'''

Michelle Luo

Cmdr. Schenk

AP Computer Science

Master Project Main (cryptomain.py)

Spring of 2023

'''

#imports

from cryptogui import CryptoGUI

#run main method

def main():

    #run the Tkinter GUI file

    michelleGUI=CryptoGUI()

#if so, run main method

if (\_\_name\_\_ == "\_\_main\_\_"):

  main()

'''

Michelle Luo

Cmdr. Schenk

AP Computer Science

Master Project Database (cryptodatabase.py)

Spring of 2023

'''

#imports

import mysql.connector

#FIX: marks where everything is

class CryptoDatabase:

    #\_\_init\_\_ method

    def \_\_init\_\_(self):

        self.connect=False

    #open connection method

    def setConnection(self):

        if (self.connect == False):

            try:

                self.connection = mysql.connector.connect(

                #connection to localhost at 127.0.0.1

                host = "localhost",

                user = "root",

                password = "!@#seojean\_and\_sam90",

                database = "crypto"

                # connection at jchs :D

                # host="192.168.0.116"

                # user="leia"

                # password="jchs"

                # database="crypto"

                )

                self.cursor = self.connection.cursor()

                self.connect=True

            except:

                return 1

            return 0

    #close connection method

    def breakConnection(self):

        if (self.connect == True):

            try:

                self.cursor.close()

                self.connection.close()

                self.connect=False

            except:

                return 1

            return 0

    #checks the account login details to make sure the credentials are true

    def checkPassword(self, username, password):

        self.cursor.execute("SELECT id\_account FROM account WHERE username='{}' AND password='{}'".format(username, password))

        resultList=self.cursor.fetchall()

        try:

            resultTuple=resultList[0]

            id\_password=resultTuple[0]

            return id\_password

        except:

            return "incorrect password"

    #checks the account's password to make sure that the user wants to access

    def checkPassword2(self, userID, password):

        self.cursor.execute("SELECT id\_account FROM account WHERE id\_account='{}' AND password='{}'".format(userID, password))

        resultList=self.cursor.fetchall()

        try:

            resultTuple=resultList[0]

            id\_password=resultTuple[0]

            return id\_password

        except:

            return "incorrect password"

    #creates new account

    def startAccount(self, email, username, password):

        self.cursor.execute("SELECT id\_account FROM account WHERE username='{}';".format(username))

        resultList=self.cursor.fetchall()

        if len(resultList)!=0:

            return "Account taken"

        else:

            thingy="INSERT INTO account (email, username, password) VALUES ('{}','{}','{}');".format(email,username,password)

            self.cursor.execute(thingy)

            self.connection.commit()

            result=self.cursor.fetchall()

    #inserts new entry into crypto db under user's account

    def insertCryptoDB(self, cryptoID,plainText,cipherText,user1,user2,userID):

        try:

            if user1=="NULL":

                user1=None

            if user2=="NULL":

                user2=None

            if user1==None:

                insertQ="INSERT INTO encrypt (id\_total\_encrypt, id\_user\_encrypt, plain\_text, cipher\_text) VALUES ('{}','{}','{}','{}');".format(cryptoID,userID,plainText,cipherText)

                self.cursor.execute(insertQ)

                self.connection.commit()

                result=self.cursor.fetchall()

                return 0

            elif user2==None:

                insertQ="INSERT INTO encrypt (id\_total\_encrypt, id\_user\_encrypt, plain\_text, cipher\_text, encryption\_type) VALUES ('{}','{}','{}','{}','{}');".format(cryptoID,userID,plainText,cipherText,user1)

                self.cursor.execute(insertQ)

                self.connection.commit()

                result=self.cursor.fetchall()

                return 0

            else:

                insertQ="INSERT INTO encrypt (id\_total\_encrypt, id\_user\_encrypt, plain\_text, cipher\_text, encryption\_type, encryption\_type2) VALUES ('{}','{}','{}','{}','{}','{}');".format(cryptoID,userID,plainText,cipherText,user1,user2)

                self.cursor.execute(insertQ)

                self.connection.commit()

                result=self.cursor.fetchall()

                return 0

        except:

            return []

    #updates entry in crypto db under user's account

    def updateCryptoDB(self, cryptoID,plainText,cipherText,user1,user2,userID):

        try:

            if user1=="NULL":

                user1=None

            if user2=="NULL":

                user2=None

            if user1==None:

                updateQ="UPDATE encrypt SET plain\_text='{}', cipher\_text='{}' WHERE id\_user\_encrypt={} AND id\_total\_encrypt='{}';".format(plainText, cipherText, userID, cryptoID)

                self.cursor.execute(updateQ)

                self.connection.commit()

                result=self.cursor.fetchall()

                return 0

            elif user2==None:

                updateQ="UPDATE encrypt SET plain\_text='{}', cipher\_text='{}', encryption\_type='{}' WHERE id\_user\_encrypt={} AND id\_total\_encrypt='{}';".format(plainText, cipherText, user1, userID, cryptoID)

                self.cursor.execute(updateQ)

                self.connection.commit()

                result=self.cursor.fetchall()

                return 0

            else:

                updateQ="UPDATE encrypt SET plain\_text='{}', cipher\_text='{}', encryption\_type='{}', encryption\_type2='{}' WHERE id\_user\_encrypt={} AND id\_total\_encrypt='{}';".format(plainText, cipherText, user1, user2, userID, cryptoID)

                self.cursor.execute(updateQ)

                self.connection.commit()

                result=self.cursor.fetchall()

                return 0

        except:

            return []

    #deletes entry in crypto db under user's account

    def deleteCryptoDB(self, cryptoID, userID):

        try:

            deleteQ="DELETE FROM encrypt WHERE id\_total\_encrypt={} AND id\_user\_encrypt={}".format(cryptoID,userID)

            self.cursor.execute(deleteQ)

            self.connection.commit()

            result=self.cursor.fetchall()

            return 0

        except:

            return []

    #go to as specific entry in crypto db under user's account

    def goToEntry(self, entryNumber,userID):

        try:

            selectQ="SELECT \* FROM encrypt WHERE id\_user\_encrypt={} AND id\_total\_encrypt={}".format(userID, entryNumber)

            self.cursor.execute(selectQ)

            listResult=self.cursor.fetchall()

            tupleResult=listResult[0]

            return tupleResult

        except:

            return "No records available to navigate"

    #find the id of the last entry in table under user's account

    def findMaxLength(self, userID):

        maxQuery="SELECT \* FROM encrypt WHERE id\_user\_encrypt={} ORDER BY id\_total\_encrypt DESC".format(userID)

        self.cursor.execute(maxQuery)

        listResult=self.cursor.fetchall()

        if listResult==[]:

            finalID=0

            return finalID

        tupleResult=listResult[0]

        finalID=tupleResult[1]

        return finalID

'''

Michelle Luo

Cmdr. Schenk

AP Computer Science

Master Project Encryption methods (encryption.py)

Spring of 2023

'''

#imports

import base64

import hashlib

from Crypto import Random

from Crypto.Cipher import AES

from base64 import b32encode

import random, math

#global public\_key, private\_key, n, prime

#Encryption class

class Encryption:

    def \_\_init\_\_(self):

        self.bs = AES.block\_size

        self.key = ""#hashlib.sha256(key.encode()).digest()

    #function for encrypting with caesar cipher

    def caesar(self, plainText, key):

        cipherText=""

        # traverse text

        for i in range(len(plainText)):

            char = plainText[i]

            # Encrypt uppercase characters

            if (char.isupper()):

                cipherText+= chr((ord(char) + key-65) % 26 + 65)

