Coffee Shop Documentation

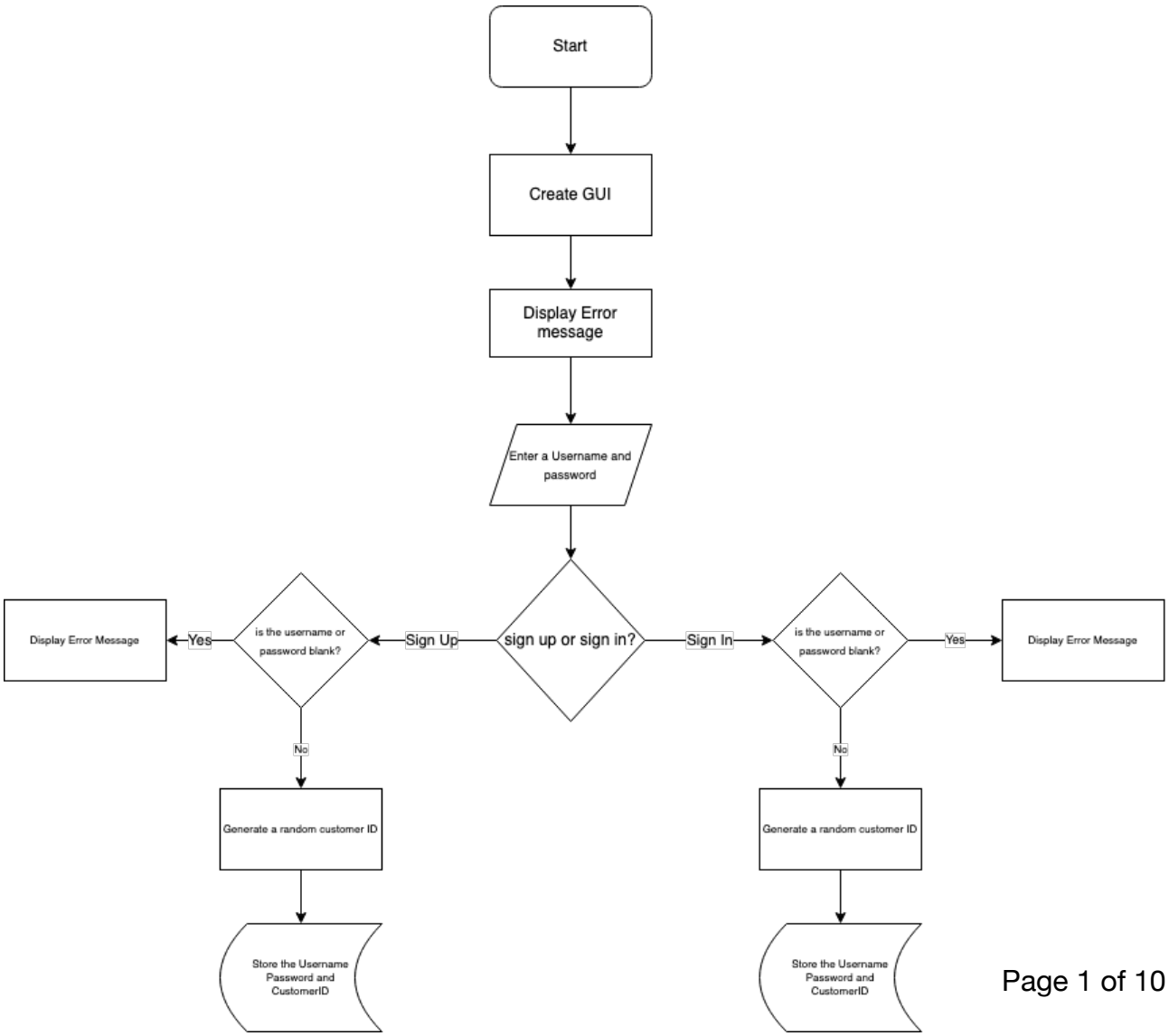
Success Criteria:

This coffee chop GUI has 4 main purposes:
- It has a login system where a user can register, and log in using a username and a password
- It allows the user to then edit data, replacing usernames and passwords on the entry of the CustomerID provided
- It allows the users to add users to the table, with the customer ID (primary key) randomly generated
- It allows users to delete a row from the table based on the customer ID provided

