**Secured Password Manager**

**Source Code**

import os, re, struct, json, tkinter as tk

from tkinter import ttk

from tkinter import messagebox

from tkinter.simpledialog import askstring

from math import gcd

import pyperclip

import string

import random

#############################################

#       Global Variables / FLAGS

#############################################

USER\_DB\_FILE = "user\_data.json" # db file path

CREDENTIALS\_FILE = "credentials.json" # creds file path

CURRENT\_USER = None # active user

#############################################

#               HASHING

#############################################

def leftRotate(x: int, c: int) -> int:

    """

    Performs a circular left bitwise rotation on a 32-bit integer.

    This function rotates the bits of the input integer to the left by a

    specified number of places, wrapping the overflowed bits back to the

    right end. The operation is constrained to 32 bits.

    Args:

        x (int): The 32-bit integer to rotate.

        c (int): The number of bit positions to rotate.

    Returns:

        int: The result of the left rotation as a 32-bit integer.

    """

    return (x << c | x >> (32 - c)) & 0xFFFFFFFF

def md5(key: str) -> str:

    """

    Computes the MD5 hash of an input string.

    This function implements the MD5 hashing algorithm to produce

    a 128-bit hash value represented as a 32-character hexadecimal

    string for a given input string.

    It performs padding, initializes state variables, processes data

    in 512-bit chunks, and applies bitwise operations and transformations

    to compute the hash.

    Args:

        key (str): The input string to hash.

    Returns:

        str: The resulting hash value as a hexadecimal string.

    """

    # Shift Amounts: number of bits to left-rotate in each step of the MD5 transformation

    S = [

        7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,

        5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20,

        4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,

        6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21,

    ]

    # K Constants: set of 64 precomputed constants used in the main MD5 algorithm loop

    K = [

        int(abs(struct.unpack("f", struct.pack("f", i))[0]) \* 2\*\*32) & 0xFFFFFFFF

        for i in range(1, 65)

    ]

    # Initial hash values

    A = 0x67452301

    B = 0xefcdab89

    C = 0x98badcfe

    D = 0x10325476

    # Preprocessing

    original\_length = len(key) \* 8

    key = bytearray(key, 'utf-8')

    key.append(0x80)

    while (len(key) \* 8) % 512 != 448:

        key.append(0)

    key += struct.pack('<Q', original\_length)

    # Process each 512-bit chunk

    for i in range(0, len(key), 64):

        chunk = key[i:i + 64]

        M = [struct.unpack('<I', chunk[j:j + 4])[0] for j in range(0, 64, 4)]

        a, b, c, d = A, B, C, D

        for i in range(64):

            if 0 <= i <= 15:

                f = (b & c) | (~b & d)

                g = i

            elif 16 <= i <= 31:

                f = (d & b) | (~d & c)

                g = (5 \* i + 1) % 16

            elif 32 <= i <= 47:

                f = b ^ c ^ d

                g = (3 \* i + 5) % 16

            elif 48 <= i <= 63:

                f = c ^ (b | ~d)

                g = (7 \* i) % 16

            temp = (a + f + K[i] + M[g]) & 0xFFFFFFFF

            temp = leftRotate(temp, S[i])

            temp = (temp + b) & 0xFFFFFFFF

            a, b, c, d = d, temp, b, c

        A = (A + a) & 0xFFFFFFFF

        B = (B + b) & 0xFFFFFFFF

        C = (C + c) & 0xFFFFFFFF

        D = (D + d) & 0xFFFFFFFF

    # Produce the final hash value (little-endian)

    return ''.join(f'{x:02x}' for x in struct.pack('<4I', A, B, C, D))

#############################################

#           ENCRYPTION/DECRYPTION

#############################################

def modInverse(e: int, phi: int) -> int:

    """

    Finds the modular multiplicative inverse of e under modulo phi.

    Uses the Extended Euclidean Algorithm.

    Args:

        e (int): The number to find the inverse for.

        phi (int): The modulo.

    Returns:

        int: The modular inverse of e modulo phi.