            # Encrypt lowercase characters

            elif (char==" "):

                cipherText+=char

            else:

                cipherText += chr((ord(char) + key - 97) % 26 + 97)

        return cipherText

    #function for encrypting with book cipher

    def bookCipher(self, plainText, bookFileAddress):

        self.cipherTextList=[]

        self.linePosition=0

        self.wordPosition=0

        self.plainTextList=plainText.split(" ")

        for stringWord in self.plainTextList:

            self.finalPosition=self.bcFindWord(stringWord, bookFileAddress)

            if self.finalPosition=="NOT A WORD FOUND":

                self.cipherTextList.append(stringWord)

            else:

                self.cipherTextList.append(self.finalPosition)

        self.cipherText=" ".join(self.cipherTextList)

        return self.cipherText

    #used with book cipher to find word in text files

    def bcFindWord(self, stringWord, bookFileAddress):

        with open(bookFileAddress, encoding='utf-8') as myBookFile:

            self.linePosition=0

            for bookLine in myBookFile:

                self.linePosition+=1

                self.wordPosition=0

                wordList=bookLine.split(" ")

                for bookWord in wordList:

                    self.wordPosition+=1

                    if (bookWord==stringWord):

                        self.finalPosition=str(self.linePosition)+"."+str(self.wordPosition)

                        return self.finalPosition

                    else:

                        continue

        return "NOT A WORD FOUND"

    #function for encrypting with hex

    def hexString(self, plainText):

        cipherText=plainText.encode().hex()

        return cipherText

    #function for encrypting with base32

    def b32(self, plainText):

        cipherText=b32encode(plainText.encode())

        return cipherText

    def AESencrypt(self, raw, key):

        self.key=hashlib.sha256(key.encode()).digest()

        raw = self.AES\_pad(raw)

        iv = Random.new().read(AES.block\_size)

        cipher = AES.new(self.key, AES.MODE\_CBC, iv)

        return base64.b64encode(iv + cipher.encrypt(raw.encode()))

    def AESdecrypt(self, enc):

        enc = base64.b64decode(enc)

        iv = enc[:AES.block\_size]

        cipher = AES.new(self.key, AES.MODE\_CBC, iv)

        return self.AES\_unpad(cipher.decrypt(enc[AES.block\_size:])).decode('utf-8')

    def AES\_pad(self, s):

        return s + (self.bs - len(s) % self.bs) \* chr(self.bs - len(s) % self.bs)

    @staticmethod

    def AES\_unpad(s):

        return s[:-ord(s[len(s)-1:])]

'''

Michelle Luo

Cmdr. Schenk

AP Computer Science

Master Project GUI (cryptogui.py)

Spring of 2023

'''

#imports

from cryptodatabase import CryptoDatabase

from encryption import Encryption

from tkinter import messagebox

from tkinter import scrolledtext

from tkinter import filedialog

from tkinter import ttk

import tkinter as tk

from PIL import ImageTk, Image

class CryptoGUI:

    #the class' \_\_init\_\_

    def \_\_init\_\_(self):

        #uniform variables for text color and font [the s stands for standard]

        self.textColor= "#00cc00"

        self.sFont= "Terminal"

        self.userMessage=""

        self.isEncryptButton="False"

        self.recentSelection1=""

        self.fileName=""

        self.userChoice1=""

        self.userChoice2=""

        self.userID=0

        self.entryNumber=0

        self.connected=True

        self.navIsDisabled=True

        self.keyResult=""

        #define connection and instantiation of the database file

        self.db=CryptoDatabase()

        self.encrypt=Encryption()

        self.db.setConnection()

        #processes to run

        self.createLoginWindow()

        self.createLoginMenu()

        self.createLoginButtons()

        #self.createCryptoButtons()

        self.loginWindow.mainloop()

    #creates the window and runs mainloop

    def createLoginWindow(self):

        #define the loginWindow and its attributes

        self.loginWindow=tk.Tk()

        self.loginWindow.title("Login")

        self.loginWindow.geometry("1280x600")

        self.loginWindow['bg']="black"

        self.loginWindow.attributes('-fullscreen',True)

        #HERE IG IDK IF THIS WORKS FOR u but here

        self.logo = Image.open("logo.png")

        self.logo = self.logo.resize((256, 120), Image.ANTIALIAS)

        self.lPic = ImageTk.PhotoImage(self.logo)

        self.lMyPic = tk.Label(self.loginWindow, image=self.lPic, bg="black")

        self.lMyPic.place(x = 50, y = 10)

    #creates the menu at the top

    def createLoginMenu(self):

        #menu system initiated

        self.menubar= tk.Menu(self.loginWindow)

        self.loginWindow.config(menu= self.menubar)

        self.fileMenu = tk.Menu(self.menubar, tearoff = 0)

        self.fileMenu.add\_command(label = "Exit", font = (self.sFont, 18), foreground = (self.textColor), command = self.loginWindow.destroy)

        self.menubar.add\_cascade(label = "File", font = (self.sFont, 18), foreground = (self.textColor), menu = self.fileMenu)

        self.helpMenu = tk.Menu(self.menubar, tearoff = 0)

        self.helpMenu.add\_command(label="About",font = (self.sFont, 18), foreground = (self.textColor), command = self.aboutWindow)

        self.menubar.add\_cascade(label = "Help", font = (self.sFont, 18), foreground = (self.textColor), menu = self.helpMenu)

    #creates the CRUD menu when user is logged into Crypto

    def createCryptoMenu(self):

        self.crudMenu = tk.Menu(self.menubar, tearoff = 0)

        self.crudMenu.add\_command(label = "Create", font = (self.sFont, 18), foreground = (self.textColor), command = self.insertCryptoGUI)

        self.crudMenu.add\_command(label = "Update", font = (self.sFont, 18), foreground = (self.textColor), command = self.updateCryptoGUI)

        self.crudMenu.add\_command(label = "Delete", font = (self.sFont, 18), foreground = (self.textColor), command = self.deleteCryptoGUI)

        self.menubar.add\_cascade(label = "CRUD", font = (self.sFont, 18), foreground = (self.textColor), menu = self.crudMenu)

    #prompt the user for the password to access navigation

    def passwordPrompt(self):

        inputtedPassword = tk.simpledialog.askstring("Verify Identity", "Please enter your password again to confirm that you want to access the decrypted messages.")

        result = self.db.checkPassword2(self.userID, inputtedPassword)

        if result == "incorrect password":

            return 1

        self.buttonCoverUp.destroy()

        self.navIsDisabled=False

    #creates the widgets needed for the login UI

    def createLoginButtons(self):

        #define buttons

        self.exitButton=tk.Button(self.loginWindow, text="Exit", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.destroyWindow)

        self.aboutButton=tk.Button(self.loginWindow, text = "About", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.aboutWindow)

        self.helpButton=tk.Button(self.loginWindow, text = "Help", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.helpWindow)

        self.loginButton=tk.Button(self.loginWindow, text = "LOGIN", font=(self.sFont, 20), fg=(self.textColor), bg=("black"),command=self.login)

        self.startSignupButton=tk.Button(self.loginWindow, text = "If you do not have a Crypto account, sign up today!", font=(self.sFont, 12), fg=(self.textColor), borderwidth= 0,bg=("black"),command=self.openSignup)

        #place buttons

        self.exitButton.place(x=325,y=50)

        self.aboutButton.place(x=415,y=50)

        self.helpButton.place(x=525,y=50)

        self.loginButton.place(x=590,y=400)

        self.startSignupButton.place(x=350,y=580)