Algorithms:

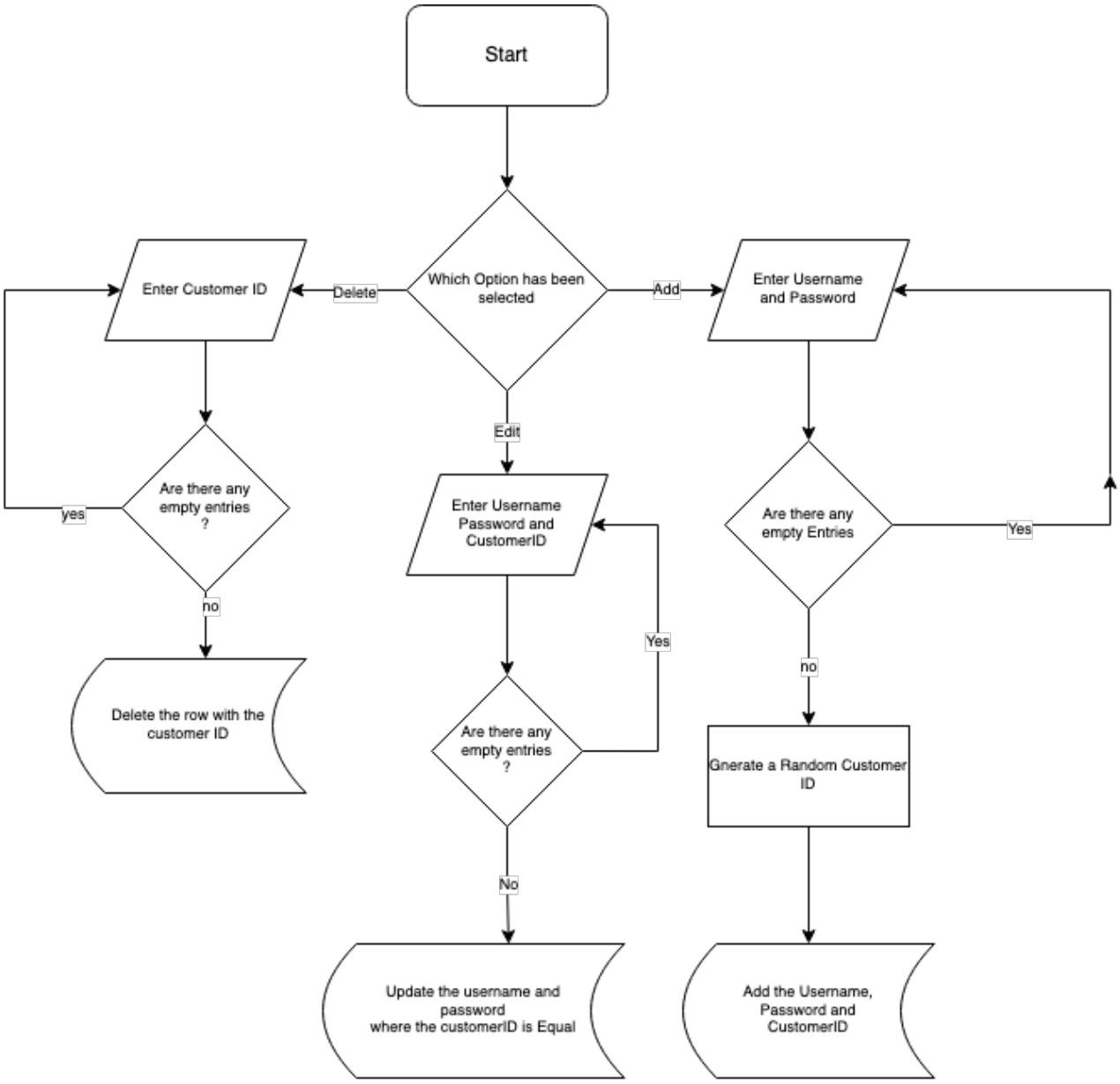
Authentication:

The first algorithm is that of the Login system. It takes in the username and password given by the User and then checks them against the data in the table.



