Project Title: Password-Manager (python based)

Group Member 1:

Roll no: 22k 4780

Name: Ahmed Raza

Group Member 2:

Roll no: 23p – 0625

Name: Abdul Ahad

Functions

1. **`load_data`**
    - **Purpose:**
      This function attempts to load data from a specified JSON file. If the file is not found, it returns an empty dictionary. It is used to read the password database from disk.

2. **`save_data`**
    - **Purpose:**
      The `save_data` function is responsible for saving all changes made to the user database back into the JSON file. When any data is modified (like adding new accounts, updating passwords, etc.), this function ensures the changes are saved persistently.

3. **`send_email`**
    - **Purpose:**
      This function sends an email to a given email address. It is used for sending the OTP during the password recovery process. It uses the SMTP protocol and requires the sender's email, password, and recipient email.

```
    def send_email(self,subject, body, to_email):
        # Email configuration
        from_email = "ahadahadansari8@gmail.com"  #email address of the sender
        password = "dpta rfyz vdar fwhb"

        # Set up the MIME
        message = MIMEMultipart()
        message['From'] = from_email
        message['To'] = to_email
        message['Subject'] = subject

        # Add the email body to the message
        message.attach(MIMEText(body, 'plain'))

        try:
            # Set up the server
            server = smtplib.SMTP("smtp.gmail.com", 587)  # Gmail SMTP server and port
            server.starttls()                             # Enable security (TLS)
            server.login(from_email, password)            # Log in to the email account

            # Send the email
            server.sendmail(from_email, to_email, message.as_string())
            print("Email sent successfully!")
        except Exception as e:
            print(f"Error: {e}")
        finally:
            server.quit()  # Close the server connection

    def forgetMasterpswd(self):
        found=False
        username=input("enter the username:")
        for user,inkeys in self.accs.items():
            if username in inkeys['username']:
                email=input("enter the email linked to the account:")
                if email in inkeys['email']:
                    found=True
        if(found):
            while(1):
                otp_validity=50
                sent_time=time.time()
                rPass=random.randint(100,1000)
                self.send_email("Your OTP for Secure Password Manager Access",f"""
Hello,

We have generated an OTP for your account in the Password Manager. Please use this OTP to access your account and update it as soon as possible.

***** OTP CODE:  {rPass} *****This OTP will expire in 50 seconds******

For security reasons, please do not share this OTP with anyone, and make sure to change it to something more secure after logging in.

If you did not request this temporary password, please contact our support team immediately.

Best regards,
The Password Manager Team
"""
,email
)
                choice=1
                print("OTP sent check your email (OTP WILL EXPIRE AFTER 50 seconds):")
                Pass=int(input("enter OTP:"))
                elapsed_time=time.time()-sent_time
                if(elapsed_time>otp_validity):
                    choice=int(input("OTP EXPIRED, press 1 to send again 2 to go back to the login screen:"))
                elif Pass==rPass:
                    print("change your password asap!")
                    #here we will call the reset masterpassword function before giving the control to the login function
                    self.loggedin(username,Pass)
                else:
                    choice=int(input("incorrect OTP CODE,press 1 to send again 2 to go back to the login screen:"))
                if choice==2:
                    break
                if choice==1:
                    continue
                else:
                    print("invalid choice! redirecting back to foreget password page:")
                    self.forgetMasterpswd()
        else:
            print("wrong email or username entered:")
            print("redirecting to the the login page...")
        self.login()
```

4. **`forgetMasterpswd`**
    - **Purpose:**
      This function helps users who have forgotten their master password. It verifies the username and associated email, sends an OTP to the email, and allows the user to reset their master password. The OTP expires after a set time for added security.

5. **`derive_key`**
    - **Purpose:**
      This function derives a cryptographic key from the user's master password using the PBKDF2 (Password-Based Key Derivation Function 2) algorithm. It uses a salt to prevent attacks like rainbow tables, ensuring the key is unique to the password.

```python
def derive_key(self, password, salt):
    #Derives an AES key from the password and salt.
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=100_000,
        backend=default_backend()
    )
    return kdf.derive(password.encode())
```

6. **`encrypt_password`**
    - **Purpose:**
      This function encrypts a given password using AES (Advanced Encryption Standard) in GCM (Galois/Counter Mode). AES-GCM is a mode of encryption that provides both confidentiality and integrity, meaning the encrypted data can't be tampered with. The function returns the initialization vector (IV), the encrypted password, and the tag needed for decryption.

```python
def encrypt_password(self,password,key):
    iv = os.urandom(12)  # AES-GCM requires a 12-byte IV
    cipher = Cipher(algorithms.AES(key), modes.GCM(iv), backend=default_backend())
    encryptor = cipher.encryptor()
    encrypted_password = encryptor.update(password.encode()) + encryptor.finalize()
    return iv, encrypted_password, encryptor.tag
```