        #define labels

        self.usernameLabel=tk.Label(self.loginWindow,text="Username",font=(self.sFont, 18), bg=("black"), fg=(self.textColor))

        self.passwordLabel=tk.Label(self.loginWindow,text="Password",font=(self.sFont, 18), bg=("black"), fg=(self.textColor))

        self.userMessageLabel=tk.Label(self.loginWindow,text=self.userMessage,font=(self.sFont, 12), bg=("black"), fg=("#ff3300"))

        #define Text fields

        self.usernameEntry=tk.Entry(self.loginWindow, font=(self.sFont, 18), relief= "sunken")

        self.passwordEntry=tk.Entry(self.loginWindow, font=(self.sFont, 18), relief= "sunken", show="\*")

        #place labels

        self.usernameLabel.place(x=480,y=200)

        self.passwordLabel.place(x=480,y=300)

        self.userMessageLabel.place(x=505,y=170)

        #place text fields

        self.usernameEntry.place(x=480,y=250)

        self.passwordEntry.place(x=480,y=350)

    #creates the widgets needed for the sign up UI

    def createSignupButtons(self):

        #define signup buttons

        self.backButton=tk.Button(self.loginWindow, text="Back", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.openLoginFromSU)

        self.aboutButton=tk.Button(self.loginWindow, text = "About", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.aboutWindow)

        self.helpButton=tk.Button(self.loginWindow, text = "Help", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.helpWindow)

        self.signupButton=tk.Button(self.loginWindow, text="Sign Up", font = (self.sFont, 20), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.signup)

        #define signup labels

        self.signupLabel=tk.Label(self.loginWindow, text="Crypto Account Sign Up", font=(self.sFont, 20), bg=("black"), fg=(self.textColor))

        self.emailSULabel=tk.Label(self.loginWindow, text="Email", font=(self.sFont, 14), bg=("black"), fg=(self.textColor))

        self.newUsernameSULabel=tk.Label(self.loginWindow, text="Enter your Username below", font=(self.sFont, 14), bg=("black"), fg=(self.textColor))

        self.newPasswordSULabel=tk.Label(self.loginWindow, text="Enter your Password below", font=(self.sFont, 14), bg=("black"), fg=(self.textColor))

        self.userMessageLabel=tk.Label(self.loginWindow,text=self.userMessage,font=(self.sFont, 12), bg=("black"), fg=("#ff3300"))

        #define signup entries

        self.emailSUEntry=tk.Entry(self.loginWindow, font=(self.sFont, 18), relief= "sunken")

        self.usernameSUEntry=tk.Entry(self.loginWindow, font=(self.sFont, 18), relief= "sunken")

        self.PasswordSUEntry=tk.Entry(self.loginWindow, font=(self.sFont, 18), relief= "sunken", show="\*")

        #place signup buttons

        self.backButton.place(x=325,y=50)

        self.aboutButton.place(x=415,y=50)

        self.helpButton.place(x=525,y=50)

        self.signupButton.place(x=540,y=500)

        #place signup labels

        self.signupLabel.place(x=440,y=150)

        self.emailSULabel.place(x=450,y=200)

        self.newUsernameSULabel.place(x=450,y=300)

        self.newPasswordSULabel.place(x=450,y=400)

        self.userMessageLabel.place(x=430,y=180)

        #place signup entries

        self.emailSUEntry.place(x=450,y=250)

        self.usernameSUEntry.place(x=450,y=350)

        self.PasswordSUEntry.place(x=450,y=450)

        self.bgPic = Image.open("hacka.png")

        #self.bgPic = self.bgPic.resize((512, 226), Image.ANTIALIAS)

        self.bPic = ImageTk.PhotoImage(self.bgPic)

        self.bMyPic = tk.Label(self.loginWindow, image=self.bPic, bg="black")

        self.bMyPic.place(x = 850, y = 200)

        self.bg2Pic = Image.open("computa.png")

        self.bg2Pic = self.bg2Pic.resize((402, 236), Image.ANTIALIAS)

        self.b2Pic = ImageTk.PhotoImage(self.bg2Pic)

        self.b2MyPic = tk.Label(self.loginWindow, image=self.b2Pic, bg="black")

        self.b2MyPic.place(x = 0, y = 230)

    #creates the widgets needed for the Crypto UI

    def createCryptoButtons(self):

        #logo to fill up space in the interface

        self.logo2 = Image.open("logo2.png")

        self.logo2 = self.logo2.resize((350, 350), Image.ANTIALIAS)

        self.l2Pic = ImageTk.PhotoImage(self.logo2)

        self.l2MyPic = tk.Label(self.loginWindow, image=self.l2Pic, bg="black")

        self.l2MyPic.place(x = 925, y = 350)

        self.createCryptoMenu()

        self.backButton=tk.Button(self.loginWindow, text="Log out", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.openLoginFromCrypto)

        self.backButton.place(x=325,y=50)

        self.connectionButton=tk.Button(self.loginWindow, text="Disconnect", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.connectionGUI)

        self.connectionButton.place(x=325,y=100)

        self.clearButton=tk.Button(self.loginWindow, text="Clear Screen", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.clearActivity)

        self.clearButton.place(x=325,y=150)

        #define self.idText

        self.idText=""

        #GUI fields' labels, text fields,etc

        #create labels:

        self.userMessageLabel=tk.Label(self.loginWindow,text=self.userMessage,font=(self.sFont, 12), bg=("black"), fg=("#ff3300"))

        self.idLabel=tk.Label(self.loginWindow, text="Encryption Number", font=(self.sFont, 18), bg=("black"), fg=(self.textColor))

        self.idText=tk.Label(self.loginWindow, text=self.idText, font=(self.sFont, 18), bg=("black"), fg=(self.textColor))

        self.plainTextLabel=tk.Label(self.loginWindow, text="Plain Text", font=(self.sFont, 18), bg=("black"), fg=(self.textColor))

        self.cipherTextLabel=tk.Label(self.loginWindow, text="Cipher Text", font=(self.sFont, 18), bg=("black"), fg=(self.textColor), state="disabled")

        self.encryptionSectionLabel=tk.Label(self.loginWindow, text="Encryption Options", font=(self.sFont, 20), bg=("black"), fg=(self.textColor))

        #place labels

        self.userMessageLabel.place(x=400,y=250)

        self.idLabel.place(x=54,y=200)

        self.idText.place(x=350,y=200)

        self.plainTextLabel.place(x=50,y=300)

        self.cipherTextLabel.place(x=500, y=300)

        self.encryptionSectionLabel.place(x=950,y= 75)

        self.plainTextFrame = tk.Frame(self.loginWindow, width = 380, height = 245)

        self.cipherTextFrame = tk.Frame(self.loginWindow, width = 380, height = 245)

        self.plainTextFrame.place(x = 50, y = 350)

        self.cipherTextFrame.place( x= 500, y = 350)

        #create text fields:

        self.plainTextField=scrolledtext.ScrolledText(self.plainTextFrame, font=(self.sFont, 12), height=15, width=30, wrap=tk.WORD)

        self.cipherTextField=scrolledtext.ScrolledText(self.cipherTextFrame, font=(self.sFont, 12), height=15, width=30, wrap=tk.WORD)

        self.cipherTextField["state"]="disabled"

        #place text fields

        self.plainTextField.place(x=0,y=0)

        self.cipherTextField.place(x=0,y=0)

