**HOW TO BUILD A PASSWORD MANAGER WITH PYTHON**

Hellooooo! Today, we’re going to be building our own custom password manager. I’m pretty sure you know what a password manager is, and its importance. But for formalities, a password manager is an application that securely stores, organizes, and manages a user's passwords for various online accounts and services. It is used to simplify the process of creating and remembering complex and unique passwords for enhanced online security.

Basically it saves us the hassle of remembering passwords for different accounts as well as remembering complex passwords. This is because security wise, we’re expected to use different passwords for different accounts so that if an account is compromised, using the same credentials wouldn’t grant hackers access to another account owned by the same victim. We are also expected to use strong, unguessable passwords. We talked about this in a previous tutorial.

The basic functionalities of our password manager include: logging in (only we should have access to our password manager using our username and master password), adding passwords, retrieving passwords (obviously), viewing saved websites (to be sure which websites we have saved) and copying the retrieved password to our clipboard as it saves us the time of having to type it. I don’t know about you, but I can be very lazy :) .

We are going to be storing the saved passwords in a JSON file. But it will be encrypted so that even if someone manages to get access to the file, it would be useless to them. We’ll also be storing our login credentials (username and master password) in a JSON file. And similarly, it would be hashed so it’s useless to unauthorized parties. So do me a favor. Create a folder, create 2 JSON files in the folder. Name one **passwords.json** and the other, **user\_data.json.** The passwords.json file is where we’ll store our saved passwords, and the user\_data.json is where we will store login credentials. Please bear in mind that you can save them however you like but they must be JSON files and you must replace the names you save them with before running the code. In the same folder create a python file. Name it **password\_manager.py** or whatever you like.

Bear in mind that we’re using python3. Any version from 3.6 is fine as we’re going to make use of **F strings**. Also, to copy our retrieved password to our clipboard, we’ll make use of the **pyperclip** module. To install, open up your command prompt (or terminal) and run:

| pip **install** pyperclip |
| --- |

That’s the only module we have to install, as the rest come built-in with python.

Now, the fun can begin. Open up your python file and let’s import the necessary libraries.

| **import** json, hashlib, getpass, os, pyperclip **from** cryptography.fernet **import** Fernet |
| --- |

**json**: is a library for encoding and decoding JSON (JavaScript Object Notation) data, commonly used for data serialization and interchange.

**Hashlib**: is a library that provides secure hash functions, including SHA-256, for creating hash values from data, often used for password hashing and data integrity verification.

**getpass**: A library for safely and interactively entering sensitive information like passwords without displaying the input on the screen. Similar to the way the linux terminals are.

**os**: A library for interacting with the operating system, allowing you to perform tasks like file and directory manipulation.

**pyperclip**: A library for clipboard operations, enabling you to programmatically copy and paste text to and from the clipboard in a platform-independent way.

**cryptography.ferne**t: Part of the cryptography library, it provides the Fernet symmetric-key encryption method for securely encrypting and decrypting data.

After importing the necessary libraries, we create a function for Hashing. This function would be used to hash our master password upon registration.

| # Function for Hashing the Master Password. **def** **hash\_password**(password):  sha256 = hashlib.sha256()  sha256.update(password.encode())  **return** sha256.hexdigest() |
| --- |

Afterwards, we create a function for generating a key. This key would be used to encrypt our passwords upon adding, and decrypt upon retrieval. Please bear in mind that only the key we used to encrypt our passwords, can be used to decrypt it. If you use another, you’ll get errors. I’m saying this because this is a function and anytime it gets executed (you call it), it generates a new key. This is worth noting in case you want to use the function in another program. In this program, we generate it once, store it and keep using it, so you have nothing to worry about.

Next up, we Fernet the key (making it able to encrypt and decrypt our passwords), and create functions for encrypting and decrypting.

| # Initialize Fernet cipher with the provided key. **def** **initialize\_cipher**(key):  **return** Fernet(key)   # Function to encrypt a password. **def** **encrypt\_password**(cipher, password):  **return** cipher.encrypt(password.encode()).decode()   # Function to decrypt a password. **def** **decrypt\_password**(cipher, encrypted\_password):  **return** cipher.decrypt(encrypted\_password.encode()).decode() |
| --- |

After that, we create a function for owner registration. Saving credentials in the **user\_data.json** file. Remember, we’ll be hashing the password. This is the Master Password. The keys to the kingdom.

| # Function to register a user. **def** **register**(username, master\_password, key):  # Hash the master password before storing it  hashed\_master\_password = hash\_password(master\_password)  user\_data = {'username': username, 'master\_password': hashed\_master\_password}  **with** open('user\_data.json', 'w') **as** file:  json.dump(user\_data, file) |
| --- |

Next up, we create a function for logging a user in. How it works is, it accepts username and password from a user, it then hashes the password entered by the user. If the hash of the entered password is the same as the hash of the saved password ( in the JSON file) and the usernames are also the same, it grants access. Otherwise, you know the rest. This is how password cracking processes take place. In systems, passwords are stored as hashes and not plain texts (in secure systems). So attackers keep trying the hashes of different known passwords in hope of gaining access to the system. This is why we are advised to use strong, unique passwords.

| **def login(username, entered\_password):  with open('user\_data.json', 'r') as file:  user\_data = json.load(file)   stored\_password\_hash = user\_data.get('master\_password')  entered\_password\_hash = hash\_password(entered\_password)   if entered\_password\_hash == stored\_password\_hash and username == user\_data.get('username'):  return True  else:  return False** |
| --- |

Next up, a function to view websites saved in our Password Manager. Remember that the order with which functions are written doesn’t really matter. It’s the calling that matters.

| # Function to view saved websites. **def** **view\_websites**():  **with** open('passwords.json', 'r') **as** data:  view = json.load(data)  print("Websites you saved...\n")  **for** x **in** view:  print(x['website'])  print('\n') |
| --- |

Next up, we generate or load our key. If it’s the first time, we generate and save our key. If it’s not, we load our key.

| # Load or generate the encryption key key\_filename = 'encryption\_key.key' **if** os.path.exists(key\_filename):  **with** open(key\_filename, 'rb') **as** key\_file:  key = key\_file.read() **else**:  key = generate\_key()  **with** open(key\_filename, 'wb') **as** key\_file:  key\_file.write(key)  cipher = initialize\_cipher(key) |
| --- |

Basically this function checks if an **encryption\_key.key** file exists. If it does, it loads it for use. If it doesn’t it creates it and saves our unique key in it. If it exists, it means it isn’t our first time running the program so it just loads our unique key. This key is the heart of this program. You want to make sure you keep it safe.

Next up, a function to add passwords.

| **def** **add\_password**(website, password):  # Check if passwords.json exists  **if** **not** os.path.exists('passwords.json'):  # If passwords.json doesn't exist, initialize it with an empty list  data = []  **else**:  # Load existing data from passwords.json  **try**:  **with** open('passwords.json', 'r') **as** file:  data = json.load(file)  **except** json.JSONDecodeError:  # Handle the case where passwords.json is empty or invalid JSON.  data = []   # Encrypt the password  encrypted\_password = encrypt\_password(cipher, password)   # Create a dictionary to store the website and password  password\_entry = {'website': website, 'password': encrypted\_password}  data.append(password\_entry)   # Save the updated list back to passwords.json  **with** open('passwords.json', 'w') **as** file:  json.dump(data, file, indent=4) |
| --- |

Here, we encrypt the passwords and save them in our JSON file.

Next up, a function to retrieve a saved password.

| **def** **get\_password**(website):  # Check if passwords.json exists  **if** **not** os.path.exists('passwords.json'):  **return** **None**   # Load existing data from passwords.json  **try**:  **with** open('passwords.json', 'r') **as** file:  data = json.load(file)  **except** json.JSONDecodeError:  data = []   **for** entry **in** data:  **if** entry['website'] == website:  # Decrypt and return the password  decrypted\_password = decrypt\_password(cipher, entry['password'])  **return** decrypted\_password   **return** **None** |
| --- |

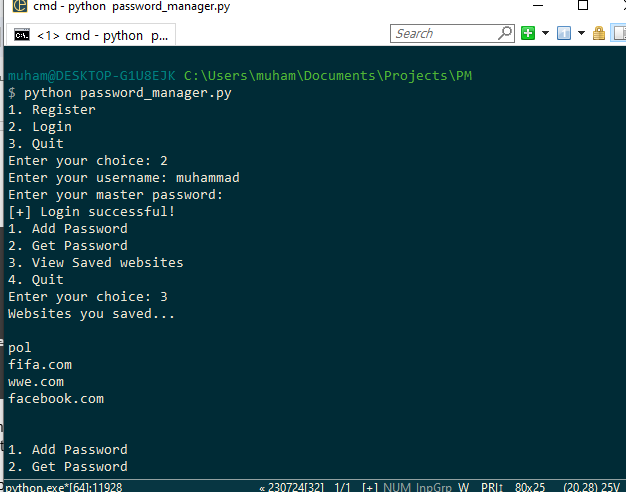
This function takes in the website as a parameter, and returns the decrypted password to the user (us).

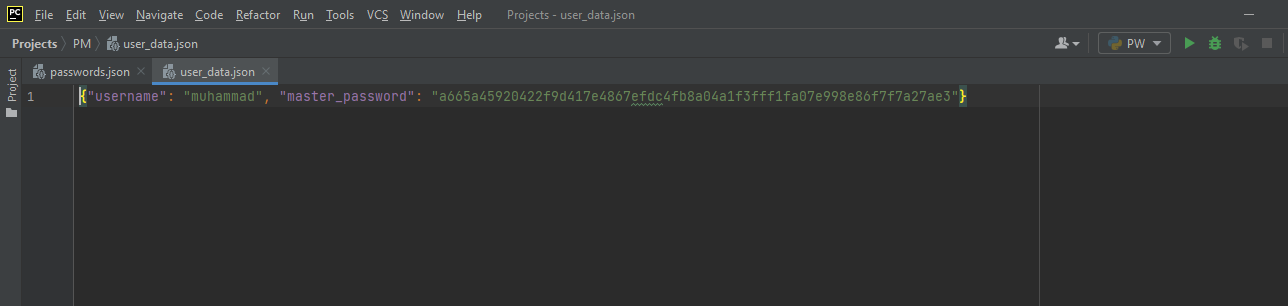
Finally the body of the program. Option displays and function calling according to the user’s input.

| # Infinite loop to keep the program running until the user chooses to quit. **while** **True**:  print("1. Register")  print("2. Login")  print("3. Quit")  choice = input("Enter your choice: ")   **if** choice == '1': # If a user wants to register  file = 'user\_data.json'  # Check if user\_data.json exists and is empty  **if** os.path.exists(file) **and** os.path.getsize(file) == 0:  username = input("Enter your username: ")  master\_password = getpass.getpass("Enter your master password: ")   # Call the register function to store user data securely  register(username, master\_password, key)  print("[+] Registration successful!")  **else**:  print("[-] Master user already exists.")   **elif** choice == '2': # If a User wants to log in  username = input("Enter your username: ")  master\_password = getpass.getpass("Enter your master password: ")  user\_data = login(username, master\_password)   **if** user\_data: # Successful login  print("[+] Login successful!")   **while** **True**:  print("1. Add Password")  print("2. Get Password")  print("3. View Saved websites")  print("4. Quit")  password\_choice = input("Enter your choice: ")   **if** password\_choice == '1': # If a user wants to add a password  website = input("Enter website: ")  password = getpass.getpass("Enter password: ")   # Encrypt and add the password  add\_password(website, password)  print("[+] Password added!")   **elif** password\_choice == '2': # If a User wants to retrieve a password  website = input("Enter website: ")  decrypted\_password = get\_password(website)  **if** website **and** decrypted\_password:  # Copy password to clipboard for convenience  pyperclip.copy(decrypted\_password)  print(f"[+] Password for {website}: {decrypted\_password}\n[+] Password copied to clipboard.")  **else**:  print("[-] Password not found! Did you save the password?")   **elif** password\_choice == '3': # If a user wants to view saved websites  view\_websites()   **elif** password\_choice == '4': # If a user wants to quit the password manager  **break**   **else**: # Login failed  print("[-] Login failed. Incorrect master password!")   **elif** choice == '3': # If a user wants to quit the program  **break** |
| --- |

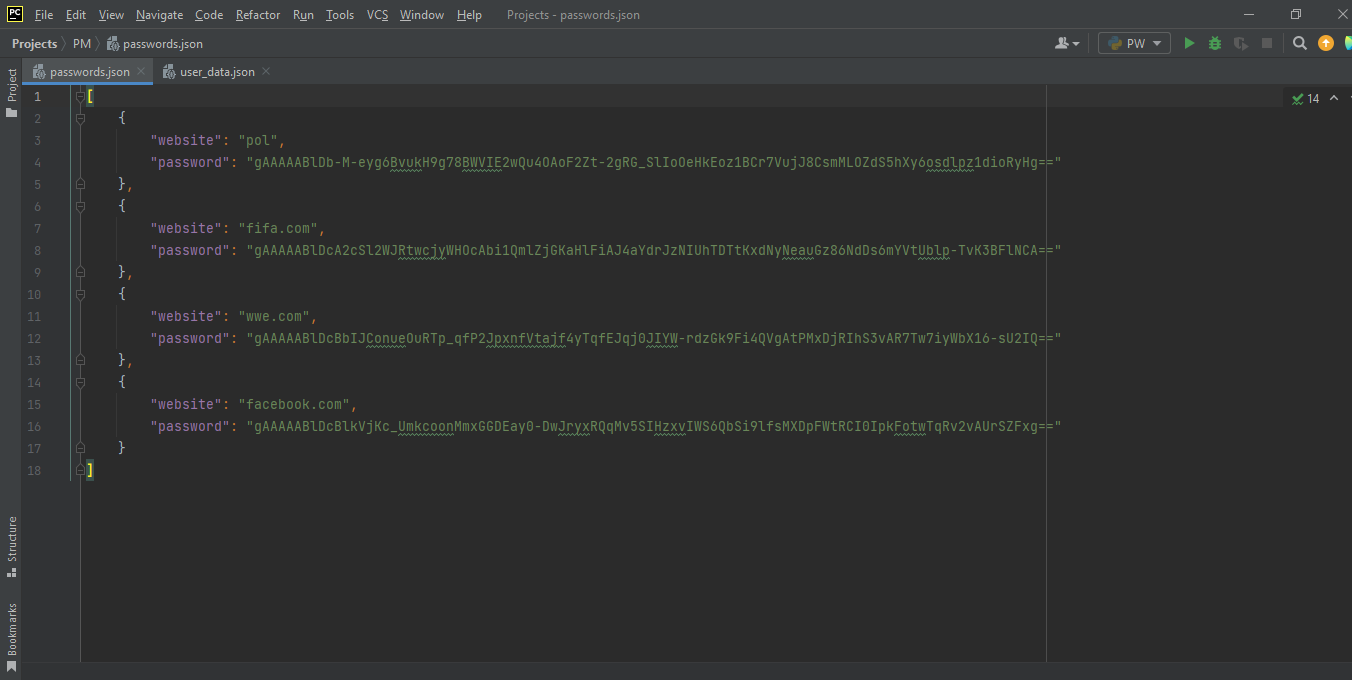
When you run the program (after fulfilling all the requirements earlier stated), the first thing you want to do is **register.** Afterwards you can login and play with all the features. Don’t forget that when you retrieve a password, it copies to the clipboard. Feel free to test that by pasting it to see for yourself.

During execution you should get output similar to:





After registering, your **user\_data.json** should look like this.



After playing around, your **passwords.json** file should look like this.

Please bear in mind that full scale password managers are often more complex than this. The essence of this tutorial is to try to show you the methodology involved in this process. I hope I was able to do that.

Till next time.

Best regards,

Abdullahi Muhammad.

Twitter - [https://www.x.com/muhammad\_xxv](https://x.com/muhammad_xxv)

Linkedin - <https://www.linkedin.com/in/muhammad-abdullah20/>

Instagram - <https://instagram.com/muhammad.xxv>