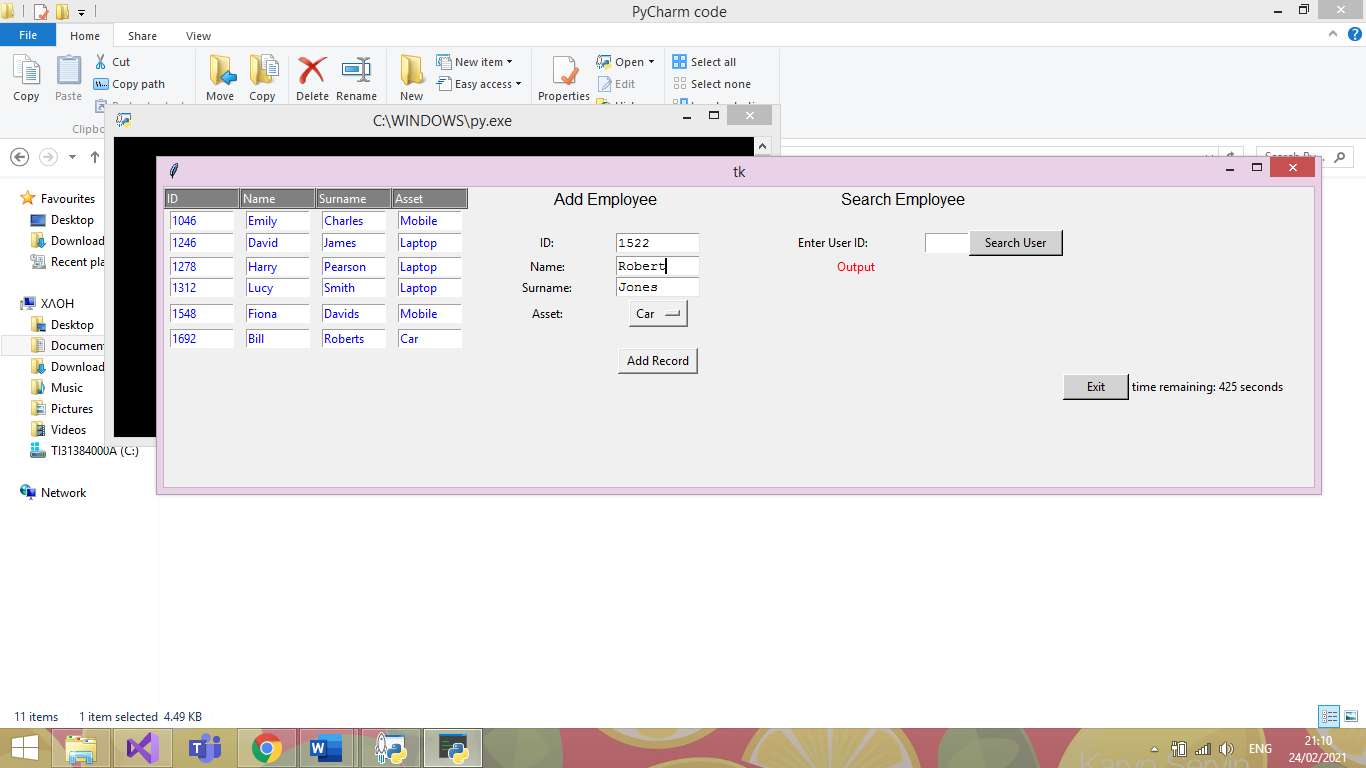
# Test Plan 7

A picture containing text, monitor, screenshot, indoor

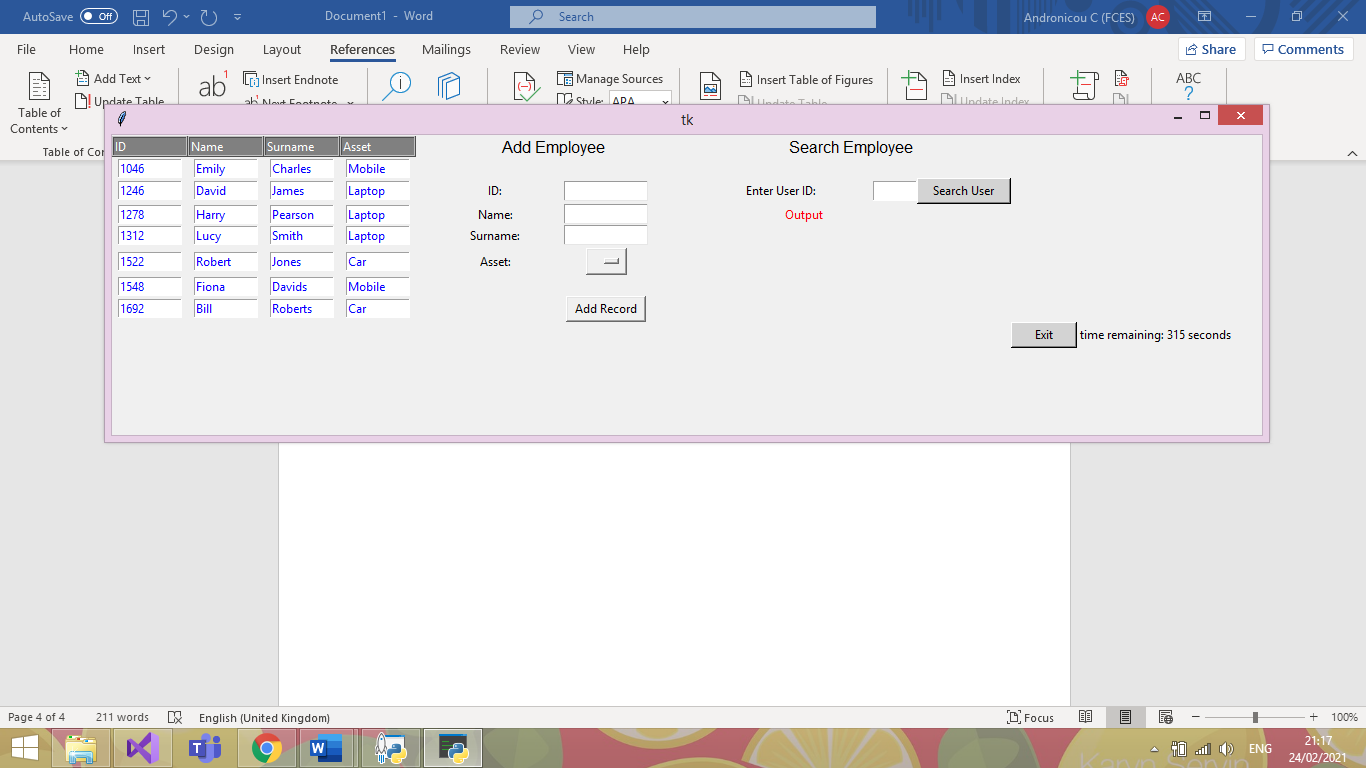
Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 25/02/21 | Tested By: | Jack Davies |
| System: | Login Page | Environment: | Python |
| Objective: | No SQL injection security was implemented in our code. This caused a vulnerability which could cause a lot of damage if not corrected. | Test ID: | 7 |
| Version: | 2 | Test Type: | Boundary |
| Status: | In Process |  |  |
|  | | | |
| Type of Tests | Before | After | Result |
| * Invalid * Boundary * Valid * Wrong * Absent | The type of test that will occur will be a boundary test. No SQL injection was implemented. | The type of test in the above screenshot is a boundary test. The line that the error occurs was SELECT \* FROM Password WHERE ID = ? AND Password = ? % id, passw)  but now we have changed it to  SELECT \* FROM Password WHERE ID = ? AND Password = ?, ( id, passw)) | This test was successful. In this statement, id and password are passed as named parameters. Now, the database will use the specified type and value of the parameters when executing the query, offering protection from Python SQL injection. |