        #CRUD Buttons for GUI

        #create buttons

        self.insertButton=tk.Button(self.loginWindow, text="Save New Message", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.insertCryptoGUI)

        self.updateButton=tk.Button(self.loginWindow, text="Save Changes", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.updateCryptoGUI)

        self.deleteButton=tk.Button(self.loginWindow, text="Delete a Message", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.deleteCryptoGUI)

        self.backAllButton=tk.Button(self.loginWindow, text="|<<", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.backAllCryptoGUI)

        self.back2Button=tk.Button(self.loginWindow, text="<<", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.back2CryptoGUI)

        self.back1Button=tk.Button(self.loginWindow, text="<", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.back1CryptoGUI)

        self.forward1Button=tk.Button(self.loginWindow, text=">", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.forward1CryptoGUI)

        self.forward2Button=tk.Button(self.loginWindow, text=">>", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.forward2CryptoGUI)

        self.forwardAllButton=tk.Button(self.loginWindow, text=">>|", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.forwardAllCryptoGUI)

        self.buttonCoverUp=tk.Button(self.loginWindow, text="Verify your identity to access Navigation.", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.passwordPrompt)

        #place buttons

        self.insertButton.place(x=600,y=50)

        self.updateButton.place(x=600,y=100)

        self.deleteButton.place(x=600,y=150)

        self.backAllButton.place(x=200,y=600)

        self.back2Button.place(x=300,y=600)

        self.back1Button.place(x=400,y=600)

        self.forward1Button.place(x=500,y=600)

        self.forward2Button.place(x=600,y=600)

        self.forwardAllButton.place(x=700,y=600)

        self.buttonCoverUp.place(x=200,y=600)

        #line Image to seperate Text Fields with the Encryption Fields on GUI

        self.bgPic = Image.open("greenLine.png")

        self.bgPic = self.bgPic.resize((30, 600), Image.ANTIALIAS)

        self.bPic = ImageTk.PhotoImage(self.bgPic)

        self.bMyPic = tk.Label(self.loginWindow, image=self.bPic, bg="black")

        self.bMyPic.place(x = 900, y = 50)

        #all SPECIFICALLY encryption selection buttons, labels, etc.

        #ComboBox 1's LABEL

        self.firstEncryptionLabel=tk.Label(self.loginWindow, text="Encryption Method:", font=(self.sFont, 14), bg=("black"), fg=(self.textColor))

        #place label

        self.firstEncryptionLabel.place(x=950,y=150)

        # ComboBox 1 for choosing method of encryption

        #list of encryption options 1

        self.optionsList1 = ["Caesar Cipher", "Base32", "Hex", "Book Cipher","AES"]

        #set a StringVar object in the window

        self.firstEncryptionOptionValue = tk.StringVar(self.loginWindow,'')

        #default value for encryption options 1

        self.firstEncryptionOptionValue.set("Choose Encryption Method")

        #establish combobox for choosing encryption 1

        self.firstEncryptionOptionMenu = ttk.Combobox(self.loginWindow, textvariable=  self.firstEncryptionOptionValue, state = 'readonly', foreground = "grey", font = (self.sFont, 16),width = 25, values = self.optionsList1)

        #place the combobox 1

        self.firstEncryptionOptionMenu.place(x=950,y=200)

        #if the combobox value is ever changed, call upon the function self.checkEncryptionResult1

        self.firstEncryptionOptionValue.trace\_add("write", self.checkEncryptionResult1)

    #checks the first combo box when it is changed

    def checkEncryptionResult1(self,var,index,mode):

        #destroy the encrypt button if it is there from a previous selection

        #FIX: the weird aes encrypt button error

        if self.isEncryptButton=="True":

            self.encryptButton.destroy()

            self.isEncryptButton="False"

        #if the previous selection was a caesar cipher or book cipher, meaning extra GUI parts, delete the GUI parts.

        if self.recentSelection1=="Caesar Cipher":

            self.secondEncryptionLabel.destroy()

            self.secondEncryptionOptionMenu.destroy()

        elif self.recentSelection1=="Book Cipher":

            self.fileUploadLabel.destroy()

            self.fileUploadButton.destroy()

        elif self.recentSelection1=="AES":

            self.secondEncryptionLabel.destroy()

            self.keyEntry.destroy()

        #get the user choice for combobox1

        self.userChoice1=self.firstEncryptionOptionMenu.get()

        #if the user chose caesar cipher, run this

        if self.userChoice1=="Caesar Cipher":

            self.recentSelection1="Caesar Cipher"

            self.secondEncryptionValueStart()

        #if the user chose book cipher, run this

        elif self.userChoice1=="Book Cipher":

            self.recentSelection1="Book Cipher"

            self.secondEncryptionValueStart()

        #if the user chose Base 32 or hex, run this

        elif self.userChoice1=="Base32":

            self.recentSelection1="Base32"

            self.encryptButton=tk.Button(self.loginWindow, text="ENCRYPT MESSAGE", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.encryptGUI)

            self.encryptButton.place(x=950,y=250)

            self.isEncryptButton="True"

        elif self.userChoice1=="Hex":

            self.recentSelection1="Hex"

            self.encryptButton=tk.Button(self.loginWindow, text="ENCRYPT MESSAGE", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.encryptGUI)

            self.encryptButton.place(x=950,y=250)

            self.isEncryptButton="True"

        elif self.userChoice1=="AES":

            self.recentSelection1="AES"

            self.secondEncryptionValueStart()

    #changes combo boxes based on new query selections from the navigation button

    def postNavigationEncryptionSetUp(self):

        #if the previous selection was a caesar cipher or book cipher, meaning extra GUI parts, delete the GUI parts.

        if self.recentSelection1=="Caesar Cipher":

            self.secondEncryptionLabel.destroy()

            self.secondEncryptionOptionMenu.destroy()

        elif self.recentSelection1=="Book Cipher":

            self.fileUploadLabel.destroy()

            self.fileUploadButton.destroy()

        elif self.recentSelection1=="AES":

            self.secondEncryptionLabel.destroy()

            self.keyEntry.destroy()

        #destroy the encrypt button if it is there from a previous selection

        if self.isEncryptButton=="True":

            self.encryptButton.destroy()

            self.isEncryptButton="False"

        #get the user choice for combobox1

        self.userChoice1=self.firstEncryptionOptionMenu.get()

        #if the user chose caesar cipher, run this

        if self.userChoice1=="Caesar Cipher":

            self.recentSelection1="Caesar Cipher"

            self.secondEncryptionValueStart()

            self.secondEncryptionOptionValue.set(self.tupleResult[6])

        #if the user chose book cipher, run this

        elif self.userChoice1=="Book Cipher":

            self.recentSelection1="Book Cipher"

            self.secondEncryptionValueStart()

        #if the user chose Base 32 or hex, run this

        elif self.userChoice1=="Base32":

            self.recentSelection1="Base32"

            self.encryptButton=tk.Button(self.loginWindow, text="ENCRYPT MESSAGE", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.encryptGUI)

            self.encryptButton.place(x=950,y=250)

            self.isEncryptButton="True"

        elif self.userChoice1=="Hex":

            self.recentSelection1="Hex"

            self.encryptButton=tk.Button(self.loginWindow, text="ENCRYPT MESSAGE", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.encryptGUI)

            self.encryptButton.place(x=950,y=250)

            self.isEncryptButton="True"

        elif self.userChoice1=="AES":

            self.recentSelection1="AES"

            self.secondEncryptionValueStart()

            self.keyVar.set(self.tupleResult[6])

        elif self.userChoice1=="Choose Encryption Method":

            self.recentSelection1=""

    #changes encryption side to the OG

    def revertEncryptionMenu(self):

        #if the previous selection was a caesar cipher or book cipher, meaning extra GUI parts, delete the GUI parts.