Database Editing

This algorithm takes data inputed and inserts, deletes or updates the data in the table.



Annotated Development

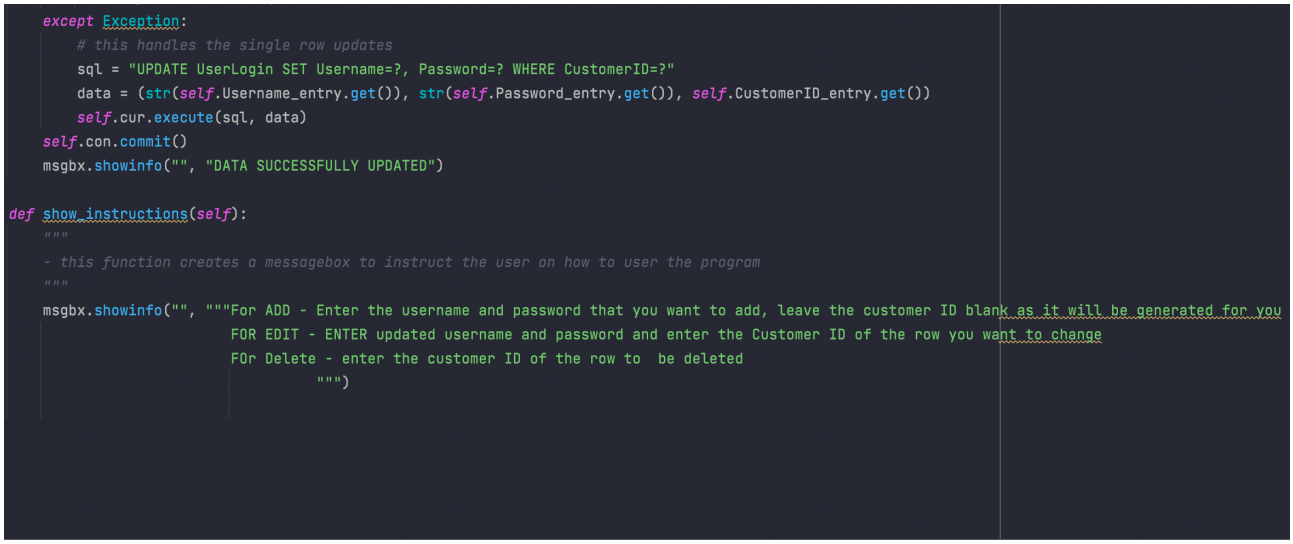
```python
import random
from tkinter import *
from tkinter import messagebox as msgbx
from EditingWindow import EditingPage
import sqlite3


class LoginWindow(Tk):

    def __init__(self, *args, **kwargs):
        """
        - The innit function is used to define key values such as width and height of the window
        - It also involves defining the connection and cursor, which allow the accessing and executing of commands on the
          database
        - The other main function of the __init__ function is to define and place all the gui widget
        """
        super().__init__(*args, **kwargs)  # this initialises the inherited gui class
        self.geometry = "400x400"  # this defines the WidthxHeight
        self.title("Db GUI")  # the title of the gui
        self.con = sqlite3.connect("CoffeeShop.db")  # connection to the database
        self.cur = self.con.cursor()  # the object used to execute sql commands

        # the base frame of the first window

        self.frame = Frame(self, width=400, height=400)  # defining the frame
        self.frame.pack()  # placing the frame

        # Defining each widget

        self.Title_label = Label(self.frame, text="Welcome To The Coffee Shop Db")
        self.username_label = Label(self.frame, text="username:")
        self.username_entry = Entry(self.frame)
        self.password_label = Label(self.frame, text="password:")
        self.password_entry = Entry(self.frame)
        self.login_button = Button(self.frame, text="login", command=self.sign_in)
        self.sign_up_button = Button(self.frame, text="sign up", command=self.sign_up)

        # this is the placing of the widgets

        self.Title_label.place(x=100, y=50)
        self.username_label.place(x=50, y=100)
        self.username_entry.place(x=140, y=100)
        self.password_label.place(x=50, y=150)
        self.password_entry.place(x=140, y=150)
        self.sign_up_button.place(x=30, y=300)
        self.login_button.place(x=300, y=300)

    def sign_up(self):
        """
        - this functions main goal is to handle the signing up of users
        - it is the function called by pressing the sign_up_button
        - First, it checks if either the username entry or the password entry are empty
          if so, it will cause a message box to pop up displaying an error
        - then the username and password are inserted into the database
        """
        if self.username_entry.get() == "" or self.password_entry.get() == "":
            # error message being shown
            msgbx.showerror("", "The Username or Password is blank - please retry")
        else:
            # sql command being executed
```

```
            self.cur.execute(
                f"INSERT INTO UserLogin(Username, Password, CustomerID) VALUES (?,?,?)",
                (self.username_entry.get(), self.password_entry.get(), random.randint(1, 1000000)))
            self.con.commit()
            msgbx.showinfo("", "You Are Signed Up")

    def sign_in(self):
        """
        - this function handles the signing in of a user
        - it does an initial check of both inputs to check if they are not empty
        - then, all the entries in the UserLogin table are retrieved
        - they are iteratively compared to the Username and Password
        - if there is a match, the user passes authentication and a new window is created
        -  if there is no match, a message is sent to the user, saying that they have been denied
        """
        if self.username_entry.get() == "" or self.password_entry.get() == "":
            # error message being displayed