    """

    t, new\_t = 0, 1

    r, new\_r = phi, e

    while new\_r != 0:

        quotient = r // new\_r

        t, new\_t = new\_t, t - quotient \* new\_t

        r, new\_r = new\_r, r - quotient \* new\_r

    if r > 1:

        raise ValueError("e is not invertible")

    if t < 0:

        t += phi

    return t

def generateRSAkeys() -> tuple[tuple[int, int], tuple[int, int]]:

    """

    Generates RSA keys manually.

    Returns:

        tuple: (public\_key, private\_key, n)

    """

    # Step 1: Choose two prime numbers

    # Example small prime numbers

    p = 61

    q = 53

    n = p \* q  # Modulus

    phi = (p - 1) \* (q - 1)  # Euler's Totient

    # Step 2: Choose e such that gcd(e, phi) = 1 and 1 < e < phi

    e = 17  # Commonly used public exponent

    if gcd(e, phi) != 1:

        raise ValueError("e and phi(n) are not coprime.")

    # Step 3: Compute d, the modular inverse of e

    d = modInverse(e, phi)

    public\_key = (e, n)

    private\_key = (d, n)

    return public\_key, private\_key

def RSAencrypt(plaintext: str, public\_key: tuple) -> str:

    """

    Encrypts a plaintext string using RSA.

    Args:

        plaintext (str): The plaintext to encrypt.

        public\_key (tuple): The public key (e, n).

    Returns:

        str: The encrypted message as a string of list of integers.

    """

    e, n = public\_key

    encrypted = [(ord(char) \*\* e) % n for char in plaintext]

    encrypted = ",".join(map(str, encrypted)) # convert list of integers to a string of csv

    return encrypted

def RSAdecrypt(encrypted\_message: str, private\_key: tuple) -> str:

    """

    Decrypts an encrypted message using RSA.

    Args:

        encrypted\_message (str): The encrypted message as a string of list of integers.

        private\_key (tuple): The private key (d, n).

    Returns:

        str: The decrypted plaintext.

    """

    d, n = private\_key

    encrypted\_message = list(map(int, encrypted\_message.split(','))) # revert to list of integers

    decrypted = ''.join([chr((char \*\* d) % n) for char in encrypted\_message])

    return decrypted

#############################################

#               STORAGE

#############################################

def initializeStorage(username: str) -> None:

    """Initializes 'credentials.json' with a username:empty array pair, if it doesn't exist."""

    # Check if the credentials.json file exists

    if os.path.exists(CREDENTIALS\_FILE):

        with open(CREDENTIALS\_FILE, "r") as j:

            data = json.load(j)

    else:

        data = {}

    # Add the username with an empty array (if it doesn't already exist)

    if username not in data:

        data[username] = []

    # Write the updated data back into the file

    with open(CREDENTIALS\_FILE, "w") as j:

        json.dump(data, j, indent=4)

def loadCredentials(username: str) -> list:

    """

    This function loads the credentials for a given username from the 'credentials.json' file.

    Args:

        username (str): The username for which to load the credentials.

    Returns:

        list: The list of credentials for the given username, or an empty list if the username is not found.

    """

    try:

        # Open the credentials.json file and load the data

        with open(CREDENTIALS\_FILE, "r") as json\_file:

            data = json.load(json\_file)

        # Return the list of credentials for the given username, or an empty list if not found

        return data.get(username, [])

    except FileNotFoundError:

        print("\*\*\* ERROR: credentials.json file not found.")

        return []

    except json.JSONDecodeError:

        print("\*\*\* ERROR: Failed to decode JSON data.")

        return []

def saveCredentials(credentials\_list) -> None:

    """

    This function saves the modified credentials list for the current user back to the JSON file.

    """

    try:

        # Load the current credentials data from the JSON file

        with open(CREDENTIALS\_FILE, "r") as json\_file:

            credentials\_data = json.load(json\_file)

        # Check if the current user exists in the data

        if CURRENT\_USER not in credentials\_data:

            # If the user doesn't exist, initialize their credentials list

            credentials\_data[CURRENT\_USER] = []

        # Update the credentials list for the current user

        credentials\_data[CURRENT\_USER] = credentials\_list

        # Save the updated data back to the file

        with open(CREDENTIALS\_FILE, "w") as json\_file:

            json.dump(credentials\_data, json\_file, indent=4)

    except (FileNotFoundError, json.JSONDecodeError) as e:

        print(f"Error: {e}")

def appendCredential(website: str, username: str, password: str, public\_key: tuple[int], private\_key: tuple[int]) -> None:

    """

    This function takes the new credential details as input, creates a new dictionary/object, appends it to the list, and saves the updated list.