        if self.recentSelection1=="Caesar Cipher":

            self.secondEncryptionLabel.destroy()

            self.secondEncryptionOptionMenu.destroy()

            if self.isEncryptButton=="True":

                self.encryptButton.destroy()

                self.isEncryptButton="False"

        elif self.recentSelection1=="Book Cipher":

            self.fileUploadLabel.destroy()

            self.fileUploadButton.destroy()

        elif self.recentSelection1=="AES":

            self.secondEncryptionLabel.destroy()

            self.keyEntry.destroy()

        #destroy the encrypt button if it is there from a previous selection

        elif self.isEncryptButton=="True":

            self.encryptButton.destroy()

            self.isEncryptButton="False"

        #add support for AES

    #checks the second combo box when it is changed

    def checkEncryptionResult2(self,var,index,mode):

        self.encryptButton=tk.Button(self.loginWindow, text="ENCRYPT MESSAGE", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.encryptGUI)

        self.encryptButton.place(x=950,y=350)

        self.isEncryptButton="True"

    #enrypts text in plaintext field and puts it in the cipher text field

    def encryptGUI(self):

        self.cipherText=""

        self.userChoice1=self.firstEncryptionOptionMenu.get()

        self.plainText=self.plainTextField.get("1.0",'end-1c')

        if self.plainText=="":

            self.userMessage="No text to encode in Plain Text"

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

        elif self.userChoice1=="Caesar Cipher":

            self.userChoice2=int(self.secondEncryptionOptionMenu.get())

            self.cipherText=self.encrypt.caesar(self.plainText, self.userChoice2)

        elif self.userChoice1=="Hex":

            self.cipherText=self.encrypt.hexString(self.plainText)

        elif self.userChoice1=="Base32":

            self.cipherText=self.encrypt.b32(self.plainText)

        elif self.userChoice1=="Book Cipher" and self.plainText!="" and self.fileName!="":

            self.cipherText=self.encrypt.bookCipher(self.plainText,self.fileName)

        elif self.userChoice1=="Book Cipher" and self.fileName=="":

            self.userMessage="No Book File to encode using B-Cipher"

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

        elif self.userChoice1=="AES":

            self.userKey=self.keyEntry.get()

            self.cipherText=self.encrypt.AESencrypt(self.plainText, self.userKey)

        else:

            self.userMessage="Some error occured: please try again."

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

        self.cipherTextField["state"]="normal"

        self.cipherTextField.delete("1.0",'end')

        self.cipherTextField.insert("end",self.cipherText)

        self.cipherTextField["state"]="disabled"

    #sets up the second combo box for caesar cipher and text file upload button for book cipher

    def secondEncryptionValueStart(self):

        try:

            self.destroyTheButton()

        except:

            pass

        if self.userChoice1=="Caesar Cipher":

            #ComboBox 2's LABEL

            self.secondEncryptionLabel=tk.Label(self.loginWindow, text="Rotate by:", font=(self.sFont, 14), bg=("black"), fg=(self.textColor))

            #place label

            self.secondEncryptionLabel.place(x=950,y=250)

            # ComboBox 2 for choosing method of encryption

            #list of encryption options 2

            self.optionsList2 = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25]

            #set a StringVar object in the window

            self.secondEncryptionOptionValue = tk.StringVar(self.loginWindow,'')

            #default value for encryption options 2

            self.secondEncryptionOptionValue.set("Choose Rotation Key")

            #establish combobox for choosing encryption 2

            self.secondEncryptionOptionMenu = ttk.Combobox(self.loginWindow, textvariable=self.secondEncryptionOptionValue, state = 'readonly', foreground = "grey", font = (self.sFont, 16),width = 25, values = self.optionsList2)

            #place the combobox 2

            self.secondEncryptionOptionMenu.place(x=950,y=300)

            #if the combobox value is ever changed, call upon the function self.checkEncryptionResult2

            self.secondEncryptionOptionValue.trace\_add("write", self.checkEncryptionResult2)

        elif self.userChoice1=="Book Cipher":

            #

            self.fileUploadLabel=tk.Label(self.loginWindow, text="Upload a Text File of a Book", font=(self.sFont, 14), bg=("black"), fg=(self.textColor))

            self.fileUploadLabel.place(x=950,y=250)

            self.fileUploadButton = tk.Button(self.loginWindow, text='Text File Upload', command=self.uploadFile, font=(self.sFont, 14))

            self.fileUploadButton.place(x=1000,y=300)

        elif self.userChoice1=="AES":

            self.secondEncryptionLabel=tk.Label(self.loginWindow, text="AES encryption key:", font=(self.sFont, 14), bg=("black"), fg=(self.textColor))

            #place label

            self.secondEncryptionLabel.place(x=950,y=250)

            self.keyVar = tk.StringVar(self.loginWindow,'')

            self.keyEntry=tk.Entry(self.loginWindow, textvariable= self.keyVar, font=(self.sFont, 16), relief= "sunken")

            self.keyEntry.place(x=950,y=300)

            self.keyVar.trace\_add("write", self.afterKeyInput)

    def afterKeyInput(self, var, index, mode):

        self.encryptButton=tk.Button(self.loginWindow, text="ENCRYPT MESSAGE", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.encryptGUI)

        self.encryptButton.place(x=950,y=350)

        self.isEncryptButton="True"

    def destroyTheButton(self):

        self.encryptButton.destroy()

        self.isEncryptButton="False"

    #when the text-file-upload-button is pressed, initiate an tkinterupload thingy

    def uploadFile(self, event=None):

        #define the various file types that are used

        fileTypes = (('text files', '\*.txt'),('All files', '\*.\*'))

        #create a dialog that prompts for a text file

        self.fileName = filedialog.askopenfilename(title="Open a Text File", initialdir="/",filetypes=fileTypes)

        if self.fileName=="":

            self.userMessage="No File selected"

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

            return []

        #establishes encrypt button after file is uploaded?

        self.fileUploadButton.config(text="Uploaded ✓")

        self.encryptButton=tk.Button(self.loginWindow, text="ENCRYPT MESSAGE", font = (self.sFont, 18), borderwidth= 0, fg = (self.textColor), bg=("black"), command=self.encryptGUI)

        self.encryptButton.place(x=950,y=350)

        self.isEncryptButton="True"

    #destroy the widgets on the login UI

    def destroyLoginButtons(self):

        #destroy all of the login buttons, labels, entries on screen

        self.userMessage=""

        self.exitButton.destroy()

        self.loginButton.destroy()

        self.startSignupButton.destroy()

        self.usernameLabel.destroy()

        self.passwordLabel.destroy()

        self.userMessageLabel.destroy()

        self.usernameEntry.destroy()

        self.passwordEntry.destroy()

        self.aboutButton.destroy()

        self.helpButton.destroy()

    #destroy the widgets on the sign up UI

    def destroySignupButtons(self):

        self.backButton.destroy()

        self.signupButton.destroy()

        self.signupLabel.destroy()

        self.emailSULabel.destroy()

        self.newUsernameSULabel.destroy()

        self.newPasswordSULabel.destroy()

        self.emailSUEntry.destroy()

        self.usernameSUEntry.destroy()

        self.PasswordSUEntry.destroy()

        self.userMessage=""

        self.userMessageLabel.destroy()

        self.aboutButton.destroy()

        self.helpButton.destroy()

        self.b2MyPic.destroy()

        self.bMyPic.destroy()

