

1. A) The access control model I selected was Role-Based Access Control (RBAC). Most requirements of the system are just dependent on the role of the user and what object they can access without environment limitations. Hence, RBAC being a good choice overall.

b) I will be using an access control matrix to represent my access control model of RBAC. This way I can visually map out which roles have what sort of influence over a specific object.

c)

CD: Contact details

| | Client Account Balance | Client Investment Portfolio | CD of FA | CD of FP | CD of IA | Money Market Instruments | Private Consumer Instruments | Derivatives Trading | Interest Instruments |
|--------------------|------------------------|-----------------------------|----------|----------|----------|--------------------------|------------------------------|---------------------|----------------------|
| Regular Client | v | v | v | | | | | | |
| Premium Client | v | vm | v | v | v | | | | |
| Teller | v | v | | | | | | | |
| Financial Advisor | v | vm | | | | | v | | |
| Compliance Officer | v | vx | | | | | | | |
| Investment Analyst | v | vm | | | | v | v | v | v |
| Financial Planner | v | vm | | | | v | v | | |
| Technical Support | a | a | | | | | | | |

R = {view (v), modify (m), validate modifications (x), view upon access (a)}

d) The access control mechanism mainly works off of the following dictionary, roles, implemented. The dictionary consists of keys which are the different types of users of the system. Then within that key there is another dictionary with the possible actions that a user can take. The value to this action key would be the list of different objects that can be accessed in that action.

```
roles = {
  "Regular Client":
  {
    "view": ["Client Account Balance", "Client Investment Portfolio",
"Contact Details of Financial Advisor"],
    "modify": [],
    "validate modifications": [],
    "view upon access": []
  },
  "Premium Client":
  {
```

```
        "view": ["Client Account Balance", "Client Investment Portfolio",
"Contact Details of Financial Advisor", "Contact Details of Financial Planner",
"Contact Details of Investment Analyst"],
        "modify": ["Client Investment Portfolio"],
        "validate modifications": [],
        "view upon access": []
    },
    "Teller":
    {
        "view": ["Client Account Balance (Between the hours of 9-5)", "Client
Investment Portfolio (Between the hours of 9-5)"],
        "modify": [],
        "validate modifications": [],
        "view upon access": []
    },
    "Financial Advisor":
    {
        "view": ["Client Account Balance", "Client Investment Portfolio",
"Private Consumer Instruments"],
        "modify": ["Client Investment Portfolio"],
        "validate modifications": [],
        "view upon access": []
    },
    "Compliance Officer":
    {
        "view": ["Client Account Balance", "Client Investment Portfolio"],
        "modify": [],
        "validate modifications": ["Client Investment Portfolio"],
        "view upon access": []
    },
    "Investment Analyst":
    {
        "view": ["Client Account Balance", "Client Investment Portfolio",
"Money Market Instruments", "Private Consumer Instruments", "Derivatives
Trading", "Interest Instruments"],
        "modify": ["Client Investment Portfolio"],
        "validate modifications": [],
        "view upon access": []
    },
    "Financial Planner":
    {
        "view": ["Client Account Balance", "Client Investment Portfolio",
"Money Market Instruments", "Private Consumer Instruments"],
        "modify": ["Client Investment Portfolio"],
        "validate modifications": [],
```

```

        "view upon access": []
    },
    "Technical Support":
    {
        "view": [],
        "modify": [],
        "validate modifications": [],
        "view upon access": ["Client Account Balance", "Client Investment
Portfolio"]
    },
}

```

The following screenshot shows where the system actually outputs what the user has access too. Where arr[4] is the role taken from the password file (after correct validation), and it looks for that role in the roles dictionary above.

```

print("You are a " + arr[4] + " and can view: " + ", ".join(roles[arr[4]]["view"]))
print("\nand can modify: " + ", ".join(roles[arr[4]]["modify"]))
print("\nand can validate modifications: " + ", ".join(roles[arr[4]]["validate modifications"]))
print("\nand can view upon access: " + ", ".join(roles[arr[4]]["view upon access"]))