    """

    encryption\_key = public\_key[0]

    decryption\_key = private\_key[0]

    modulusN = public\_key[1]

    credentials\_list = loadCredentials(CURRENT\_USER)

    new\_credential = {

        "website": website,

        "username": username,

        "password": password,

        "publickey": encryption\_key,

        "privatekey": decryption\_key,

        "modulus": modulusN

    }

    credentials\_list.append(new\_credential)

    saveCredentials(credentials\_list)

def deleteCredential(index) -> None:

    """

    This function takes the index of the credential to be deleted, removes it from the list, and saves the updated list.

    """

    credentialsList = loadCredentials(CURRENT\_USER)

    del credentialsList[index]

    saveCredentials(credentialsList)

#############################################

#               UTILS

#############################################

def ifUsersExist(file\_path: str) -> bool:

    """

    Checks if the JSON file contains any users.

    Args:

        file\_path (str): Path to the JSON file.

    Returns:

        bool: True if at least one user exists, otherwise False.

    """

    if not os.path.exists(file\_path):

        return False

    try:

        with open(file\_path, 'r') as file:

            data = json.load(file)

            return bool(data)  # Return True if the dictionary is not empty

    except (FileNotFoundError, json.JSONDecodeError):

        return False

def storeLoginCredentials(user\_data: dict) -> None:

    """

    Stores user data (username and hashed password) in a JSON file.

    Args:

        user\_data (dict): A dictionary containing the username and hashed password.

    """

    if os.path.exists(USER\_DB\_FILE):

        with open(USER\_DB\_FILE, "r") as file:

            data = json.load(file)  # Load existing data as a dictionary

    else:

        data = {}  # Initialize as an empty dictionary

    # Check if the username already exists

    if user\_data["username"] in data:

        raise ValueError("Username already exists. Choose a different one.")

    # Add new user data (username:password\_hash pair)

    data[user\_data["username"]] = user\_data["password\_hash"]

    initializeStorage(user\_data["username"])

    # Write the updated data back to the file

    with open(USER\_DB\_FILE, "w") as file:

        json.dump(data, file, indent=4)

def validatePasskey(passkey: str) -> tuple[bool, str]:

    """

    Validates a passkey based on the following criteria:

    - Minimum length: 8 characters

    - Maximum length: 25 characters

    - At least one uppercase letter

    - At least one number

    - At least one special character

    Args:

        passkey (str): The passkey to validate.

    Returns:

        tuple: A tuple containing a boolean (True if valid) and a string (reason if invalid, or "Valid").

    """

    # Check length

    if not (8 <= len(passkey) <= 25):

        return False, "Passkey must be between 8 and 25 characters."

    # Check for at least one uppercase letter

    if not any(char.isupper() for char in passkey):

        return False, "Passkey must include at least one uppercase letter."

    # Check for at least one digit

    if not any(char.isdigit() for char in passkey):

        return False, "Passkey must include at least one number."

    # Check for at least one special character

    if not re.search(r"[!#\"$%&'()\*+,\-./:;<=>?@[\\\]^\_`{|}~]", passkey):

        return False, "Passkey must include at least one special character."

    return True, "Valid"

def isUserValid(username: str, password: str) -> bool:

    """

    Validates the user by checking the username and hashed password in the JSON file.

    Args:

        username (str): The entered username.

        password (str): The entered password.

    Returns:

        bool: True if the username and password match the stored data, False otherwise.