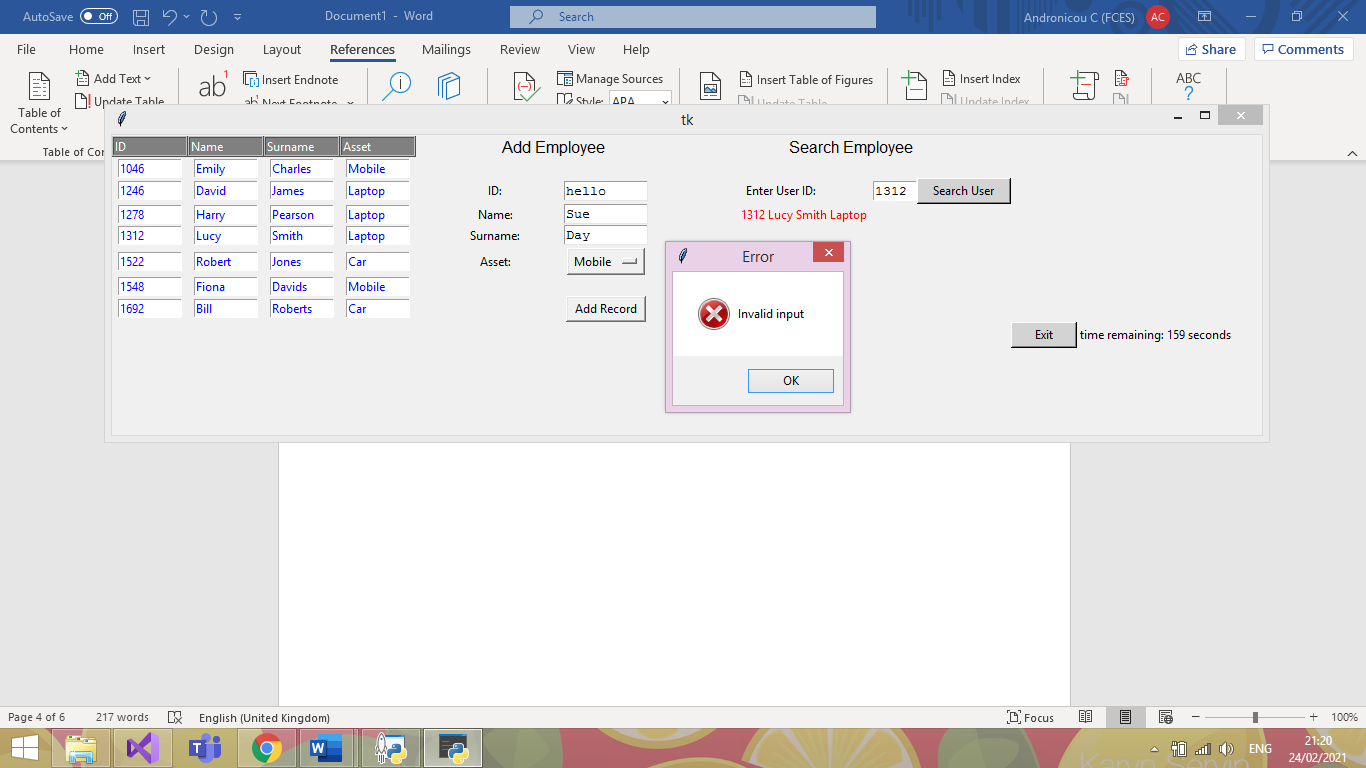
# Test Plan 8



|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 25/02/21 | Tested By: | Jenni Whewell |
| System: | Login Page | Environment: | Python |
| Objective: | Checking to see if details of ID, Name, Surname and Asset can be added to the database | Test ID: | 8 |
| Version: | 1 | Test Type: | Valid |
| Status: | In Process |  |  |
|  | | | |
| Type of Tests | Before | After | Result |
| * Invalid * Boundary * Valid * Wrong * Absent | The test above is to check whether user details can be added to the database with the correct values to the text boxes. | The information in the test above is to check if a user can add valid information. | From the image below, it is clearly indicated that the test was successful where the entry is seen with user ID 1522 in the table adjacent. |

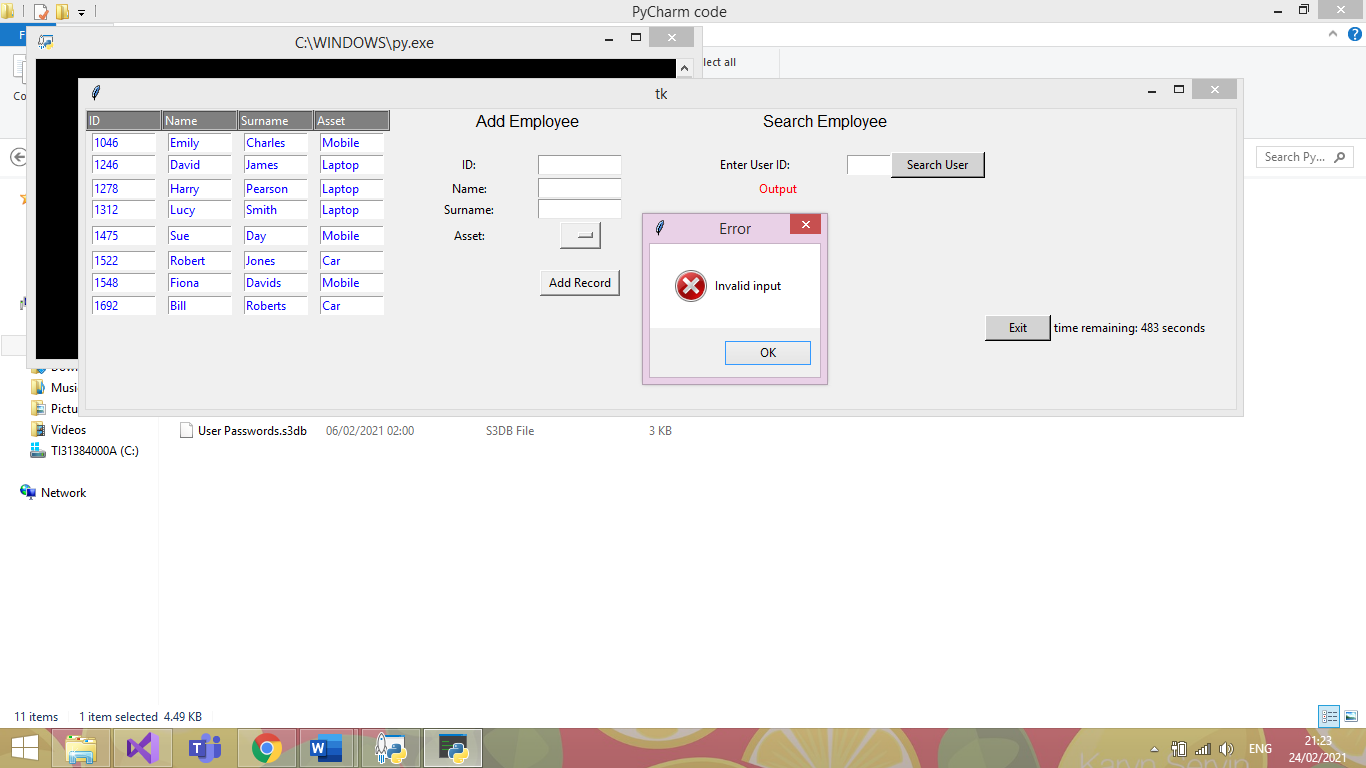


# Test Plan 9



|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 25/02/21 | Tested By: | Jenni Whewell |
| System: | Login Page | Environment: | Python |
| Objective: | Entering a string in the user ID text box. | Test ID: | 9 |
| Version: | 1 | Test Type: | Invalid |
| Status: | In Process |  |  |
|  | | | |
| Type of Tests | Before | After | Result |
| * Invalid * Boundary * Valid * Wrong * Absent | An invalid test is executed to see if it throws an error. This type of test is known as an invalid test.  For the test we are entering the word “hello” | The type of test in the above screenshot is an invalid test. As you can see, the first name and surname have all be entered correctly with just the user ID being inputted wrong. | This test was successful because an error message has appeared informing the user that the only error that has occurred is an input error. Meaning that the user ID has been entered incorrectly. |