    #destroy the widgets on the crypto UI

    def destroyCryptoButtons(self):

        self.userMessage=""

        self.backButton.destroy()

        self.userMessageLabel.destroy()

        self.idLabel.destroy()

        self.idText.destroy()

        self.plainTextLabel.destroy()

        self.cipherTextLabel.destroy()

        self.encryptionSectionLabel.destroy()

        self.plainTextFrame.destroy()

        self.cipherTextFrame.destroy()

        self.insertButton.destroy()

        self.updateButton.destroy()

        self.deleteButton.destroy()

        self.clearButton.destroy()

        self.backAllButton.destroy()

        self.back2Button.destroy()

        self.back1Button.destroy()

        self.forward1Button.destroy()

        self.forward2Button.destroy()

        self.forwardAllButton.destroy()

        self.bMyPic.destroy()

        self.firstEncryptionLabel.destroy()

        self.firstEncryptionOptionMenu.destroy()

        self.connectionButton.destroy()

        self.l2MyPic.destroy()

        if self.userChoice1=="Caesar Cipher":

            self.secondEncryptionLabel.destroy()

            self.secondEncryptionOptionMenu.destroy()

        if self.userChoice1=="Book Cipher":

            self.fileUploadLabel.destroy()

            self.fileUploadButton.destroy()

        if self.userChoice1=="AES":

            self.secondEncryptionLabel.destroy()

            self.keyEntry.destroy()

        if self.isEncryptButton=="True":

            self.encryptButton.destroy()

            self.isEncryptButton="False"

        if self.navIsDisabled==True:

            self.buttonCoverUp.destroy()

        self.menubar= tk.Menu(self.loginWindow)

        self.loginWindow.config(menu= self.menubar)

        self.fileMenu = tk.Menu(self.menubar, tearoff = 0)

        self.fileMenu.add\_command(label = "Exit", font = (self.sFont, 18), foreground = (self.textColor), command = self.loginWindow.destroy)

        self.menubar.add\_cascade(label = "File", font = (self.sFont, 18), foreground = (self.textColor), menu = self.fileMenu)

        self.helpMenu = tk.Menu(self.menubar, tearoff = 0)

        self.helpMenu.add\_command(label="About",font = (self.sFont, 18), foreground = (self.textColor), command = self.aboutWindow)

        self.menubar.add\_cascade(label = "Help", font = (self.sFont, 18), foreground = (self.textColor), menu = self.helpMenu)

    #destroy the window

    def destroyWindow(self):

        #destroy/exit out the window

        self.loginWindow.destroy()

    #makes a smaller window telling all about crypto

    def aboutWindow(self):

        messagebox.showinfo(title="About Crypto",

                            message="Crypto (short for Cryptography's Really Your Power To Own) is a database cryptographic encryption system developed by Michelle Luo.",

                            detail="This app allows you to encode messages and save them to your user specific account. Only YOU can access your saved encrypted messages.",

                            icon=messagebox.INFO

                            )

    #makes a smaller window providing helpful instructions

    def helpWindow(self):

        messagebox.showinfo(title="Basic Navigation of Crypto",

                            message="1. Sign Up for an Account \n2. Login with your Account \n3. Input your message in the PLAIN TEXT section,\n then choose an encryption method on the right.\n4. Once you are done filling out the encryption options, click ENCRYPT. \n5. You should now see an encrypted section of text in the CIPHER TEXT Field.\n\nYou can also insert, navigate, update, and delete these messages using the crypto database.",

                            detail="For more detailed instructions, please actually pay attention to Michelle Luo in her Master Project Presentation. :D",

                            icon=messagebox.INFO

                            )

    #hide the red user message, this function is called after 3 seconds elsewhere

    def hideUserMessage(self):

        #hide the user warning message on screen

        self.userMessage=""

        self.userMessageLabel.config(text=self.userMessage)

    #attempt to login to crypto

    def login(self):

        #gets user inputted username and password

        username=self.usernameEntry.get()

        password=self.passwordEntry.get()

        #sends it for a check through the checkPassword() in cryptodatabase.py

        self.loginResult=self.db.checkPassword(username, password)

        #if the credentials were incorrect, send a message to the user on screen

        if self.loginResult=="incorrect password":

            self.userMessage="Incorrect Credentials!"

            self.userMessageLabel.config(text=self.userMessage)

            #the user message disappears after 3 seconds

            self.userMessageLabel.after(3000, self.hideUserMessage)

        else:

            #if the credentials are right, open the crypto server with the password\_id in the server

            self.userID=self.loginResult

            self.openCrypto()

    #attempt to sign up with a new account

    def signup(self):

        #gets user inputted email, username, and password

        email=self.emailSUEntry.get()

        username=self.usernameSUEntry.get()

        password=self.PasswordSUEntry.get()

        if email=="" or username=="" or password=="":

            self.userMessage="Answer ALL Fields."

            self.userMessageLabel.config(text=self.userMessage)

            #the user message disappears after 3 seconds

            self.userMessageLabel.after(3000, self.hideUserMessage)

            return 1

        signupResult=self.db.startAccount(email,username,password)

        if signupResult=="Account taken":

            self.userMessage="This Username is already taken!"

            self.userMessageLabel.config(text=self.userMessage)

            #the user message disappears after 3 seconds

            self.userMessageLabel.after(3000, self.hideUserMessage)

        else:

            self.destroySignupButtons()

            self.createLoginButtons()

    #transitions from the sign up UI to the login UI

    def openLoginFromSU(self):

        #destroy the sign up GUI

        self.destroySignupButtons()

        #change window title to login

        self.loginWindow.title("Login")

        #implement login GUI buttons

        self.createLoginButtons()

    #transition from the crypto UI to the login UI

    def openLoginFromCrypto(self):

        #if the user disconnected earlier, connect again for the login part

        if self.connected==False:

            self.connected=True

            self.db.setConnection()

        #destroy crypto GUI

        self.destroyCryptoButtons()

         #change window title to login

        self.loginWindow.title("Login")

        #implement login GUI buttons

        self.createLoginButtons()

    #transition from login UI to the sign up UI

    def openSignup(self):

        #destroy the login GUI

        self.destroyLoginButtons()

        #change window title to signup

        self.loginWindow.title("Sign Up")

        #implement signup GUI buttons

        self.createSignupButtons()

    #transition from the login UI to the crypto UI

    def openCrypto(self):

        #destroy the login GUI

        self.destroyLoginButtons()

        #change window title to Crypto

        self.loginWindow.title("Crypto")

        self.createCryptoButtons()

    #connects or disconnects from db #CRYPTO UI ONLY

    def connectionGUI(self):

        if self.connected==True:

            self.connected=False

            self.connectionButton.config(text="Connect")

            self.db.breakConnection()

            self.plainTextField.delete("1.0","end")

            self.plainTextField["state"]="disabled"

            self.cipherTextField["state"]="normal"

            self.cipherTextField.delete("1.0","end")

            self.cipherTextField["state"]="disabled"

            self.firstEncryptionOptionValue.set("Choose Encryption Method")

            self.revertEncryptionMenu()

            self.firstEncryptionOptionMenu["state"]="disabled"

            self.entryNumber=0

            self.idText.configure(text="")

        else:

            self.connected=True

            self.connectionButton.config(text="Disconnect")

            self.db.setConnection()

            self.plainTextField["state"]="normal"

            self.cipherTextField["state"]="normal"

            self.firstEncryptionOptionMenu["state"]="readonly"

    #clears the plainText field, the cipherText field, and the encryption selection menu of any previous user inputted activity.

    def clearActivity(self):

        self.plainTextField.delete("1.0","end")

        self.cipherTextField["state"]="normal"

        self.cipherTextField.delete("1.0","end")

        self.cipherTextField["state"]="disabled"

        self.firstEncryptionOptionValue.set("Choose Encryption Method")

        self.revertEncryptionMenu()