    """

    try:

        with open(USER\_DB\_FILE, "r") as file:

            users = json.load(file)  # Load the data from the JSON file

        # Check if the username exists in the data

        if username not in users:

            print("\*\*\* ALERT: Username not found.")

            return False

        # Get the stored hashed password for the username

        stored\_hashed\_password = users[username]

        # Hash the entered password and compare it with the stored hashed password

        entered\_hashed\_password = md5(password)

        return entered\_hashed\_password == stored\_hashed\_password

    except (FileNotFoundError, json.JSONDecodeError) as e:

        print(f"\*\*\* ERROR: Failed to read or parse {USER\_DB\_FILE}. {e}")

        return False

def checkPasswordStrength(password) -> str:

    # Policies

    min\_length = 8

    ideal\_length = 15

    common\_passwords = {"123456", "password", "123456789", "qwerty", "abc123", "letmein"} # can extend it as per use in future

    # Checks

    length = len(password)

    contains\_upper = bool(re.search(r'[A-Z]', password))

    contains\_lower = bool(re.search(r'[a-z]', password))

    contains\_digit = bool(re.search(r'[0-9]', password))

    contains\_special = bool(re.search(r'[!@#$%^&\*(),.?":{}|<>]', password))

    is\_common = password in common\_passwords

    # Classification

    if is\_common:

        return "Weak (commonly used password)"

    elif length < min\_length:

        return "Weak (too short)"

    elif length < ideal\_length and (not contains\_upper or not contains\_special or not contains\_digit):

        return "Average (short with minimal complexity)"

    elif length >= ideal\_length and (contains\_upper or contains\_special or contains\_digit):

        return "Strong (long and complex)"

    else:

        return "Average (improved but could be stronger)"

def analyzePasswords() -> str:

    """Decrypts passwords and analyzes their strength."""

    credentials = loadCredentials(CURRENT\_USER)

    if not credentials:

        messagebox.showinfo("No Credentials", "No credentials found.")

        return

    weak\_passwords = []

    average\_passwords = []

    strong\_passwords = []

    for cred in credentials:

        decrypted\_password = RSAdecrypt(cred['password'], (cred['privatekey'], cred['modulus']))

        strength = checkPasswordStrength(decrypted\_password)

        website = cred['website']  # Get website from the credential data

        if "Weak" in strength:

            weak\_passwords.append(website)  # Store website name for weak passwords

        elif "Average" in strength:

            average\_passwords.append(website)

        elif "Strong" in strength:

            strong\_passwords.append(website)

    # Formatting the message

    message\_parts = []

    if weak\_passwords:

        message\_parts.append(f"Weak passwords at: {', '.join(weak\_passwords)}")

    if average\_passwords:

        message\_parts.append(f"Average passwords at: {', '.join(average\_passwords)}")

    if strong\_passwords:

        message\_parts.append(f"Strong passwords at: {', '.join(strong\_passwords)}")

    message = "\n".join(message\_parts) if message\_parts else "All passwords are strong!"

    messagebox.showinfo("Password Strength Analysis", message)

################## NEW FEATURES ##################

# Function to check password strength

def checkPasswordStrength(password: str) -> tuple:

    """Checks the strength of the password and returns strength and a reason."""

    if len(password) == 0:

        return "Empty", "None"

    elif len(password) < 8:

        return "Weak", "Password too short"

    elif not any(char.isdigit() for char in password):

        return "Medium", "Password needs a digit"

    elif not any(char.isupper() for char in password):

        return "Medium", "Password needs an uppercase letter"

    elif not any(char in r"[!#\"$%&'()\*+,\-./:;<=>?@[\]^\_`{|}~]" for char in password):

        return "Medium", "Password needs a special character"

    return "Strong", "Good password"

# Function to update the strength meter in real-time

def update\_strength\_meter(password: str, strength\_label: tk.Label, progress: ttk.Progressbar) -> None:

    """Updates the password strength label and progress bar."""

    strength, reason = checkPasswordStrength(password)

    # Update the strength label

    strength\_label.config(text=f"Strength: {strength} ({reason})")

    # Update the progress bar based on strength

    if strength == "Empty":

        progress['value'] = 0

        progress.config(style="danger.Horizontal.TProgressbar")

    elif strength == "Weak":

        progress['value'] = 33

        progress.config(style="danger.Horizontal.TProgressbar")

    elif strength == "Medium":

        progress['value'] = 66

        progress.config(style="warning.Horizontal.TProgressbar")

    elif strength == "Strong":

        progress['value'] = 100

        progress.config(style="success.Horizontal.TProgressbar")

# Function to generate a random password

def generate\_password(length=12) -> str:

    """Generates a random password with specified length."""