# Test 10



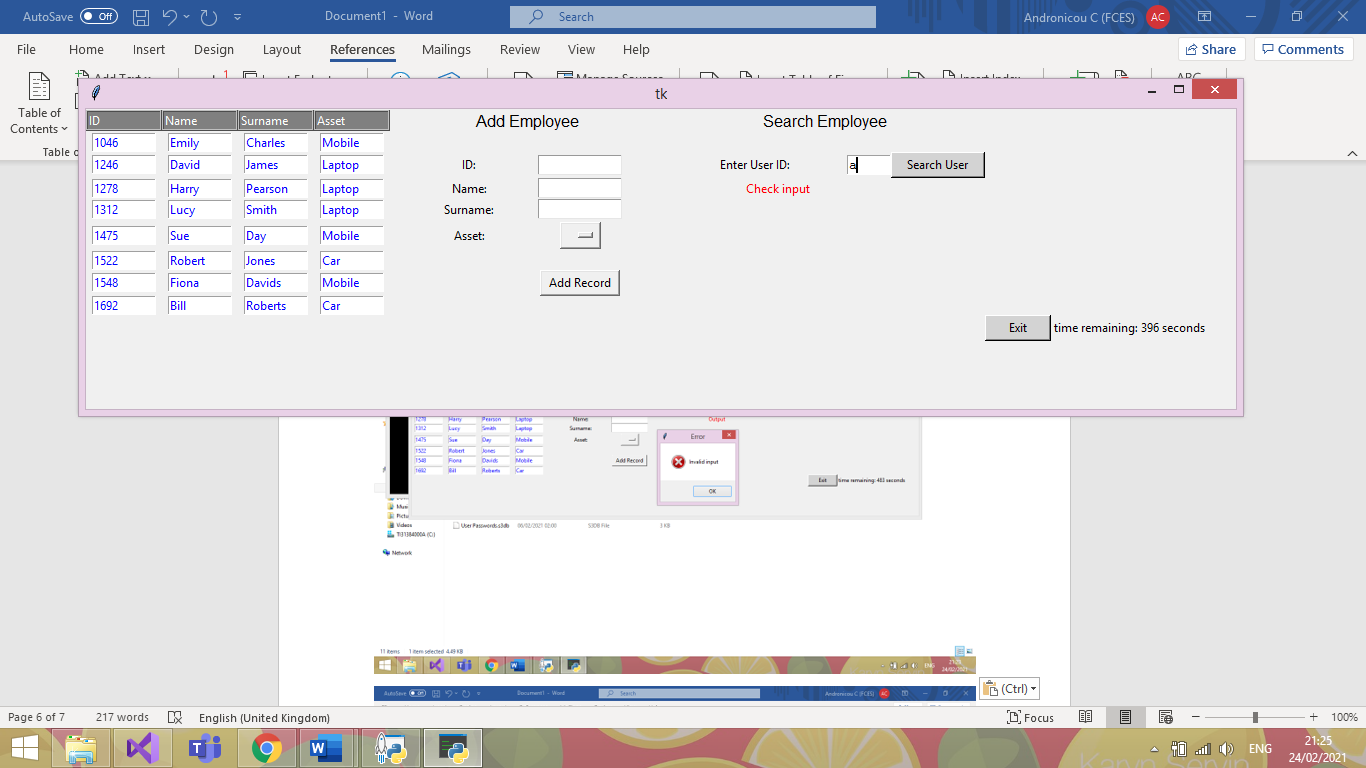
|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 25/02/21 | Tested By: | Jenni Whewell |
| System: | Login Page | Environment: | Python |
| Objective: | The user clicks add without entering any details. | Test ID: | 10 |
| Version: | 1 | Test Type: | Absent |
| Status: | In Process |  |  |
|  | | | |
| Type of Tests | Before | After | Result |
| * Invalid * Boundary * Valid * Wrong * Absent | An absent test is executed in the screenshot above. The purpose of the test is to check what happens if the text boxes are left empty. | An absent test is good to check so that there are no empty entries in the database. This means that the primary keys of the database will not except a NULL value. | This test was successful because an error message has appeared informing the user that the only error that has occurred is an input error. Meaning that information needs to be added to the text boxes for it to be a successful entry. |