    #inserts new entry into the crypto table #CRYPTO UI ONLY

    def insertCryptoGUI(self):

        if self.connected==False:

            self.userMessage="Not connected to the database"

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

            return 1

        newCryptoID=self.db.findMaxLength(self.userID)+1

        user1="NULL"

        user2="NULL"

        #something for the id label

        plainText=self.plainTextField.get("1.0", tk.END)

        cipherText=self.cipherTextField.get("1.0", tk.END)

        if self.userChoice1=="Caesar Cipher":

            user1="caesar"

            user2=self.secondEncryptionOptionMenu.get()

        elif self.userChoice1=="Book Cipher":

            user1="book"

            user2=self.fileName

        elif self.userChoice1=="Hex":

            user1="hex"

        elif self.userChoice1=="Base32":

            user1="base"

        elif self.userChoice1=="AES":

            user1="aes"

        elif self.userChoice1=="":

            user1="NULL"

            user2="NULL"

        else:

            pass

        result= self.db.insertCryptoDB(newCryptoID,plainText,cipherText,user1,user2,self.userID)

        if result==[]:

                    self.userMessage="Did not Insert: Error Occured"

                    self.userMessageLabel.config(text=self.userMessage)

                    #the user message disappears after 3 seconds

                    self.userMessageLabel.after(3000, self.hideUserMessage)

    #updates entry in the crypto table #CRYPTO UI ONLY

    def updateCryptoGUI(self):

        if self.connected==False:

            self.userMessage="Not connected to the database"

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

            return 1

        user1="NULL"

        user2="NULL"

        #something for the id label

        plainText=self.plainTextField.get("1.0", tk.END)

        cipherText=self.cipherTextField.get("1.0", tk.END)

        if self.userChoice1=="Caesar Cipher":

            user1="caesar"

            user2=self.secondEncryptionOptionMenu.get()

        elif self.userChoice1=="Book Cipher":

            user1="book"

            user2=self.fileName

        elif self.userChoice1=="Hex":

            user1="hex"

        elif self.userChoice1=="Base32":

            user1="base"

        elif self.userChoice1=="AES":

            user1="aes"

        elif self.userChoice1=="":

            user2="NULL"

        else:

            pass

        result= self.db.updateCryptoDB(self.entryNumber,plainText,cipherText,user1,user2,self.userID)

        if result==[]:

                    self.userMessage="Did not Update: Error Occured"

                    self.userMessageLabel.config(text=self.userMessage)

                    #the user message disappears after 3 seconds

                    self.userMessageLabel.after(3000, self.hideUserMessage)

    #deletes entry in the crypto table #CRYPTO UI ONLY

    def deleteCryptoGUI(self):

        if self.connected==False:

            self.userMessage="Not connected to the database"

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

            return 1

        if self.entryNumber != 0:

            userPromptDelete=messagebox.askyesno("WARNING", "Do you really want to delete this record?")

            if userPromptDelete == True:

                result=self.db.deleteCryptoDB(self.entryNumber, self.userID)

                if result==[]:

                    self.userMessage="Did not delete: Error Occured"

                    self.userMessageLabel.config(text=self.userMessage)

                    #the user message disappears after 3 seconds

                    self.userMessageLabel.after(3000, self.hideUserMessage)

                self.entryNumber=0

                self.idText.configure(text="")

                self.plainTextField.delete("1.0","end")

                self.cipherTextField["state"]="normal"

                self.cipherTextField.delete("1.0","end")

                self.cipherTextField["state"]="disabled"

                self.firstEncryptionOptionValue.set("Choose Encryption Method")

                self.postNavigationEncryptionSetUp()

            else:

                self.userMessage="Did not delete"

                self.userMessageLabel.config(text=self.userMessage)

                #the user message disappears after 3 seconds

                self.userMessageLabel.after(3000, self.hideUserMessage)

        else:

            self.userMessage="Cannot delete a non-existent Entry"

            self.userMessageLabel.config(text=self.userMessage)

            #the user message disappears after 3 seconds

            self.userMessageLabel.after(3000, self.hideUserMessage)

    #goes to first entry in crypto table #CRYPTO UI ONLY

    def backAllCryptoGUI(self):

        if self.connected==False:

            self.userMessage="Not connected to the database"

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

            return 1

        self.tupleResult=self.db.goToEntry(1, self.userID)

        if self.tupleResult=="No records available to navigate":

            self.userMessage=self.tupleResult

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

            return 1

        self.entryNumber=self.tupleResult[1]

        self.idText.configure(text=str(self.entryNumber))

        self.plainTextField.delete("1.0","end")

        self.plainTextField.insert("end", self.tupleResult[3])

        self.cipherTextField["state"]="normal"

        self.cipherTextField.delete("1.0","end")

        self.cipherTextField.insert("end", self.tupleResult[4])

        self.cipherTextField["state"]="disabled"

        if self.tupleResult[5]=="caesar":

            self.firstEncryptionOptionValue.set("Caesar Cipher")

        elif self.tupleResult[5]=="hex":

            self.firstEncryptionOptionValue.set("Hex")

        elif self.tupleResult[5]=="base":

            self.firstEncryptionOptionValue.set("Base32")

        elif self.tupleResult[5]=="book":

            self.fileName=self.tupleResult[6]

            self.firstEncryptionOptionValue.set("Book Cipher")

        elif self.tupleResult[5]=="aes":

            self.firstEncryptionOptionValue.set("AES")

        self.postNavigationEncryptionSetUp()

    #goes back two entries in crypto table #CRYPTO UI ONLY

    def back2CryptoGUI(self):

        if self.connected==False:

            self.userMessage="Not connected to the database"

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

            return 1

        self.entryNumber=self.entryNumber-2

        if self.entryNumber<1:

            self.entryNumber=1

        self.tupleResult=self.db.goToEntry(self.entryNumber, self.userID)

        if self.tupleResult=="No records available to navigate":

            self.userMessage=self.tupleResult

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

            return 1

        self.entryNumber=self.tupleResult[1]

        self.idText.configure(text=str(self.entryNumber))

        self.plainTextField.delete("1.0","end")

        self.plainTextField.insert("end", self.tupleResult[3])

        self.cipherTextField["state"]="normal"

        self.cipherTextField.delete("1.0","end")

        self.cipherTextField.insert("end", self.tupleResult[4])

        self.cipherTextField["state"]="disabled"

        if self.tupleResult[5]=="caesar":

            self.firstEncryptionOptionValue.set("Caesar Cipher")

        elif self.tupleResult[5]=="hex":

            self.firstEncryptionOptionValue.set("Hex")

        elif self.tupleResult[5]=="base":

            self.firstEncryptionOptionValue.set("Base32")

        elif self.tupleResult[5]=="book":

            self.fileName=self.tupleResult[6]

            self.firstEncryptionOptionValue.set("Book Cipher")

        elif self.tupleResult[5]=="aes":

            self.firstEncryptionOptionValue.set("AES")

        self.postNavigationEncryptionSetUp()

    #goes back one entry in the crypto table #CRYPTO UI ONLY

    def back1CryptoGUI(self):

        if self.connected==False:

            self.userMessage="Not connected to the database"