    # Define the characters pool: uppercase, lowercase, digits, and punctuation

    lowercase = string.ascii\_lowercase

    uppercase = string.ascii\_uppercase

    special\_chars = string.punctuation

    digits = string.digits

    password = [random.choice(lowercase), random.choice(uppercase), random.choice(special\_chars), random.choice(digits)]

    # Fill the rest of the characters with random characters

    all\_chars = uppercase + special\_chars + digits + lowercase

    for \_ in range(length - 4):

       password.append(random.choice(all\_chars))

    # Shuffle the list to avoid the first four characters always being in the same character set order

    random.shuffle(password)

    # Join the characters into a single string

    password = ''.join(password)

    return password

# Function to generate and display password

def generate\_and\_display\_password(password\_entry: tk.Entry, password\_label: tk.Label, strength\_label: tk.Label, progress: ttk.Progressbar, length=12) -> None:

    """Generates a password, displays it in the label and entry, and updates the strength meter."""

    generated\_password = generate\_password(length)

    password\_entry.delete(0, tk.END)

    password\_entry.insert(0, generated\_password)

    # Update the strength meter and label

    update\_strength\_meter(generated\_password, strength\_label, progress)

    # Display the generated password

    password\_label.config(text=f"Generated Password: {generated\_password}")

    return generated\_password

def copy\_password\_to\_clipboard(password: str) -> None:

    """Copies the generated password to the clipboard."""

    pyperclip.copy(password)  # Copy the password to clipboard

    messagebox.showinfo("Copied", "Password copied to clipboard!")

def setup\_styles() -> None:

    """Sets up the styles for the progress bar."""

    style = ttk.Style()

    style.configure("danger.Horizontal.TProgressbar",

                    thickness=20,

                    background="red")

    style.configure("warning.Horizontal.TProgressbar",

                    thickness=20,

                    background="yellow")

    style.configure("success.Horizontal.TProgressbar",

                    thickness=20,

                    background="green")

#############################################

# GUI FUNCTIONS

#############################################

def setup\_user() -> None:

    """Handles user setup through the GUI."""

    """Sets up a new user with a generated or custom password."""

    user\_window = tk.Toplevel(root)

    user\_window.title("Set Up User")

    username\_label = tk.Label(user\_window, text="Enter Username:")

    username\_label.pack(pady=5)

    username\_entry = tk.Entry(user\_window)

    username\_entry.pack(pady=5)

    # Label for the generated password

    password\_label = tk.Label(user\_window, text="Generated Password: Not generated yet")

    password\_label.pack(pady=5)

    password\_entry = tk.Entry(user\_window, show="\*")

    password\_entry.pack(pady=5)

    # Create the strength label and progress bar for the strength meter

    strength\_label = tk.Label(user\_window, text="Strength: Not checked")

    strength\_label.pack(pady=5)

    progress = ttk.Progressbar(user\_window, style="danger.Horizontal.TProgressbar", length=200, mode='determinate')

    progress.pack(pady=5)

    # Generate password button

    def generate\_password\_and\_display():

        generated\_password = generate\_and\_display\_password(password\_entry, password\_label, strength\_label, progress)

        return generated\_password

    generate\_button = tk.Button(user\_window, text="Generate Password", command=generate\_password\_and\_display)

    generate\_button.pack(pady=5)

    # Copy password button

    def copy\_password():

        password = password\_entry.get()

        copy\_password\_to\_clipboard(password)

    copy\_button = tk.Button(user\_window, text="Copy Password", command=copy\_password)

    copy\_button.pack(pady=5)

    # Update strength meter on password change

    def on\_password\_change(\*args):

        password = password\_entry.get()  # Get the password from the entry widget

        update\_strength\_meter(password, strength\_label, progress)

    password\_entry.bind("<KeyRelease>", on\_password\_change)

    # Submit the user data

    def validate\_and\_store\_user():

        username = username\_entry.get()

        password = password\_entry.get()

        # Validate password strength

        is\_valid, reason = validatePasskey(password)

        if not is\_valid:

            messagebox.showerror("Error", reason)

            return

        hashed\_password = md5(password)

        user\_data = {"username": username, "password\_hash": hashed\_password}

        try:

            storeLoginCredentials(user\_data)

            messagebox.showinfo("Setup Complete", "User setup successfully. You can now log in.")