            msgbx.showerror("", "The Username or Password is blank - please retry")
        else:
            self.checkInDb(username=self.username_entry.get(), password=self.password_entry.get())

    def checkInDb(self, username, password):
        # selecting all rows in user login and looping through them
        for row in self.cur.execute("SELECT * FROM UserLogin"):
            # indexing the list returned from the the line above to access the username and password
            if row[1] == username and row[2] == password:
                msgbx.showinfo("", "SIGN IN SUCCESSFUL")
                self.con.close()
                edit = EditingPage()
                edit.mainloop()
```

```
                edit.mainloop()
            else:
                pass
        msgbx.showinfo("", "SIGN IN DENIED")


if __name__ == "__main__":
    app = LoginWindow()
    app.mainloop()
```

```python
from tkinter import *
from tkinter import messagebox as msgbx
import sqlite3
import random


class EditingPage(Tk):
    def __init__(self, *args, **kwargs):
        """
         - The innit function is used to define key values such as width and height of the window
        - It also involves defining the connection and cursor, which allow the accessing and executing of commands on the
          database
        - The other main function of the __init__ function is to define and place all the gui widget
        - there is also an information message sent to the user, advising

        """
        super().__init__(*args, **kwargs)
        self.geometry = "400x400"
        self.title("Db GUI")
        self.con = sqlite3.connect("CoffeeShop.db")
        self.cur = self.con.cursor()
        # options contains the three options for the user
        self.options = ["Add", "Edit", "Delete"]
        self.drop_down_value = StringVar(self)
        self.drop_down_value.set("Edit")
        self.count = 0

        # base frame
        self.frame = Frame(self, width=400, height=400)
        self.frame.pack()
        # widgets for the gui
        self.mode_label = Label(self.frame, text="Mode:")
        self.mode_choice = OptionMenu(self.frame, self.drop_down_value, *self.options)
        self.Username_label = Label(self.frame, text="Username:")
        self.Username_entry = Entry(self.frame)
        self.Password_label = Label(self.frame, text="Password:")
        self.Password_entry = Entry(self.frame)
        self.CustomerID_label = Label(self.frame, text="CustomerID:")
        self.CustomerID_entry = Entry(self.frame)
        self.submit_button = Button(self.frame, text="Submit", command=self.check_mode)
        self.instructions_button = Button(self.frame, text="Instructions", command=self.show_instructions)
        # the placements for the widgets
        self.mode_label.place(x=100, y=70)
        self.mode_choice.place(x=150, y=70)
        self.Username_label.place(x=80, y=120)
        self.Username_entry.place(x=150, y=120)
        self.Password_label.place(x=80, y=170)
        self.Password_entry.place(x=150, y=170)
        self.CustomerID_label.place(x=70, y=220)
        self.CustomerID_entry.place(x=150, y=220)
        self.submit_button.place(x=100, y=250)
        self.instructions_button.place(x=170, y=250)

        # information for the user on how to use the program
        msgbx.showinfo("", """For ADD - Enter the username and password that you want to add, leave the customer ID blank as it will be generated for you
                              | seperate multiple values with commas
                       FOR EDIT - ENTER updated username and password and enter the Customer ID of the row you want to change
                       FOr Delete - enter the customer ID of the row to  be deleted""")

    # checks the mode that the user wants to use
```