```

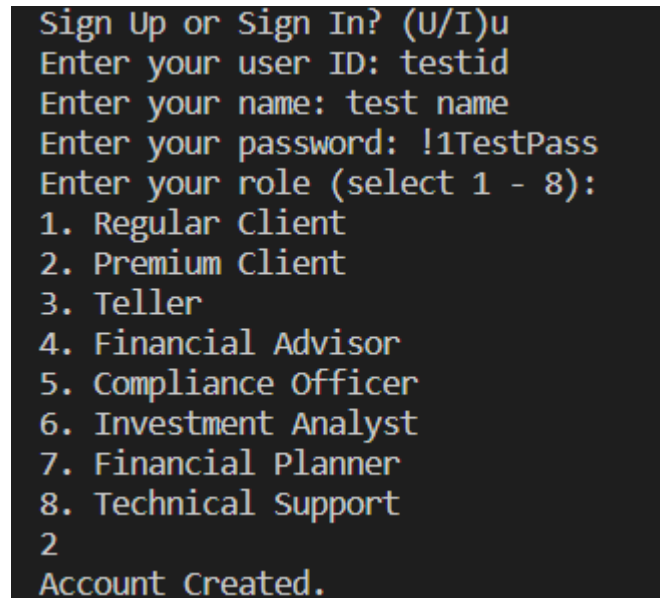
2. A) The hash function I selected was SHA256 with a hash length of 64 bytes as this will serve our purpose of simulating this system. The salt will have a length of 32 bytes and will be generated with `os.urandom(blake2b.SALT_SIZE)` from the python hashlib library. BLAKE2 supports keyed mode, a faster and simpler replacement for HMAC, and salted hashing. `Os.urandom()` allows the salt to be 16 or more bytes from a proper source which is better for passwords.
- b) Each record in the password file will have a user id, which the user will use to log in with. There will also be a user name just for the name of the user. There will then be the hashed password and the plaintext salt. These will be used for login verification. And last will be the role of the user so this can be used to check what access they have to when they log in. This is all separated in a colon.
- c) In the following screenshot we can see that we open or create our password.txt file and append our new entry into it. We can see the salt creation on the second line and the hashed password generated on the third line. These are both stores as hexadecimal values in the text file. The last line shows the format in which the information is printed starting with user id: name of user: hashed password: plaintext salt: role of user.

```

with open('passwd.txt', 'a') as f:
    salt = os.urandom(blake2b.SALT_SIZE)
    dk = pbkdf2_hmac('sha256', password.encode('utf-8'), salt, 500_000)
    f.write(userid+':'+username+':'+dk.hex()+':'+salt.hex()+':'+role+'\n')

```

3. A) The user interface will consist of just text inputs and outputs in the terminal. TO enroll a user the system will require a user id used for login and a name to display upon a valid login. Additionally, a password that meets the password requirements. And the last the role of the new user. This will be enough information to validate a user and check their authentication access control.



```
Sign Up or Sign In? (U/I)u
Enter your user ID: testid
Enter your name: test name
Enter your password: !1TestPass
Enter your role (select 1 - 8):
1. Regular Client
2. Premium Client
3. Teller
4. Financial Advisor
5. Compliance Officer
6. Investment Analyst
7. Financial Planner
8. Technical Support
2
Account Created.
```

b) The design of this password checker will be a bunch of ifs and elif statements to check if the requirements are met. There will be an if for:

- Password is less than 8 characters
- Password is greater than 12 characters
- Password has no uppercase letter
- Password has no lower-case letter
- Password has no number
- Password has no special character
- Password is equal to any password in a list of common passwords (provided in an external text file called common_passwords.txt)
- Password is the same as the user id or user name

If any of these tests pass, the system will ask the user to enter a new password.

A note, that we do not need to check for passwords matching the format of calendar dates, telephone number, or license plates because those don't have special characters. The password check fails the inputted password without a special character always so it will be impossible for a password with a special character to be a calendar date, licence plate, or telephone number.

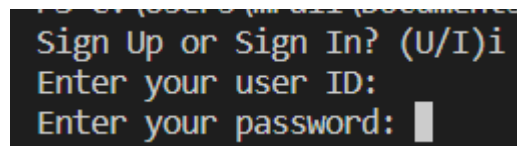
c) We can see from the screenshot when the user selects U as in sign-up, the interface the asks for user id, user name, password, and to select a role via a numbering system. The first two just need inputs, however for the password checker we create a loop for the input so it keeps asking the user for a password until the user inputs a valid password. All the check can be seen in the screenshot as well. The second While True loop is for selecting a number that is valid in regards to corresponding to one of the

role options listed on the interface. This can be seen from the screenshot in 3a. After all of those are done then the user will get a message that the account was created after the information was loaded into the password file as a new record.

```
option = input("Sign Up or Sign In? (U/I)")

if option == "U" or option == "u":
    userid = input("Enter your user ID: ")
    username = input("Enter your name: ")
    while True:
        password = input("Enter your password: ")
        if len(password) < 8:
            print("Password is too short")
        elif len(password) > 12:
            print("Password is too long")
        elif not any(x.isupper() for x in password):
            print("Password needs an uppercase")
        elif not any(x.islower() for x in password):
            print("Password needs an lowercase")
        elif not any(x.isdigit() for x in password):
            print("Password needs a number")
        elif not any(x in password for x in special):
            print("Password needs a special character from the set {!, @, #, $, %, ^, &, *}")
        elif any(x in password for x in common):
            print("Password is too common")
        elif password == userid or password == username:
            print("Password can't be same as User ID or Name")
        else:
            break
    while True:
        role = input("Enter your role (select 1 - 8):\n1. Regular Client\n2. Premium Client\n3. Teller\n4. Financial Advisor\n5. Compliance Officer\n6. Investment Analyst\n7. Loan Officer\n8. Branch Manager\n")
        if not role.isdigit():
            print("Please enter a number")
        elif int(role) > 8 or int(role) < 1:
            print("Please enter a valid number")
        else:
            role = list(roles.keys())[int(role) - 1]
            break
    print("Account Created.")
```