            user\_window.destroy()  # Close the password setup window

            #show\_main\_interface()  # Go back to main interface

        except Exception as e:

            messagebox.showerror("Error", f"Failed to set up user: {e}")

    # Add the 'Submit' button to submit the credentials

    submit\_button = tk.Button(user\_window, text="Submit", command=validate\_and\_store\_user)

    submit\_button.pack(pady=10)

def validate\_user() -> None:

    """Handles user validation through the GUI."""

    global CURRENT\_USER

    username = askstring("Login", "Enter your username:")

    password = askstring("Login", "Enter your password:", show="\*")

    if not username or not password:

        messagebox.showerror("Error", "Username and password are required.")

        return

    if isUserValid(username, password):

        CURRENT\_USER = username

        messagebox.showinfo("Success", "User validated successfully.")

        show\_main\_interface()

        # Analyze passwords after successful login

        strength\_summary = analyzePasswords()

        if strength\_summary:

            messagebox.showinfo("Password Analysis", strength\_summary)

    else:

        messagebox.showerror("Error", "Invalid username or password.")

def logout(main\_window: tk.Toplevel) -> None:

    """Logs out the current user and returns to the login/setup screen."""

    global CURRENT\_USER

    CURRENT\_USER = None  # Reset the current user

    main\_window.destroy()  # Close the main interface window

    root.deiconify()  # Redisplay the root login/setup window

def show\_main\_interface() -> None:

    """Displays the main interface after user validation."""

    main\_window = tk.Toplevel(root)

    main\_window.title("Credential Manager")

    tk.Button(main\_window, text="Add Credential", command=add\_credential).pack(pady=5)

    tk.Button(main\_window, text="View Credentials", command=view\_credentials).pack(pady=5)

    tk.Button(main\_window, text="Delete Credential", command=delete\_credential).pack(pady=5)

    tk.Button(main\_window, text="Analyze Passwords", command=analyzePasswords).pack(pady=5)

    tk.Button(main\_window, text="Logout", command=lambda: logout(main\_window)).pack(pady=20)

    tk.Button(main\_window, text="Exit", command=root.quit).pack(pady=20)

def add\_credential() -> None:

    """Adds a new credential with password strength meter."""

    # Open a new window to add credentials

    credential\_window = tk.Toplevel(root)

    credential\_window.title("Add Credential")

    # Fields to input the website, username, and password

    website\_label = tk.Label(credential\_window, text="Enter Website:")

    website\_label.pack(pady=5)

    website\_entry = tk.Entry(credential\_window)

    website\_entry.pack(pady=5)

    username\_label = tk.Label(credential\_window, text="Enter Username:")

    username\_label.pack(pady=5)

    username\_entry = tk.Entry(credential\_window)

    username\_entry.pack(pady=5)

    # Password fields

    password\_label = tk.Label(credential\_window, text="Enter Password:")

    password\_label.pack(pady=5)

    password\_entry = tk.Entry(credential\_window, show="\*")

    password\_entry.pack(pady=5)

    # Strength meter label

    strength\_label = tk.Label(credential\_window, text="Strength: Not checked")

    strength\_label.pack(pady=5)

    progress = ttk.Progressbar(credential\_window, style="danger.Horizontal.TProgressbar", length=200, mode='determinate')

    progress.pack(pady=5)

    # Function to update password strength meter

    def on\_password\_change(\*args):

        password = password\_entry.get()  # Get the password from the entry widget

        update\_strength\_meter(password, strength\_label, progress)

    password\_entry.bind("<KeyRelease>", on\_password\_change)

    def submit\_credential():

        website = website\_entry.get()

        username = username\_entry.get()

        password = password\_entry.get()

        # Check if any field is empty

        if not website or not username or not password:

            messagebox.showerror("Error", "All fields are required.")

            return

        # Validate password strength

        strength, reason = checkPasswordStrength(password)

        summary = strength + " Password! " +  reason

        messagebox.showinfo("Password Analysis", summary)

        #return

        # Encrypt the password using RSA (as per the existing logic)

        public\_key, private\_key = generateRSAkeys()

        encrypted\_password = RSAencrypt(password, public\_key)