# Test Plan 11

# 

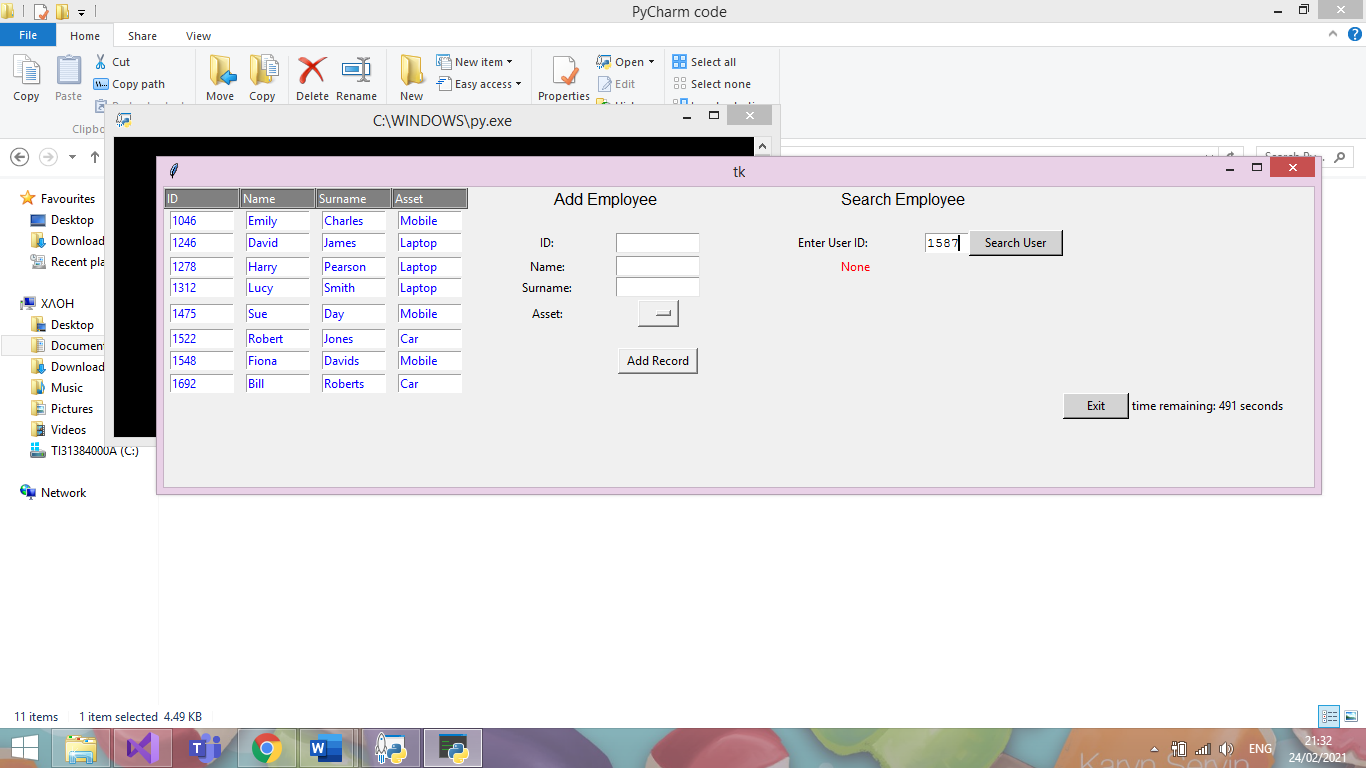
|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 25/02/21 | Tested By: | Jenni Whewell |
| System: | Login Page | Environment: | Python |
| Objective: | Searching for a user by their ID number. | Test ID: | 11 |
| Version: | 1 | Test Type: | Valid |
| Status: | In Process |  |  |
|  | | | |
| Type of Tests | Before | After | Result |
| * Invalid * Boundary * Valid * Wrong * Absent | In the first screenshot, the user ID 1312 is entered for the purpose of the search test. The purpose of a valid test is to check if the program is outputting the correct result when asking the system for details. | The details that should be outputted in the result of the test should include the first name, surname, and the asset that the employee has taken.  With the user ID 1312, the name should be Lucy smith and the asset should be a laptop. | This test was successful because all the correct details have appeared. This means that the form can take valid inputs from the system and it will output the correct details. |