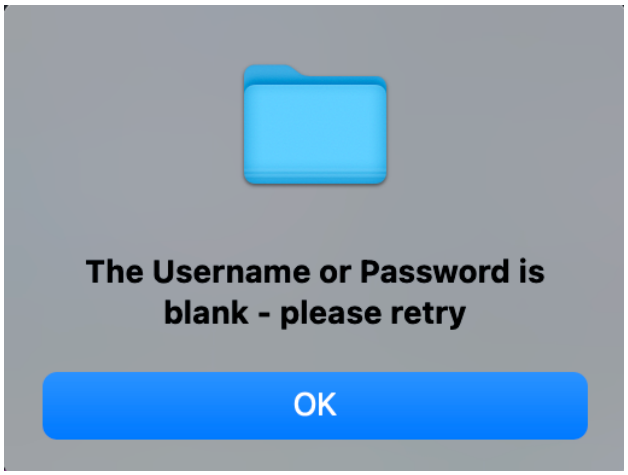
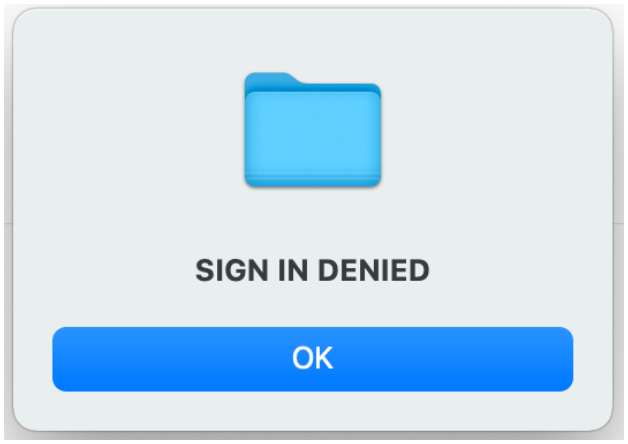
```
def check_mode(self):
    """
    - this function is run when the submit button is pressed
    - it will run a function based on the option selected by the user
      except Exception:
          # used for single deletionos
          sql = f"DELETE FROM UserLogin WHERE CustomerID=?"
          data = (str(self.CustomerID_entry.get()),)
          self.cur.execute(sql, data)
      self.con.commit()
      msgbx.showinfo("", "DATA SUCCESSFULLY DELETED")

  # Edit data
# def edit_data(self):
d     """
      - this function allows for the editing of multiple rows
      - you can change the username and password of a user by inputting the username, password and customer ID
      - it can handle multiple or single updates
      - this is enabled by the try and except clause
      """
      try:
          sql = "UPDATE UserLogin SET Username=?, Password=? WHERE CustomerID=?"
          # this retrieves the data and converts each one to a string
          data = (str(self.Username_entry.get()), str(self.Password_entry.get()), str(self.CustomerID_entry.get()))
          # these create lists of the Username, Password and CustomerID
          usernames = data[0].split(",")
          passwords = data[1].split(",")
          Customer_IDs = data[2].split(",")
          # each set of values is iterated over and the rows are each updated
          for i in range(len(usernames)):
              data = (str(usernames[(i-1)]), str(passwords[i-1]), str(Customer_IDs[i-1]))
              self.cur.execute(sql, data)
      except Exception:
          # this handles the single row updates
        # this inputs a single row into the table
        sql = f"INSERT INTO UserLogin(Username, Password, CustomerID) Values(?,?,?)"
        data = (str(self.Username_entry.get()), str(self.Password_entry.get()), random.randint(1, 1000000000000))
        self.cur.execute(sql, data)
    self.con.commit()
    msgbx.showinfo("", "DATA SUCCESSFULLY ADDED")

# Deleting Data
def delete_data(self):
    """
    - this function allows the deleting of rows of data
    - it takes the customer ID (primary key) and deletes the corresponding rows
    - it can handle both single and multiple row deletions using the try, execpt clause
    """
    try:
        # used for multiple deletions
        sql = f"DELETE FROM UserLogin WHERE CustomerID=?"
        # the customer ID is retrieved from the input and is converted to a string
        data = (str(self.CustomerID_entry.get()))
        # a list of customer IDs is created
        Customer_IDs = data.split(",")
        # each customer ID is iterated over and the row is deleted
        for i in range(len(Customer_IDs)):
            data = (str(Customer_IDs[(i-1)]),)
            self.cur.execute(sql, data)
```