        # Store the credential (you will need to modify the storage function as required)

        appendCredential(website, username, encrypted\_password, public\_key, private\_key)

        messagebox.showinfo("Success", "Credential added successfully!")

        credential\_window.destroy()  # Close the add credential window

    # Submit button to add the credential

    submit\_button = tk.Button(credential\_window, text="Add Credential", command=submit\_credential)

    submit\_button.pack(pady=10)

def view\_credentials() -> None:

    """Displays all saved credentials with search functionality."""

    def search\_credentials():

        """Filters and displays credentials based on the search query."""

        query = search\_entry.get().lower()  # Get search query and convert to lowercase

        filtered\_credentials = []

        # Load all credentials

        credentials = loadCredentials(CURRENT\_USER)

        # Filter credentials by website or username

        for cred in credentials:

            if query in cred['website'].lower() or query in cred['username'].lower():

                filtered\_credentials.append(cred)

        # Display filtered results

        display\_credentials(filtered\_credentials)

        # Bind the search functionality to the key release event for dynamic search

        search\_entry.bind("<KeyRelease>", lambda event: search\_credentials())

    def display\_credentials(credentials):

        """Displays the filtered credentials."""

        if not credentials:

            messagebox.showinfo("No Credentials", "No matching credentials found.")

            return

        credential\_list = "\n".join(

            [f"Website: {cred['website']}, Username: {cred['username']}, Password: {RSAdecrypt(cred['password'], (cred['privatekey'], cred['modulus']))}" for cred in credentials]

        )

        credentials\_label.config(text=credential\_list)

    # Create a new window to view credentials

    credentials\_window = tk.Toplevel(root)

    credentials\_window.title("View Credentials")

    # Add a search bar at the top

    search\_label = tk.Label(credentials\_window, text="Search Credentials (Website or Username):")

    search\_label.pack(pady=10)

    search\_entry = tk.Entry(credentials\_window, width=30)

    search\_entry.pack(pady=5)

    # Search button to filter credentials

    search\_button = tk.Button(credentials\_window, text="Search", command=search\_credentials)

    search\_button.pack(pady=5)

    # Display all credentials initially

    credentials\_label = tk.Label(credentials\_window, text="", justify=tk.LEFT)

    credentials\_label.pack(pady=10)

    # Load and display all credentials initially

    credentials = loadCredentials(CURRENT\_USER)

    display\_credentials(credentials)

def delete\_credential() -> None:

    """Deletes a credential by index."""

    index = askstring("Delete Credential", "Enter credential index to delete:")

    try:

        index = int(index) - 1

        credentials = loadCredentials(CURRENT\_USER)

        if not credentials:

            messagebox.showinfo("No Credentials", "No credentials to delete.")

            return

        if index < 0 or index >= len(credentials):

            raise IndexError

        deleteCredential(index)

        messagebox.showinfo("Success", "Credential deleted successfully.")

    except ValueError:

        messagebox.showerror("Error", "Invalid input. Enter a numeric index.")

    except IndexError:

        messagebox.showerror("Error", "Invalid index. Please select a valid credential.")

    except Exception as e:

        messagebox.showerror("Error", f"An unexpected error occurred: {e}")

#############################################

# APPLICATION ENTRY POINT

#############################################

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

    # Create the root window

    root = tk.Tk()

    root.title("User Validation - PW Manager")

    # Check if users exist in the USER\_DB\_FILE

    if not ifUsersExist(USER\_DB\_FILE):

        # No users found, ask the user to set up a new login

        tk.Label(root, text="No logins saved. Please set up a new login.").pack(pady=10)

        tk.Button(root, text="Set Up User", command=setup\_user).pack(pady=20)

    else:

        # Users exist, prompt the user to log in or set up a new user

        tk.Label(root, text="Please log in or set up a new user.").pack(pady=10)

        tk.Button(root, text="Log In", command=validate\_user).pack(pady=5)

        tk.Button(root, text="Set Up New User", command=setup\_user).pack(pady=5)

        tk.Button(root, text="Exit", command=root.quit).pack(pady=5)

    # Start the Tkinter event loop

    root.mainloop()