# Test Plan 12



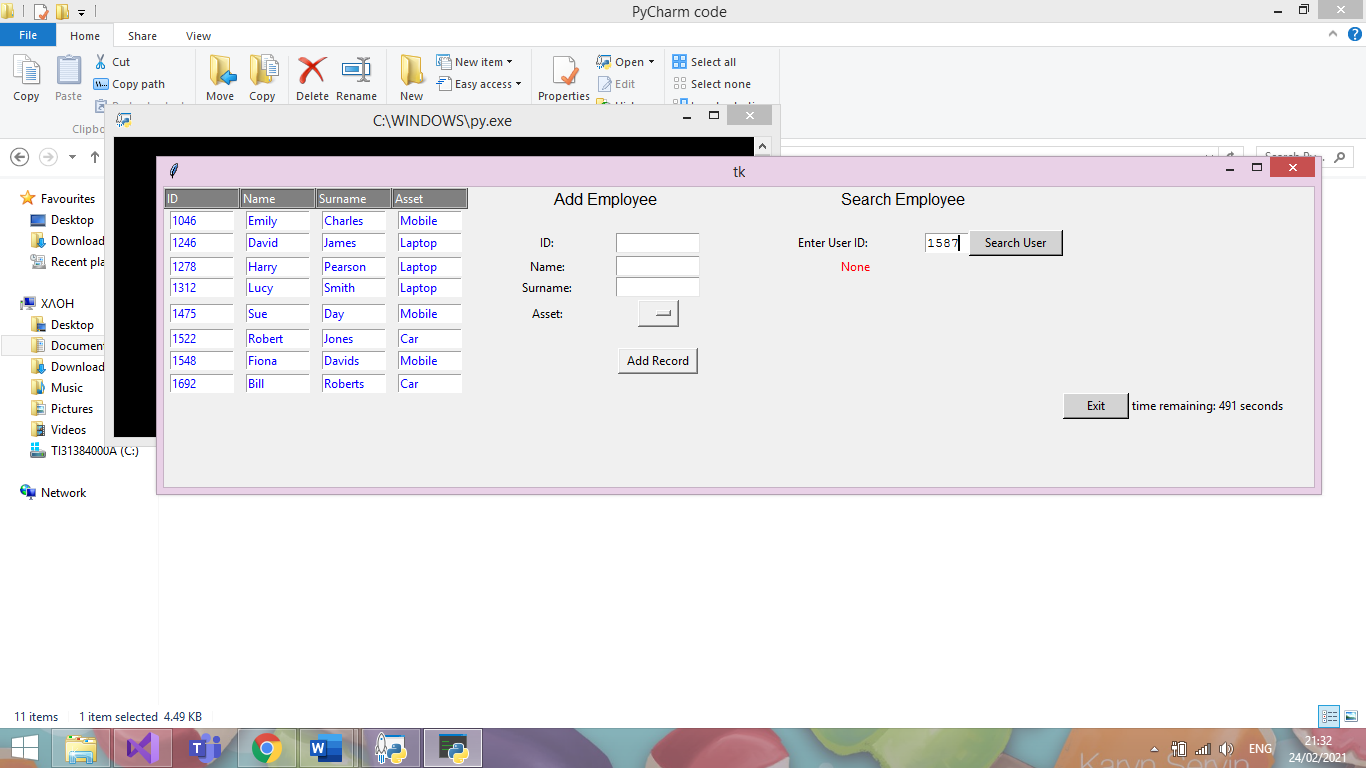
|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 25/02/21 | Tested By: | Jenni Whewell |
| System: | Login Page | Environment: | Python |
| Objective: | The user enters invalid input to search | Test ID: | 12 |
| Version: | 1 | Test Type: | Invalid |
| Status: | In Process |  |  |
|  | | | |
| Type of Tests | Before | After | Result |
| * Invalid * Boundary * Valid * Wrong * Absent | The type of test that will occur will be an Invalid test. This test will try to search for a user by a string which is not the correct data type. | An invalid test is good to check so that there are no unwanted entries in the database. This means that the test will not return the first entry in the database if the input is wrong. | This test was successful because an error message can be seen in red to indicate to check the input. |