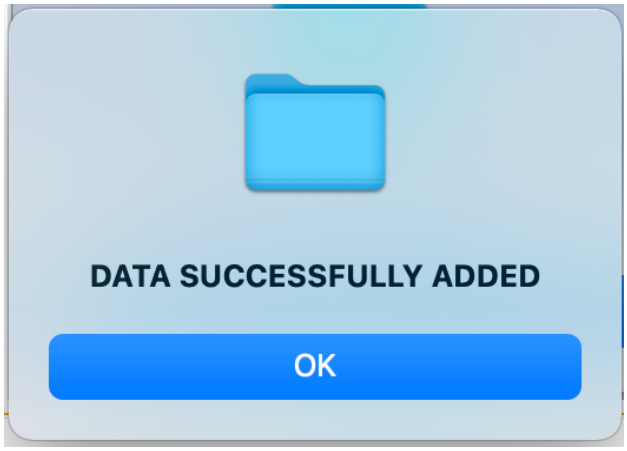
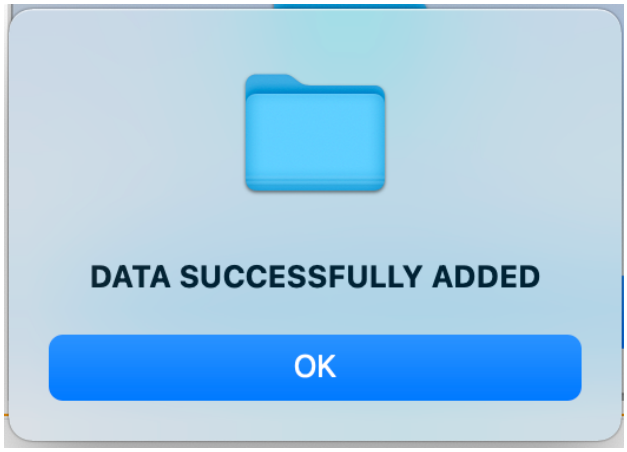
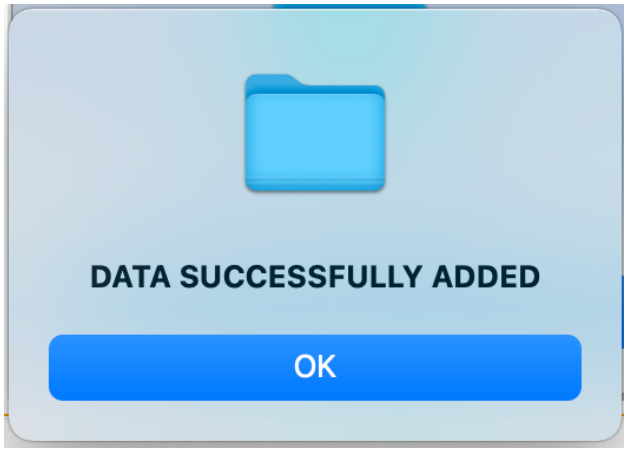
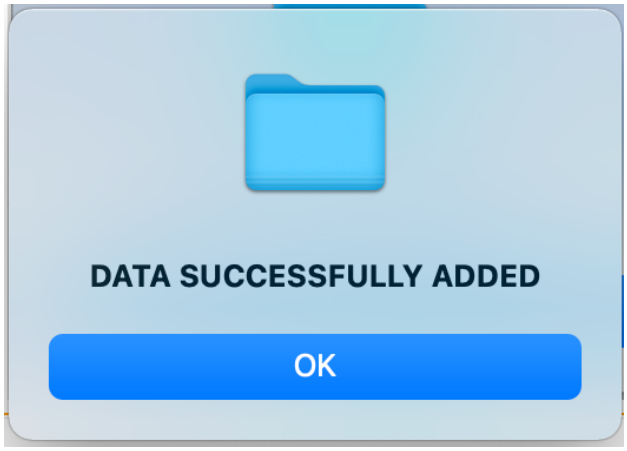