7. **`Signup`**
    - **Purpose:**
      This function handles the sign-up process. It prompts the user for a username, email, and master password. It checks if the username already exists in the system and if not, it furthers checks the password strength and suggests users strong passwords for their user profile moreover, it encrypts the master password and saves the user details securely in the database.

```python
def signup(self):
    print("Welcome to the Signup Process")
    username = input("Enter your username: ").strip()


    if username in self.accs:
        print("Username already exists. Please choose a different one.")
        return


    email = input("Enter your email address for recovery: ").strip()

    # Password setup
    while True:
        print("\nWould you like a suggested password? (y/n)")
        suggest = input().lower()
        if suggest == 'y':
            sPassword = self.passSuggest()
            print(f"Suggested Password: {sPassword}")

        password = input("Enter your password: ").strip()

        # Check password strength
        strength = self.passStrengthCheck(password)
        print(f"Password Strength: {strength}")
        if strength == "Weak":
            print("Your password is too weak. Please try again.")
        else:
            break

    confirm_password = input("Confirm your password: ").strip()
    if password != confirm_password:
        print("Passwords do not match. Signup failed.")
        return

    # Encrypt master password
    salt = os.urandom(16)
    key = self.derive_key(password, salt)
    iv, encrypted_password, tag = self.encrypt_password(password, key)

    # Save user data
    self.accs[username] = {
        "username": username,
        "email": email,
        "salt": urlsafe_b64encode(salt).decode(),
        "password": urlsafe_b64encode(encrypted_password).decode(),
        "iv": urlsafe_b64encode(iv).decode(),
        "tag": urlsafe_b64encode(tag).decode(),
        "accounts": []  # Empty list to hold accounts
    }
    self.save_data("database2.json", self.accs)
    print(f"Signup successful! Welcome, {username}. You can now log in.")
```

8. **`decrypt_password`**
    - **Purpose:**
      The function decrypts a password using the AES-GCM decryption process. The decryption process requires the IV, encrypted password, and the integrity tag. The decrypted password is returned to the user for use.

```python
def decrypt_password(self,key,iv,tag,encrypted_password):
    cipher = Cipher(algorithms.AES(key), modes.GCM(iv, tag), backend=default_backend())
    decryptor = cipher.decryptor()

    try:
        decrypted_password = decryptor.update(encrypted_password) + decryptor.finalize()
        return decrypted_password.decode()
    except Exception:
        return None
```

9. **`findMasterCredentials`**
    - **Purpose:**
      This function verifies the user's login credentials. It compares the entered password with the stored, encrypted password after decrypting it using the appropriate key. If the passwords match, the user is logged in.

10. **`addAcc`**
    - **Purpose:**
      This function allows users to add a new account to their password manager. It asks for the account ID and password, verifies the strength of the password, and encrypts it before storing it securely. The user can also request a password suggestion from the system.

11. **`removeAcc`**
    - **Purpose:**
      This function enables the user to remove an account by choosing an account from a list of saved accounts. Once selected, the account is deleted from the user's stored accounts, and the database is updated.

12. **`modifyPass`**
    - **Purpose:**
      This function allows the user to modify the password for an existing account. It prompts the user for a new password, checks its strength, and then encrypts and stores the new password. It ensures that the new password is stored securely and that the old password is replaced.

13. **`passSuggest`**
    - **Purpose:**
      This function generates a strong, random password containing a mix of uppercase letters, lowercase letters, numbers, and special characters. This function is used when the user wants to create a secure password for a new account

      .

14. **`passStrengthCheck`**
    - **Purpose:**
      This function checks the strength of a given password. It evaluates the password against several criteria such as length, the presence of uppercase and lowercase letters, numbers, and special characters. Based on these checks, it categorizes the password as "Strong," "Moderate," or "Weak."

15. **`retrievePass`**
    - **Purpose:**
      This function allows users to retrieve the password for one of their stored accounts. After verifying the user's identity with their master password, the function decrypts the selected account's password and copies it to the clipboard for easy access.

16. **`showAccs`**
    - **Purpose:**
      This function lists all the accounts that a user has stored in the password

manager. It displays each account with a masked version of the password to ensure privacy. The user can then choose an account to retrieve, modify, or remove.