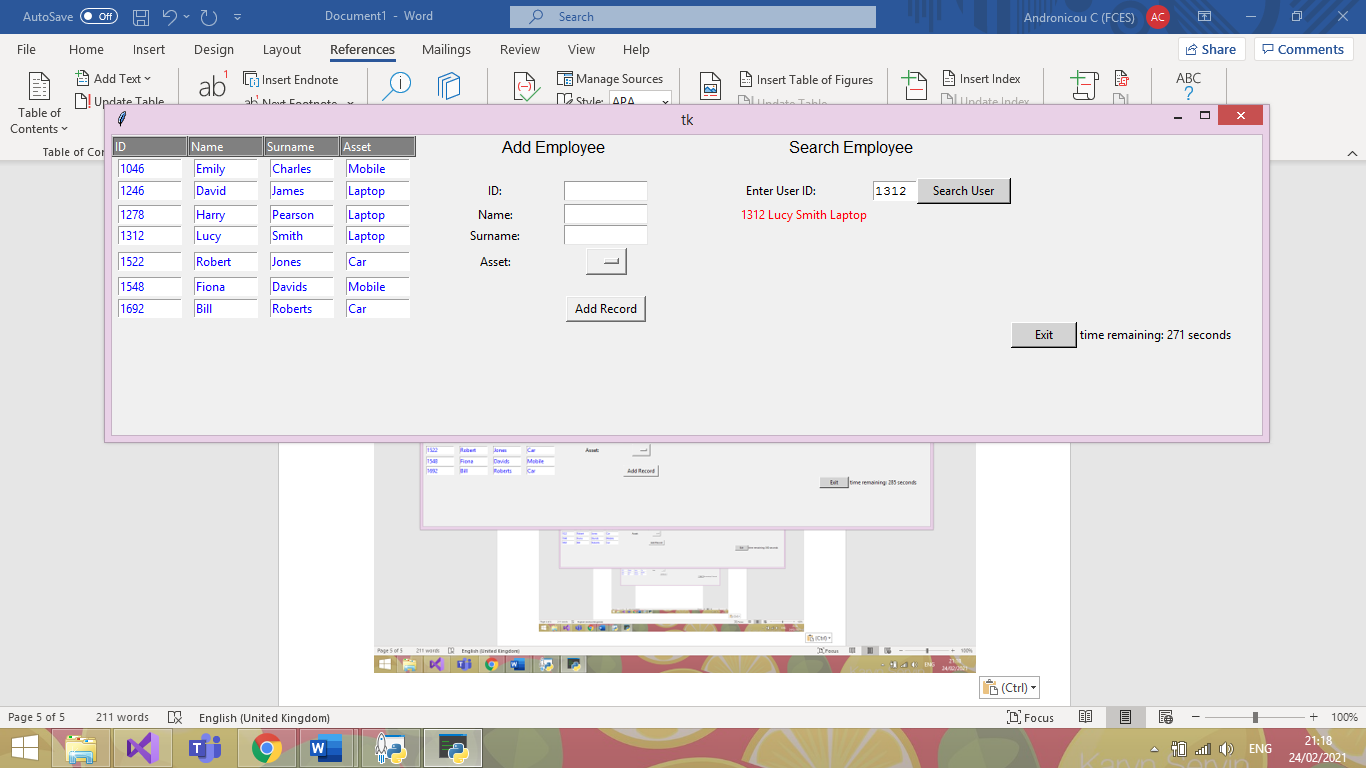
# Test 13



|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 25/02/21 | Tested By: | Jenni Whewell |
| System: | Login Page | Environment: | Python |
| Objective: | Entering a user’s ID that is not present. | Test ID: | 13 |
| Version: | 1 | Test Type: | Wrong |
| Status: | In Process |  |  |
|  | | | |
| Type of Tests | Before | After | Result |
| * Invalid * Boundary * Valid * Wrong * Absent | The type of test that will occur will be a wrong test. This test will try to search for a user using the correct data type, however, the user does not exist. | A wrong test is good to check so that a user is not getting accused of taking out an asset if that is not the case. This is vital to ensure that correct data is being stored in the database. | This test was successful because an error message can be seen in red to indicate that no such user exists in the database. Which in this case is true. |

# Test Plan 14





|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 25/02/21 | Tested By: | Jenni Whewell |
| System: | Login Page | Environment: | Python |
| Objective: | Checking that the exit buttons timer works for security purposes. | Test ID: | 14 |
| Version: | 1 | Test Type: | Valid |
| Status: | In Process |  |  |
|  | | | |
| Type of Tests | Before | After | Result |
| * Invalid * Boundary * Valid * Wrong * Absent | This valid test is to check whether the timer for the exit button works successful, counting down so that the form closes within the time limit unless something is inputted to the GUI. | In the first screenshot, the timer is seen starting when the form is initially opened.  This test is important as this means that if the user does not interact with the GUI then it will shut down.  This allows for the system to be secure so if the user leaves to computer unattended then unauthorised users cannot access it. | As you can seen from the second screenshot, the test was successful with the time remaining decreasing from the first screenshot. An important aspect of authorisation and authentication. |

# Test Plan 15

|  |  |  |
| --- | --- | --- |
| Before Deleting | When Deleting | After Deleting |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 25/02/21 | Tested By: | Jenni Whewell |
| System: | Login Page | Environment: | Python |
| Objective: | Checking that the exit buttons timer works for security purposes. | Test ID: | 15 |
| Version: | 1 | Test Type: | Valid |
| Status: | In Process |  |  |
|  | | | |
| Type of Tests | Before | After | Result |
| * Invalid * Boundary * Valid * Wrong * Absent | This test is important as it utilizes the ability to delete a user if they no longer have an asset in the company or end up leaving the company. | The first screenshot indicates all the entries in the database. The second screenshot is what happens when a user is selected. For the purpose of the, test user ID 1278. The outcome should see this user being deleted from the database and the entries below and above should all stay the same. | The test was partially successful with the entry that was required to be deleted was. However, one thing that does happen is that it duplicates another value in the database.  For fixing this error, we could set the primary key which is the user ID to not except duplicate values and this will stop the error. This can be fixed in the next phase. |

# Unit Testing:

## Author: Brendon

The main goal with the automatic testing is to ensure that any quick refactoring’s done to the project, will not damage the core functionality. Due to our project not only using a GUI, but also its use of SQLite to add a database, it makes it very difficult to conduct traditional Unit testing appropriately.

Additionally, I have attempted to incorporate DocTesting in hopes to have more success with the GUI/Database but still to no avail.



Due to the 2 external dependencies, as aforementioned, it increases the difficulty of employing these forms of testing quite significantly as you can see in the above *PrintScreen*. This as far as my research has shown, there is no real way for the testing suite to ‘see’ the changes made to the GUI.

We can however test that our input validation works correctly, and that the program wont crash when given values it cannot handle. Essentially it is only the core logic that may be tested through these means, other areas of our program must be tested manually to validate their current working state.

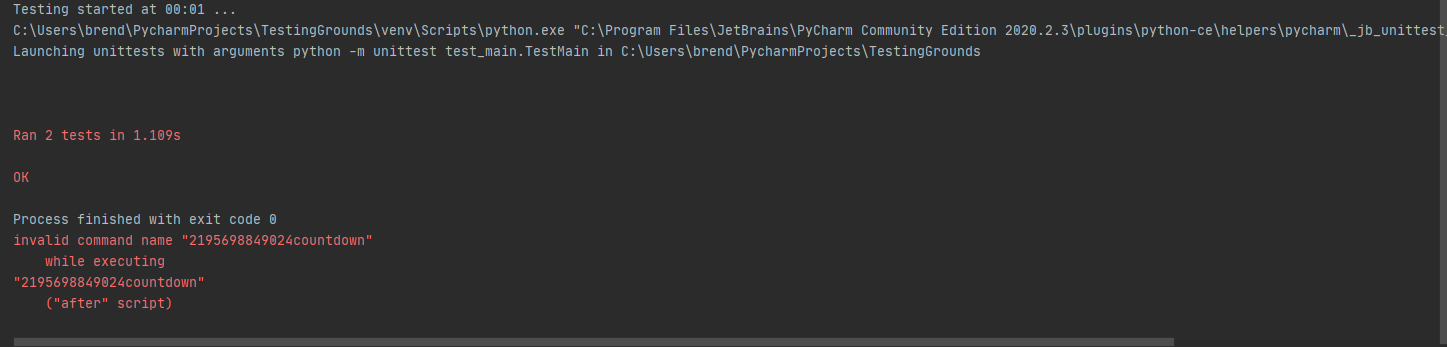
**Database GUI**

**Functions being tested:**

* Add\_Data()
* Search(ID)

**Test Purpose:**

* Check correct exceptions thrown where appropriate.
* Logic works as expected.