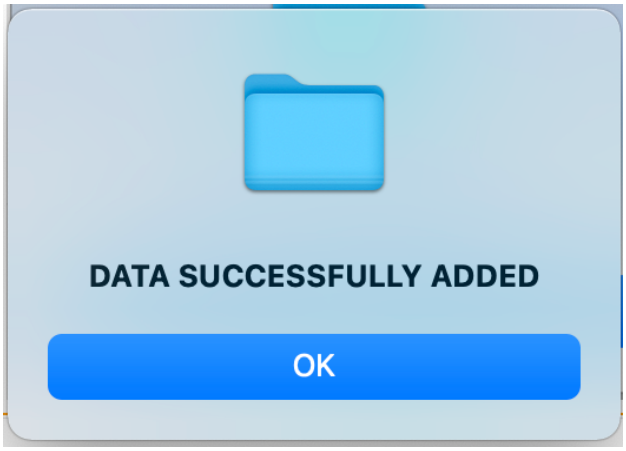
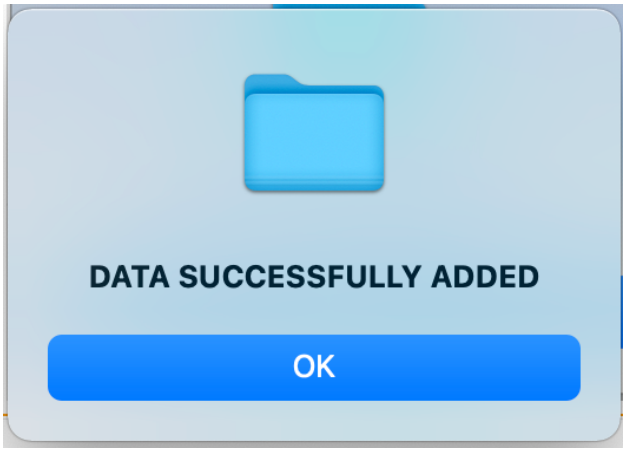
```
    except Exception:
        # this handles the single row updates
        sql = "UPDATE UserLogin SET Username=?, Password=? WHERE CustomerID=?"
        data = (str(self.Username_entry.get()), str(self.Password_entry.get()), self.CustomerID_entry.get())
        self.cur.execute(sql, data)
    self.con.commit()
    msgbx.showinfo("", "DATA SUCCESSFULLY UPDATED")

def show_instructions(self):
    """
    - this function creates a messagebox to instruct the user on how to user the program
    """
    msgbx.showinfo("", """For ADD - Enter the username and password that you want to add, leave the customer ID blank as it will be generated for you
                        FOR EDIT - ENTER updated username and password and enter the Customer ID of the row you want to change
                        FOr Delete - enter the customer ID of the row to  be deleted
                                """)
```

These screenshots cover both LoginWindow.py and EditingWindow.py. The comments detail the functionality of each function and specific lines.

Testing Screenshots

Testing Scenario	Screenshot of message returned
Blank Username and Password input	
Authenticating Incorrectly	
Authenticating Correctly	

Adding Single users	
Adding Multiple users	
Editing Single users	
Editing Multiple users	

Deleting a Single user	
Deleting multiple users	

Evaluation

Overall, this project has been a success as it has met the success criteria. I think the implementation of multiple entries in one go using try, except clauses was a particular success. However, the database GUI lacks an ability to view data, which can hold it back. If I were to redo this project, I would have added a data viewing screen to allow for easier use.