17. **`loggedin`**
    - **Purpose:**
      This function is the main interface that the user interacts with after a successful login. It presents a menu with options to retrieve a password, modify a password, add a new account, or remove an account or to reset the masterPassword. It keeps the user engaged until they choose to log out.

```
def loggedin(self,username,password):
    while(1):
        print("welcome! "+username)
        print("Registered Accs:")
        passArr,idArr=self.showAccs(username)
        choice=int(input("press 1 to retrieve a password \n press 2 to modify a password \n press 3 to add an account\n press 4 to remove an account\n press 5 to reset masterpassword\n press -1 to exit"))
        if choice==1:
            self.retrievePass(passArr,username,password)
        elif choice==2:
            #password suggestor and strength checker will be called inside this
            self.modifyPass(idArr,username,password)
        elif choice==3:
            #password suggestor and strength checker will be called inside this
            self.addAcc(username,password)
        elif choice==4:
            self.removeAcc(idArr,username)
        elif choice == 5:
            self.resetMasterPassword(username, password)
        elif choice==-1:
            passArr.clear()
            idArr.clear()
            self.mainmenu()
        else:
            print("wrong choice! enter again:")
```

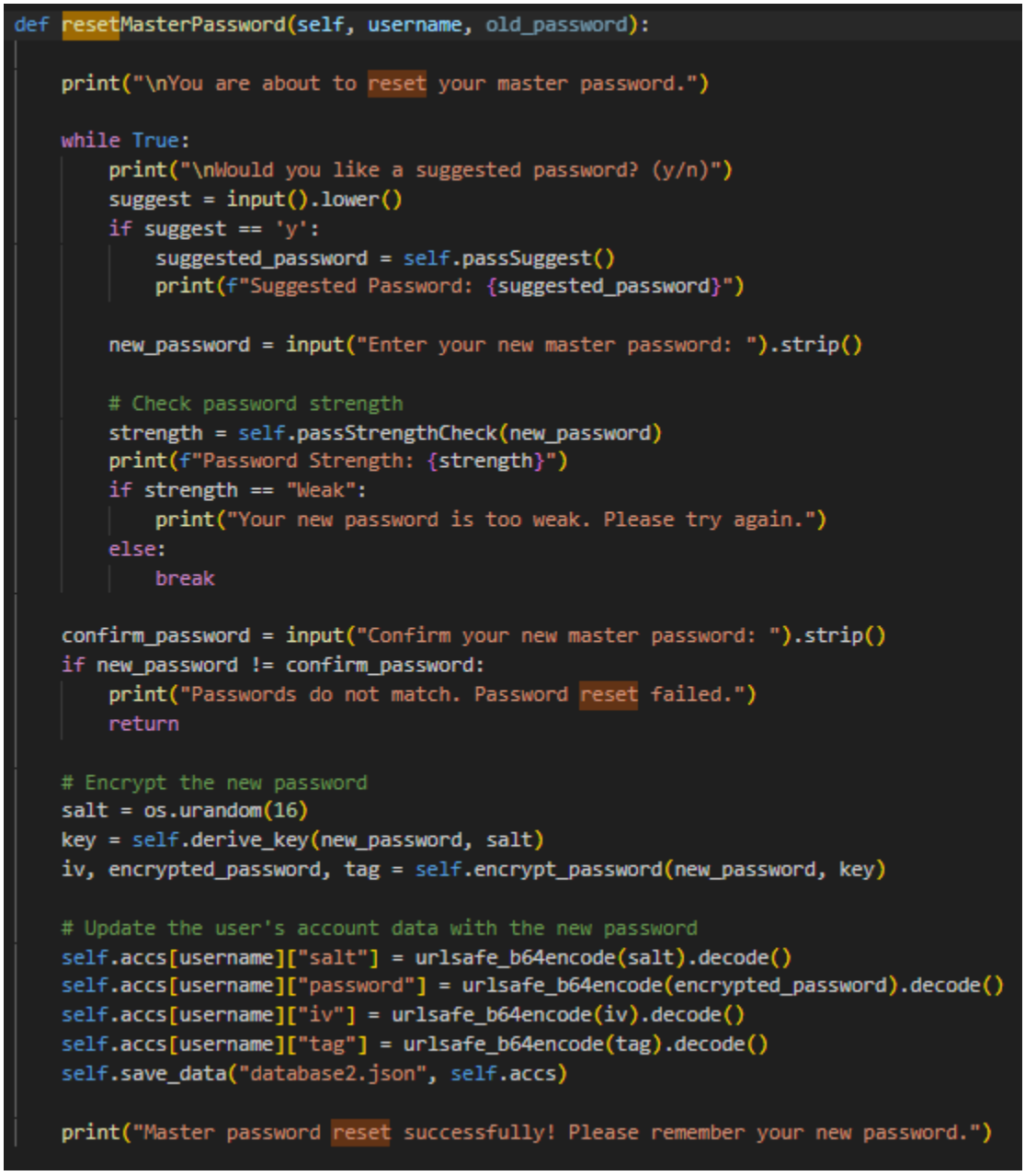
18. **`login`**
    - **Purpose:**
      This function manages the login process. It prompts the user for their username and password and verifies them against the stored credentials. If the user is successful, they are granted access to the password management features. If they forget their password, they can request a reset via OTP.

19. **`mainmenu`**
    - **Purpose:**
      The `mainmenu` function is the entry point of the program. It presents the user with options to log in, sign up, or exit the program. It directs the user

to the appropriate section of the application based on their input.

20. **`resetMasterPassword`**
    - **Purpose:**

      The resetMasterPassword function allows users to reset their master password in a secure and user-friendly manner. It ensures that the new password meets security standards and updates the user database accordingly. This function is critical for maintaining account security and preventing unauthorized access.



```
def resetMasterPassword(self, username, old_password):

    print("\nYou are about to reset your master password.")

    while True:
        print("\nWould you like a suggested password? (y/n)")
        suggest = input().lower()
        if suggest == 'y':
            suggested_password = self.passSuggest()
            print(f"Suggested Password: {suggested_password}")

        new_password = input("Enter your new master password: ").strip()

        # Check password strength
        strength = self.passStrengthCheck(new_password)
        print(f"Password Strength: {strength}")
        if strength == "Weak":
            print("Your new password is too weak. Please try again.")
        else:
            break

    confirm_password = input("Confirm your new master password: ").strip()
    if new_password != confirm_password:
        print("Passwords do not match. Password reset failed.")
        return

    # Encrypt the new password
    salt = os.urandom(16)
    key = self.derive_key(new_password, salt)
    iv, encrypted_password, tag = self.encrypt_password(new_password, key)

    # Update the user's account data with the new password
    self.accs[username]["salt"] = urlsafe_b64encode(salt).decode()
    self.accs[username]["password"] = urlsafe_b64encode(encrypted_password).decode()
    self.accs[username]["iv"] = urlsafe_b64encode(iv).decode()
    self.accs[username]["tag"] = urlsafe_b64encode(tag).decode()
    self.save_data("database2.json", self.accs)

    print("Master password reset successfully! Please remember your new password.")
```

Security Considerations

The project uses strong encryption methods to ensure the security of user data:

1. **AES-GCM Encryption:**
   This mode of AES encryption provides both confidentiality and integrity. AES is a symmetric encryption algorithm, meaning the same key is used for both encryption and decryption. GCM ensures that any tampering with the data is detected.

2. **PBKDF2 for Key Derivation:**
   PBKDF2 is used to derive a cryptographic key from the user's master password. By using a salt and performing multiple iterations, it prevents attackers from using precomputed tables (rainbow tables) to crack the password.

3. **Email for OTP Verification:**
   The system uses OTPs sent via email to ensure that users can securely recover their master password if they forget it. The OTP is time-sensitive to prevent unauthorized access.

4. **Password Strength Verification:**
   The system checks the strength of passwords before allowing users to save them. This ensures that weak passwords (such as common or short passwords) are not stored.

User Experience

The system is designed to be intuitive and easy to use:

- The **login** process is straightforward, with clear prompts for the username and password.
- Users are guided step-by-step through the **signup** process, where they are required to enter a username, email, and password.
- Once logged in, users can view a list of their stored accounts and perform various actions (such as retrieving, modifying, or deleting passwords).
- The password manager also provides options for password recovery using OTPs, which makes it convenient for users who forget their master password.
- There are various menus and choices that allow the user to interact with the system based on their needs, whether it's adding new accounts, viewing stored passwords, or managing existing ones.

Conclusion

This Password Manager(python based) project provides a secure, user-friendly solution for managing passwords. With strong encryption, OTP-based recovery, password strength checking, and password suggestion features, it offers a robust set of tools for users to keep their passwords safe. The user interface is intuitive, and the system ensures that sensitive data is protected at all times.

The project has demonstrated the importance of security practices in handling sensitive information and provides an efficient way for users to store and manage their passwords in a safe and reliable manner.