4. a) The login user interface will consist of 2 inputs asking for the user id and user password.



```
Sign Up or Sign In? (U/I)
Enter your user ID:
Enter your password:
```

b) The else in the screenshot is from the if option where the user entered U for sign up. So, if the user does not select to sign up to defaults them to the sign in page. After we get the user id and password, we open up the password file and check for a record with the entry of that user id (if userid in line). Then we split up the record by the colons to get an array with all our information. As done in Figure 2, we then take the inputted login password and hash with the plaintext salt in the record and see if it matches the hashed password in the record. If this is the case then we let the user know that access has been granted to their account.

```

else:
    userid = input("Enter your user ID: ")
    password = input("Enter your password: ")

    with open('passwd.txt') as f:
        lines = f.readlines()
        for line in lines:
            if userid in line:
                line = line.strip()
                arr = line.split(':')
                plaintext_salt = bytes.fromhex(arr[3])
                check = pbkdf2_hmac('sha256', password.encode('utf-8'), plaintext_salt, 500_000)
                if check.hex() == arr[2]:
                    print("\n\nACCESS GRANTED\nWelcome "+arr[1]+".\n")

```

C) This screenshotted code happens after the user is validated on login. We can then take the role of the user from the record and see all of the different actions and objects they have access to from the dictionary shown in 1 d and list them for the user. We then close the password file as we don't need to use it anymore.

```

if check.hex() == arr[2]:
    print("\n\nACCESS GRANTED\nWelcome "+arr[1]+".\n")
    print("You are a " + arr[4] + " and can view: " + ", ".join(roles[arr[4]]["view"]))
    print("\nand can modify: " + ", ".join(roles[arr[4]]["modify"]))
    print("\nand can validate modifications: " + ", ".join(roles[arr[4]]["validate modifications"]))
    print("\nand can view upon access: " + ", ".join(roles[arr[4]]["view upon access"]))
    f.close()

```

- d) I created a test account using sing up. It has the user id alifahd, name Ali Fahd, password of 1!SomeNumber, and the role of premium client (Selection 2). This led to the record entry alifahd:Ali Fahd:b095b10011579ce044fe7078cd975550d48bb5306003344a0d664a02270dfb72:1fe3b1847c5d774b5e582394442777d5:Premium Client in passwd.txt. I then logged in with the user ID and password and the login was successful and the rights accesses were also correct in accordance to the role user authentication created in Question 1.

```

Sign Up or Sign In? (U/I)
Enter your user ID: alifahd
Enter your password: 1!SomeNumber

ACCESS GRANTED
Welcome Ali Fahd.

You are a Premium Client and can view: Client Account Balance, Client Investment Portfolio, Contact Details of Financial Advisor, Contact Details of Financial Planner, Contact Details of Investment Analyst
and can modify: Client Investment Portfolio
and can validate modifications:
and can view upon access:

```

Summary:

To compile and run the prototype, in the terminal within the folder the files are located write: python system.py. This will need to be run each time you want to sign up or sign into an account.

We can see a sample case was generated here for the user Ali Fahd. The account was created successfully as a premium client.

```
Sign Up or Sign In? (U/I)u
Enter your user ID: afahd
Enter your name: Ali Fahd
Enter your password: !1SomeNumber
Enter your role (select 1 - 8):
1. Regular Client
2. Premium Client
3. Teller
4. Financial Advisor
5. Compliance Officer
6. Investment Analyst
7. Financial Planner
8. Technical Support
2
Account Created.
```

The following is the resulting record created in password.txt: afahd:Ali Fahd:8557efed0ad8aff0f24f6dffe6c795574233138bade8c4816413b9b0327e7d61:c1f11bc1a3c43ff04ebba760f4f01df4:Premium Client. We can see the correct information is here of user id, name, hashed password, plaintext salt, and the role selected.

We can then try logging into afahd with the password we created and we see all the access we have and they are correct in correlation to the requirements of the access control policy.

```
Sign Up or Sign In? (U/I)i
Enter your user ID: afahd
Enter your password: !1SomeNumber
```

```
ACCESS GRANTED
Welcome Ali Fahd.
```

```
You are a Premium Client and can view: Client Account Balance, Client Investment Portfolio, Contact Details of Financial Advisor, Contact Details of Financial Planner, Contact Details of Investment Analyst
```

```
and can modify: Client Investment Portfolio
```

```
and can validate modifications:
```

```
and can view upon access:
```

As we can see the system meets all the outlined requirements and has all the restrictions asked for implemented.