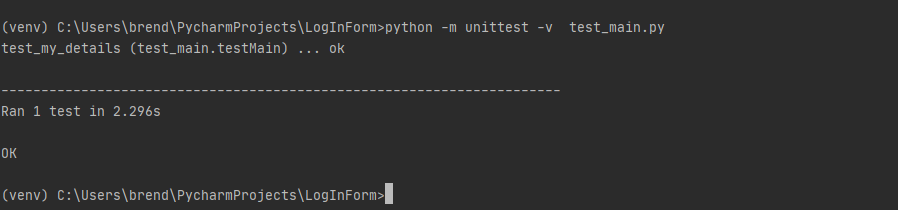
**LogInForm (with DB verification)**

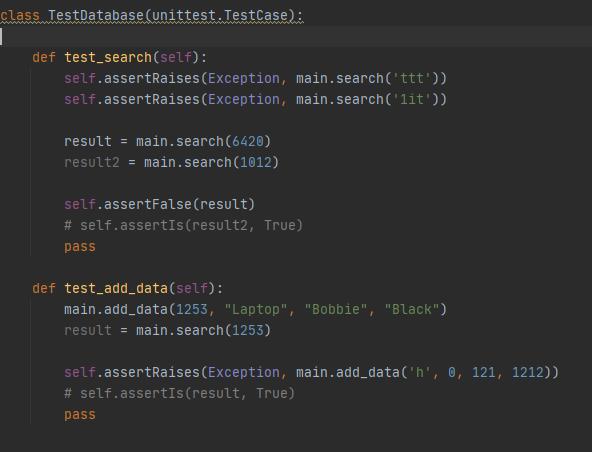
**Functions being tested:**

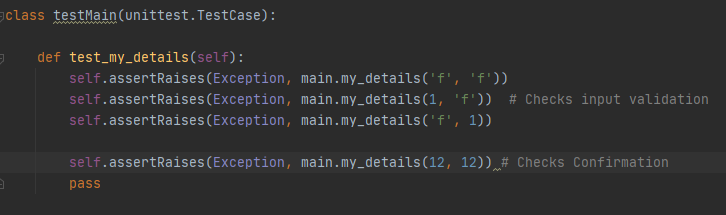
* My\_details()

**Test Purpose:**

* Check correct exceptions thrown where appropriate.



****s



References/Helpful Links (Why GUI test automation bad)

[https://www.theregister.com/2007/10/22/gui\_unit\_testing/#:~:text=Of%20course%20GUI%20applications%20can,test%20GUIs%22%20because%20it's%20difficult.](https://www.theregister.com/2007/10/22/gui_unit_testing/%23:~:text=Of%20course%20GUI%20applications%20can,test%20GUIs%22%20because%20it's%20difficult.)

Arguments that GUIs should be tested with Units in mind, however not the UI itself necessarily but rather the implemented logic of the program. Essentially confirms the methods we have followed, i.e., test base logic, throwing exceptions so on, rather than the GUIs response to them.

<https://softwareengineering.stackexchange.com/questions/207274/should-i-bother-to-write-unit-test-for-ui-ux-components>

A forum exclaiming the same thing, Unit testing has its place more for actual logic testing whereas testing the GUI and how it reacts to the code is more for **‘functional/integration’** testing.

<https://agilewarrior.wordpress.com/2015/04/18/classical-vs-mockist-testing/>

This is where I found the information regarding the ‘Mock’ class and how it can be used for emulating objects in code that you don’t necessarily want to run, a good way to explain it is if you were say hooking up to a website, rather than actually connecting to the site during every test case you run, it’ll instead connect to a ‘mock’ version of the website thus emulating the experience of connecting to a site.

Useful when working with something difficult to test or is legacy code/cannot be refactored.

I did try to implement this also, I tried to emulate a GUI framework to work with but had no real research to help with that one. I then tried to emulate the DB just to test the add\_Data() functionality, but I then had issues with SQL queries trying to run on the ‘fake’ DB.

A way we could implement the unit testing appropriately is by using an MVC pattern (cool how our modules cross over though!). Due to it being split into, you could say, ‘units’ it makes it far simpler to test. Again, only the core functionality of a given function can be tested, GUI testing still proves to be best done by human eye (as far as my research shows so far anyway, it can be done in other frameworks quite easily but so far nothing for tkinter within reason)

<https://www.semicolonworld.com/question/59599/how-do-i-run-unittest-on-a-tkinter-app>

So, this link is the closest to a solution I have found, and it entails essentially setting the GUI to execute in another thread while the testing continues in the main thread. Of course, my knowledge of threading is stretched as it is, but in this sense, I was completely lost on what to do. However, it does seem like a viable option for the future.

Here I will chuck some of my attempts at setting other tests for the program just in case you find any use for them:

This was an attempt to create a fake database just to test the add\_data function but with a separate database not to edit our original. This did not work, at all.

class TestDatabase(unittest.TestCase):  
  
 def setUp(self):  
 *"""  
 Setup a temporary database  
 """* conn = sqlite3.connect("testDatabase.db")  
 cursor = conn.cursor()  
 # create a table  
 cursor.execute("""CREATE TABLE test  
 (id text, name text, surname text,  
 options text)  
 """)  
 # insert some data  
 cursor.execute("INSERT INTO test VALUES "  
 "('1012', 'Dave', 'Parkhouse',"  
 "'Laptop'")  
 # save data to database  
 conn.commit()  
 # insert multiple records using the more secure "?" method  
 testVals = [('1654', 'Andy', 'Hunter',  
 'Laptop'),  
 ('9865', 'Red', 'Davies',  
 'Laptop'),  
 ('6545', 'Krutch',  
 'Jones', 'Tablet'),  
 ('7845', 'Lee', 'Evans',  
 'Laptop')]  
 cursor.executemany("INSERT INTO test VALUES (?,?,?,?)",  
 testVals)  
 conn.commit()  
 def tearDown(self):  
os.remove("mydatabase.db")

This was then an attempt at using ‘mock’ to emulate a database, again this did not work, at all. The SQL queries were not able to execute on my ‘test’ DB sadly so would just get constant exceptions.