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

            return 1

        self.entryNumber=self.entryNumber-1

        if self.entryNumber<1:

            self.entryNumber=1

        self.tupleResult=self.db.goToEntry(self.entryNumber, self.userID)

        if self.tupleResult=="No records available to navigate":

            self.userMessage=self.tupleResult

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

            return 1

        self.entryNumber=self.tupleResult[1]

        self.idText.configure(text=str(self.entryNumber))

        self.plainTextField.delete("1.0","end")

        self.plainTextField.insert("end", self.tupleResult[3])

        self.cipherTextField["state"]="normal"

        self.cipherTextField.delete("1.0","end")

        self.cipherTextField.insert("end", self.tupleResult[4])

        self.cipherTextField["state"]="disabled"

        if self.tupleResult[5]=="caesar":

            self.firstEncryptionOptionValue.set("Caesar Cipher")

        elif self.tupleResult[5]=="hex":

            self.firstEncryptionOptionValue.set("Hex")

        elif self.tupleResult[5]=="base":

            self.firstEncryptionOptionValue.set("Base32")

        elif self.tupleResult[5]=="book":

            self.fileName=self.tupleResult[6]

            self.firstEncryptionOptionValue.set("Book Cipher")

        elif self.tupleResult[5]=="aes":

            self.firstEncryptionOptionValue.set("AES")

        self.postNavigationEncryptionSetUp()

    #goes forward one entry in the crypto table #CRYPTO UI ONLY

    def forward1CryptoGUI(self):

        if self.connected==False:

            self.userMessage="Not connected to the database"

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

            return 1

        maxNum=self.db.findMaxLength(self.userID)

        self.entryNumber=self.entryNumber+1

        if self.entryNumber>maxNum:

            self.entryNumber=maxNum

        self.tupleResult=self.db.goToEntry(self.entryNumber, self.userID)

        if self.tupleResult=="No records available to navigate":

            self.userMessage=self.tupleResult

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

            return 1

        self.entryNumber=self.tupleResult[1]

        self.idText.configure(text=str(self.entryNumber))

        self.plainTextField.delete("1.0","end")

        self.plainTextField.insert("end", self.tupleResult[3])

        self.cipherTextField["state"]="normal"

        self.cipherTextField.delete("1.0","end")

        self.cipherTextField.insert("end", self.tupleResult[4])

        self.cipherTextField["state"]="disabled"

        if self.tupleResult[5]=="caesar":

            self.firstEncryptionOptionValue.set("Caesar Cipher")

        elif self.tupleResult[5]=="hex":

            self.firstEncryptionOptionValue.set("Hex")

        elif self.tupleResult[5]=="base":

            self.firstEncryptionOptionValue.set("Base32")

        elif self.tupleResult[5]=="book":

            self.fileName=self.tupleResult[6]

            self.firstEncryptionOptionValue.set("Book Cipher")

        elif self.tupleResult[5]=="aes":

            self.firstEncryptionOptionValue.set("AES")

        self.postNavigationEncryptionSetUp()

    #goes forward two entry in the crypto table #CRYPTO UI ONLY

    def forward2CryptoGUI(self):

        if self.connected==False:

            self.userMessage="Not connected to the database"

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

            return 1

        maxNum=self.db.findMaxLength(self.userID)

        self.entryNumber=self.entryNumber+2

        if self.entryNumber>maxNum:

            self.entryNumber=maxNum

        self.tupleResult=self.db.goToEntry(self.entryNumber, self.userID)

        if self.tupleResult=="No records available to navigate":

            self.userMessage=self.tupleResult

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

            return 1

        self.entryNumber=self.tupleResult[1]

        self.idText.configure(text=str(self.entryNumber))

        self.plainTextField.delete("1.0","end")

        self.plainTextField.insert("end", self.tupleResult[3])

        self.cipherTextField["state"]="normal"

        self.cipherTextField.delete("1.0","end")

        self.cipherTextField.insert("end", self.tupleResult[4])

        self.cipherTextField["state"]="disabled"

        if self.tupleResult[5]=="caesar":

            self.firstEncryptionOptionValue.set("Caesar Cipher")

        elif self.tupleResult[5]=="hex":

            self.firstEncryptionOptionValue.set("Hex")

        elif self.tupleResult[5]=="base":

            self.firstEncryptionOptionValue.set("Base32")

        elif self.tupleResult[5]=="book":

            self.fileName=self.tupleResult[6]

            self.firstEncryptionOptionValue.set("Book Cipher")

        elif self.tupleResult[5]=="aes":

            self.firstEncryptionOptionValue.set("AES")

        self.postNavigationEncryptionSetUp()

    #goes to last entry in the crypto table #CRYPTO UI ONLY

    def forwardAllCryptoGUI(self):

        if self.connected==False:

            self.userMessage="Not connected to the database"

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

            return 1

        maxLength=self.db.findMaxLength(self.userID)

        self.tupleResult=self.db.goToEntry(maxLength, self.userID)

        if self.tupleResult=="No records available to navigate":

            self.userMessage=self.tupleResult

            self.userMessageLabel.config(text=self.userMessage)

            self.userMessageLabel.after(3000, self.hideUserMessage)

            return 1

        self.entryNumber=self.tupleResult[1]

        self.idText.configure(text=str(self.entryNumber))

        self.plainTextField.delete("1.0","end")

        self.plainTextField.insert("end", self.tupleResult[3])

        self.cipherTextField["state"]="normal"

        self.cipherTextField.delete("1.0","end")

        self.cipherTextField.insert("end", self.tupleResult[4])

        self.cipherTextField["state"]="disabled"

        if self.tupleResult[5]=="caesar":

            self.firstEncryptionOptionValue.set("Caesar Cipher")

        elif self.tupleResult[5]=="hex":

            self.firstEncryptionOptionValue.set("Hex")

        elif self.tupleResult[5]=="base":

            self.firstEncryptionOptionValue.set("Base32")

        elif self.tupleResult[5]=="book":

            self.firstEncryptionOptionValue.set("Book Cipher")

        elif self.tupleResult[5]=="aes":

            self.firstEncryptionOptionValue.set("AES")

        self.postNavigationEncryptionSetUp()

'''

Michelle Luo

Cmdr. Schenk

AP Computer Science

Master Project POCO (cryptopoco.py)

Spring of 2023

'''

class CryptoPoco:

    def \_\_init\_\_(self, accountID, username, password, id\_user\_specific\_in\_tableID, plainText, cipherText, encryption1,encryption2):

        self.accountID=accountID

        self.username=username

        self.password=password

        self.id\_user\_specific\_in\_tableID=id\_user\_specific\_in\_tableID

        self.plainText=plainText

        self.cipherText=cipherText

        self.encryption1=encryption1

        self.encryption2=encryption2

    def getAccountID(self):

        return self.accountID

    def setAccountID(self, accountID):

        this.accountID=accountID

    def getusername(self):

        return self.username

    def setusername(self, username):

        this.username=username

    def getpassword(self):

        return self.password

    def setpassword(self, password):

        self.password=password

    def getId\_user\_specific\_in\_tableID(self):

        return self.id\_user\_specific\_in\_tableID

    def setId\_user\_specific\_in\_tableID(self, id\_user\_specific\_in\_tableID):

        this.id\_user\_specific\_in\_tableID=id\_user\_specific\_in\_tableID

    def getPlainText(self):

        return self.plainText

    def setPlainText(self, plainText):

        this.plainText=plainText

    def getCipherText(self):

        return self.cipherText

    def setCipherText(self, cipherText):

        this.cipherText=cipherText

    def getEncryption1(self):

        return self.encryption1

    def setEncryption1(self, encryption1):

        this.encryption1=encryption1

    def getEncryption2(self):

        return self.encryption2

    def setEncryption2(self, encryption2):

        this.encryption2=encryption2

Schemas used:

Graphical user interface, application

Description automatically generated

Table

Description automatically generated

Screenshots of it running:

Graphical user interface

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated