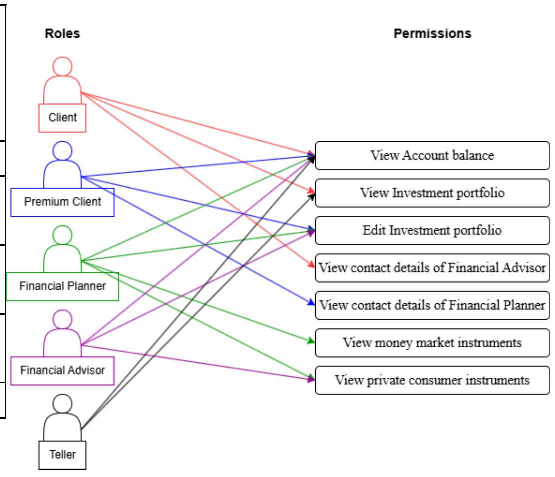
Problem 1:

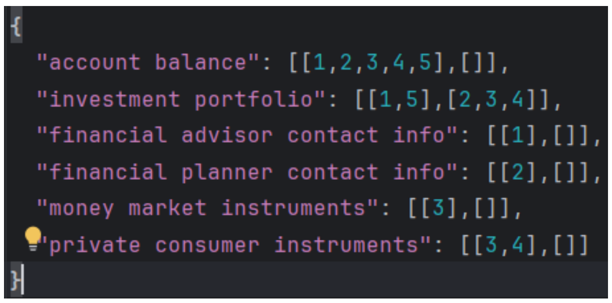
a. The access control model that was used in the development of this prototype is RBAC. We can define roles for each type of user in the system like client, premium client, etc. Each role is tied to permitted objects with specific permission like view and edit. So depending on the user role, they would have specific objects they can access.

b. In the table below, the **r** signifies that the role has viewing permissions, and **w** signifies that the role has viewing and editing permissions.

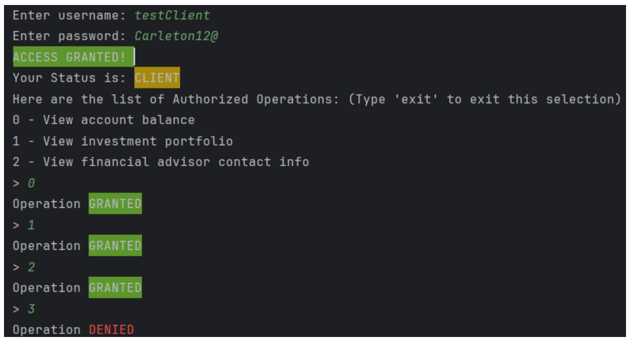
Role /Permissions	Account balance	Investment portfolio	contact details of Financial Advisor	contact details of Financial Planner	money market instruments	private consumer instruments
Client	r	r	r			
Premium Client	r	rw		r		
Financial Planner	r	rw			r	r
Financial Advisor	r	rw				r
Teller	r	r				


Roles
Permissions
Client
Premium Client
Financial Planner
Financial Advisor
Teller
View Account balance
View Investment portfolio
Edit Investment portfolio
View contact details of Financial Advisor
View contact details of Financial Planner
View money market instruments
View private consumer instruments

c. For the implementation, I implemented the RBAC access control as a JSON list where for each subject, I defined the allowed objects that can access the view or edit of that operation. Each number corresponds to a Role from the ACL table.



```
{
  "account balance": [[1,2,3,4,5],[]],
  "investment portfolio": [[1,5],[2,3,4]],
  "financial advisor contact info": [[1],[]],
  "financial planner contact info": [[2],[]],
  "money market instruments": [[3],[]],
  "private consumer instruments": [[3,4],[]]
}
```



```
Enter username: testClient
Enter password: Carleton12@
ACCESS GRANTED!
Your Status is: CLIENT
Here are the list of Authorized Operations: (Type 'exit' to exit this selection)
0 - View account balance
1 - View investment portfolio
2 - View financial advisor contact info
> 0
Operation GRANTED
> 1
Operation GRANTED
> 2
Operation GRANTED
> 3
Operation DENIED
```

For test cases, I created different accounts with different roles and made sure that only the allowed operations could be selected. So, I created one account for each of the roles (Client, Premium Client, Financial Planner, Financial Advisor, Teller), and then I checked the granted operations and the denied operations.

For example: I signup and log in as testClient account that has a Client role, then I check the operations that I am allowed to pick and then try to select it through the interface. The list of options that is outputted depends on the user role, so in this case the Client role user displayed the View Account balance, View Investment Portfolio and View financial portfolio.

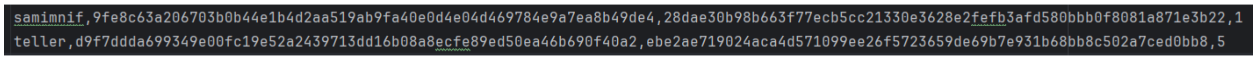
Another test performed to check the Access Control is the teller role, where they are allowed to login to system only between 9:00 AM and 5:00 PM, so the system checks the current time and then decides to allow the user to login or not. For testing purposes, we can change the system time by selecting the change time option.

Problem 2:

a. To hash the passwords stored, I used a hash algorithm provided by the **hashlib** python library which is **SHA-256**. This algorithm is widely used and is fast which makes it suitable for the prototype I am developing. The hash function I created uses salt to prevent identical passwords from generating the same hash and to slow down brute-force attacks. The function generates a random **32-byte hexadecimal salt** that adds to the plain password and then it gets hashed as a whole. This will ensure the H2 hash property where it will be hard to find a second hash that is the same in the file.

b. The password file will be structured to include *username, salt, hashed_password, role*
- *username:* The first part is the username so we can keep track of the user login credentials and also associate the password with it.
- *salt:* The second part is salt which was generated randomly as a 32-byte long hexadecimal.
- *hashed_password:* The third part is the hashed password that is combined with the salt mentioned previously.
- *role***:** The fourth part is the user role, which specifies the user role in the system to be able to access specific functions.

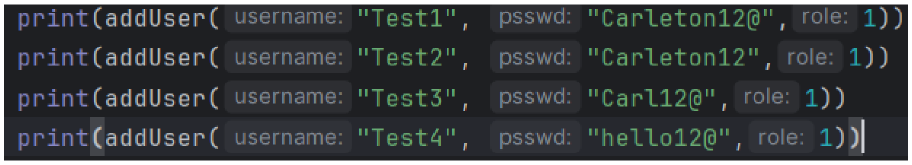
Example of the password stored file is shown below:



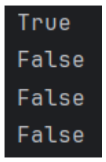
```
samimnif,9fe8c63a206703b0b44e1b4d2aa519ab9fa40e0d4e04d469784e9a7ea8b49de4,28dae30b98b663f77ecb5cc21330e3628e2fefb3afd580bbb0f8081a871e3b22,1
teller,d9f7ddda699349e00fc19e52a2439713dd16b08a8ecfe89ed50ea46b690f40a2,ebe2ae719024aca4d571099ee26f5723659de69b7e931b68bb8c502a7ced0bb8,5
```

c. Please see the user.py file for the implementation

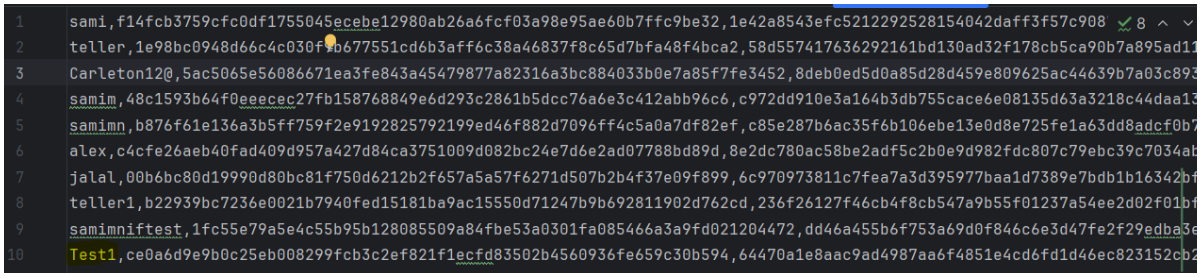
d. For Testing, I created 4 different accounts, Test1 has a valid password, and the other 3 test accounts have invalid accounts. The result was correct on the console and Test1 account was added to the passwd.txt file as seen below. The system checks if the user already exists and also checks if the password is valid. It compares it to a deny list that is provided in the project folder (weak_passwords.txt). This deny list contains around 1 million commonly used passwords. Another check that was performed on the inputted passwords is the structure requirement. The password must have one upper-case letter, one lower-case letter, one special character from the following (!, @, #, $, %, *, &), and must have a length between 8 and 12 characters.



```
print(addUser( username: "Test1",  psswd: "Carleton12@", role: 1))
print(addUser( username: "Test2",  psswd: "Carleton12", role: 1))
print(addUser( username: "Test3",  psswd: "Carl12@", role: 1))
print(addUser( username: "Test4",  psswd: "hello12@", role: 1))
```



```
True
False
False
False
```



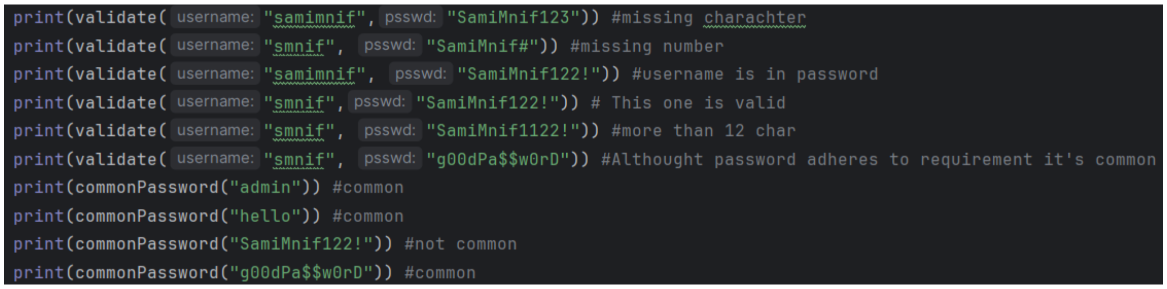
```
sami,f14fcb3759cfc0df1755045ecebe12980ab26a6fcf03a98e95ae60b7ffc9be32,1e42a8543efc5212292528154042daff3f57c908
teller,1e98bc0948d66c4c030f[illegible]b677551cd6b3aff6c38a46837f8c65d7bfa48f4bca2,58d557417636292161bd130ad32f178cb5ca90b7a895ad11
Carleton12@,5ac5065e56086671ea3fe843a45479877a82316a3bc884033b0e7a85f7fe3452,8deb0ed5d0a85d28d459e809625ac44639b7a03c893
samim,48c1593b64f0eeecec27fb158768849e6d293c2861b5dcc76a6e3c412abb96c6,c972dd910e3a164b3db755cace6e08135d63a3218c44daa13
samimn,b876f61e136a3b5ff759f2e9192825792199ed46f882d7096ff4c5a0a7df82ef,c85e287b6ac35f6b106ebe13e0d8e725fe1a63dd8adcf0b7
alex,c4cfe26aeb40fad409d957a427d84ca3751009d082bc24e7d6e2ad07788bd89d,8e2dc780ac58be2adf5c2b0e9d982fdc807c79ebc39c7034ab
jalal,00b6bc80d19990d80bc81f750d6212b2f657a5a57f6271d507b2b4f37e09f899,6c970973811c7fea7a3d395977baa1d7389e7bdb1b16342bf
teller1,b22939bc7236e0021b7940fed15181ba9ac15550d71247b9b692811902d762cd,236f26127f46cb4f8cb547a9b55f01237a54ee2d02f01bf
samimniftest,1fc55e79a5e4c55b95b128085509a84fbe53a0301fa085466a3a9fd021204472,dd46a455b6f753a69d0f846c6e3d47fe2f29edba3e
Test1,ce0a6d9e9b0c25eb008299fcb3c2ef821f1ecfd83502b4560936fe659c30b594,64470a1e8aac9ad4987aa6f4851e4cd6fd1d46ec823152cb2
```

Problem 3:

a. Please see the interface.py file for the signup interface implementation

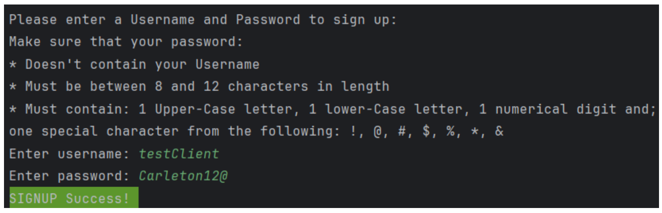
b. Please see passwd_validation.py file for the proactive password checker

c. To test the signup implementation, I first tested the functions that it used to validate the password and make sure that the password adheres to the requirements mentioned in the assignment description.



```
print(validate( username: "samimnif", psswd: "SamiMnif123")) #missing charachter
print(validate( username: "smnif", psswd: "SamiMnif#")) #missing number
print(validate( username: "samimnif", psswd: "SamiMnif122!")) #username is in password
print(validate( username: "smnif", psswd: "SamiMnif122!")) # This one is valid
print(validate( username: "smnif", psswd: "SamiMnif1122!")) #more than 12 char
print(validate( username: "smnif", psswd: "g00dPa$$w0rD")) #Althought password adheres to requirement it's common
print(commonPassword("admin")) #common
print(commonPassword("hello")) #common
print(commonPassword("SamiMnif122!")) #not common
print(commonPassword("g00dPa$$w0rD")) #common
```

Then I used the user interface to directly input the values manually and checked if the signup was successful or not. For example, I tried to make a Client user and entered a password that adheres to the requirement as shown below.

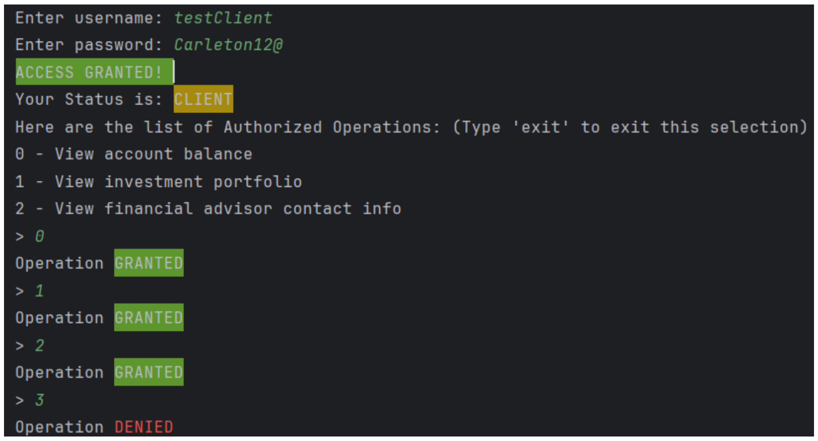


```
Please enter a Username and Password to sign up:
Make sure that your password:
* Doesn't contain your Username
* Must be between 8 and 12 characters in length
* Must contain: 1 Upper-Case letter, 1 lower-Case letter, 1 numerical digit and;
one special character from the following: !, @, #, $, %, *, &
Enter username: testClient
Enter password: Carleton12@
SIGNUP Success!
```

Problem 4:

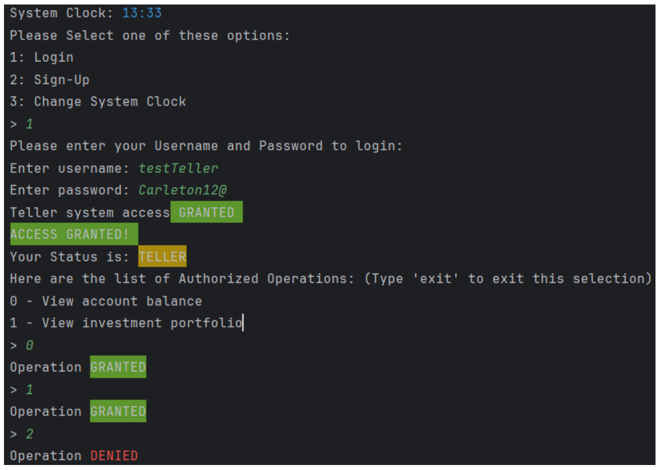
a. Please see the interface.py file for the login interface implementation
b. Please see the interface.py file for the login interface implementation, under Main
c. To test the user login and the access control, I created multiple accounts with different roles and tried to sign in and check if the user has the correct corresponding permitted operations. For example:

- I created a Client user *testClient* with password: *Carleton12@* and then logged in
  The Client role has only 3 permitted operations as shown below that follows the RBAC defined.

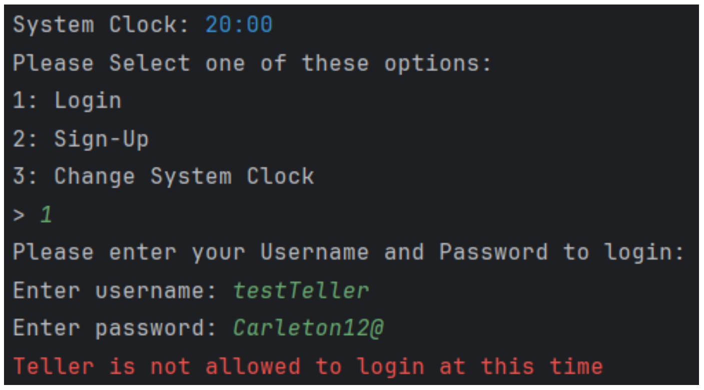


```
Enter username: testClient
Enter password: Carleton12@
ACCESS GRANTED!
Your Status is: CLIENT
Here are the list of Authorized Operations: (Type 'exit' to exit this selection)
0 - View account balance
1 - View investment portfolio
2 - View financial advisor contact info
> 0
Operation GRANTED
> 1
Operation GRANTED
> 2
Operation GRANTED
> 3
Operation DENIED
```

- I also created a Teller user *testTeller* with password: *Carleton12@* and made sure that they can only log in within their working hours specified in the assignment instructions



```
System Clock: 13:33
Please Select one of these options:
1: Login
2: Sign-Up
3: Change System Clock
> 1
Please enter your Username and Password to login:
Enter username: testTeller
Enter password: Carleton12@
Teller system access GRANTED
ACCESS GRANTED!
Your Status is: TELLER
Here are the list of Authorized Operations: (Type 'exit' to exit this selection)
0 - View account balance
1 - View investment portfolio
> 0
Operation GRANTED
> 1
Operation GRANTED
> 2
Operation DENIED
```



```
System Clock: 20:00
Please Select one of these options:
1: Login
2: Sign-Up
3: Change System Clock
> 1
Please enter your Username and Password to login:
Enter username: testTeller
Enter password: Carleton12@
Teller is not allowed to login at this time
```