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
```

```python
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
##############################################
```

```python
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

##############################################
```

```python
#               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
```

```python
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()
```