import hashlib #library used to compute hashes

import secrets #used to get a secure inital salt

import pickle #used to store objects

import os #used for clear screen command'''

#-----------------------

# Storage Locations

#-----------------------

USER\_PICKLE = 'user\_pickle.dat' #storage location of user data

ACL\_PICKLE = 'acl\_pickle.dat' #storage location of access control list ACL

#-----------------------

# Exceptions

#-----------------------

class AuthException(Exception):

    def \_\_init\_\_(self, username, user=None):

        super().\_\_init\_\_(username, user)

        self.username = username

        self.user = user

class UserAlreadyLoggedIn(AuthException):

    '''this exception shows, if a user

    tries to login, that is already logged in'''

    pass

class InvalidCredentials(AuthException):

    '''this exception shows, if a user

    enters unknown credentials'''

    pass

class UsernameAlreadyExists(AuthException):

    '''this excpetion shows, if a username

    already exists'''

    pass

class PasswordTooShort(AuthException):

    '''this exception shows, if a chosen

    password is too short'''

    pass

class PermissionExistsError(AuthException):

    pass

class PermissionError(AuthException):

    pass

class UserUnknown(AuthException):

    pass

#-----------------------

# Objects

#-----------------------

class User:

    def \_\_init\_\_(self, username, password, salt, medical\_history):

        '''the User class has username, password, medical\_history & salt as

        arguments. Salt is a random string, that is added prior hashing the

        password in a hopefully unknwon way -- keep source code hidden :P --

        to a hacker so the hash is more tricky to break. The arguments are

        stored via pickle.

        '''

        self.username = username

        self.salt = salt

        self.password = password

        self.medical\_history = medical\_history

    def read\_other\_patients\_medical\_history(self, username):

        '''This method is to view certain patient data. This method can view

        very sensitive information. Only certain users should be able to view

        it. This method has username as input. This function raises exceptions

        (e.g. if the user is not known). This function does not have a return

        value.'''

        users = []

        with open(USER\_PICKLE, 'rb') as f:

            users = (pickle.load(f)) #load objects from pickle

        user\_found = False

        for user in users:

            if user.username == username:

                print("The medical history of", username, "is:")

                print(user.medical\_history)

                user\_found = True

        if user\_found == False:

            raise UserUnknown(username)

    def read\_own\_medical\_history(self):

        '''This method is to view your patient data (if any). Since the method

        can only view your information, more users may be allowed to use this

        methods. This method has no input. This method has no return value '''

        print("Your medical history is:")

        for record in self.medical\_history:

            print(record)

class AccessControlListEntry:

    def \_\_init\_\_(self, username, user\_permissions):

        '''the AccessControlListEntry class is used to store

        a users permissions.

        an example would be: User 'Tim' is permitted to do

        'show\_patient\_history'

        The class does not have methods.'''

        self.username = username

        self.user\_permissions = user\_permissions

class Authenticator:

    def \_\_init\_\_(self):

        '''The Authenticator class stores all logged in users. This version of

        the ASMIS is not split in client and server side. This array will

        only ever store one user, since there are no multiple clients. Due to

        this this array will be used for a purpose other than intended. This

        array will not be used to identify all logged in users. This array will

        be used as the "client cookie" here. In a true client server setup this

        array of course may not be used as the client's cookie'''

        self.logged\_in\_users = []

    def \_salt\_hash\_pw(self, password, salt):

        '''The Authenticator can compute a salted hash for password storage.

        This method is meant to be used within this class only. It takes

        password and salt as argument. It will return the salted hash'''

        salted =  salt + password

        salted = salted.encode('utf8') #convert to utf8 for sha256 hashing

        salted\_hashed = hashlib.sha256(salted).hexdigest()

        return (salted\_hashed)

    def add\_user(self, username, password):

        '''The Authenticator can add users to the ASMIS. This method utilizes

        the afore mentioned method to salt hash the new password of the new

        usser. The method takes username & password as arguments.

        The method raises excpetions in case a username is already taken or

        if the password is to short. It has no return value'''

        users = []

        with open(USER\_PICKLE, 'rb') as f:

            users = (pickle.load(f)) #load objects from pickle

        for user in users:

            if user.username == username: #check if username is not yet taken

                raise UsernameAlreadyExists(username)

        if len(password) < 8: #check if password length is too short

            raise PasswordTooShort(username)

        ''' The password policy check can be extended by '''

        salt = (secrets.token\_hex(32)) #create new salt

        salted\_hashed = self.\_salt\_hash\_pw(password, salt) #salt hash new pw

        medical\_history = [] #new user starts with empty medical history

        users.append(User(username,salted\_hashed, salt, medical\_history))

        with open(USER\_PICKLE, 'wb') as f:

            pickle.dump(users, f) #store user objects in pickle

        ''' new user needs to be added to ACL, too'''

        acl = [] #flush array

        with open(ACL\_PICKLE, 'rb') as f:

            acl = (pickle.load(f)) #load objects from pickle

        new\_acl\_entry = AccessControlListEntry(username, [])

        acl.append(new\_acl\_entry) #append the current acl array

        with open(ACL\_PICKLE, 'wb') as f:

            pickle.dump(acl, f) #store objects in pickle

    def login(self, username, password):

        '''This method logins in a user if the credentials are correct and if

        the user is not already logged in. The method takes username & password

        as arguments. The method raises excpetions in case the credentials are

        not correct or if the user is already logged in. It has no return value

        '''

        if (self.is\_logged\_in(username)) == True: #do not login a user twice

            raise UserAlreadyLoggedIn(username)

        users = []

        with open(USER\_PICKLE, 'rb') as f:

            users = (pickle.load(f)) #load objects from pickle

        for user in users: #check if we have username & password match

            salted\_hashed = self.\_salt\_hash\_pw(password, user.salt)

            if user.username == username and salted\_hashed == user.password:

                user\_credentials\_correct = True

                self.logged\_in\_users.append(username)

                return #if we have a match the function is exited w/o exception

        raise InvalidCredentials(username) #w/o a match there is an exception

    def is\_logged\_in(self, username):

        '''this method checks if a user is logged in. It will take a username

        as an argument. It will return either True or False'''

        return\_value = False

        for user in self.logged\_in\_users:

            if user == username:

                return\_value = True

                break

            else:

                return\_value = False

        return return\_value

class Authorizor:

    def \_\_init\_\_(self):

        ''' the attribute permissions stores the global permissions any user

        can have. Note: The permissions are not stored in the backend within

        this test version of the ASMIS '''

        self.global\_permissions = ['add\_user',

        'add\_new\_global\_permission',

        'change\_user\_permissions',

        'read\_other\_patients\_medical\_history',

        'read\_own\_medical\_history',

        'is\_logged\_in',

        'print\_user\_permissions']

    def add\_new\_global\_permission (self, new\_permission):

        '''This method creates a new global permission that users

        can be obtain. This method takes new\_permission as an argument. If the

        new global permission already exists it raises an error. The mehtod has

        no return argument'''

        for permission in self.global\_permissions:

            if permission == new\_permission:

                raise PermissionExistsError(new\_permission)

        self.global\_permissions.append(new\_permission)

    def change\_user\_permissions (self, perm\_name, username, action):

        '''This mehtods can change permissions an individual user does have. It

        for example can either grant user xyz the privillege to access method

        zxy or it can revoke the privillege to do so. The method takes the name

        of the permission you want to change. It takes the action (revoke /

        provide). And it takes the username. The function has no return value.

        The function raises exception in case e.g. user or permission does not

        exist'''

        if perm\_name not in self.global\_permissions:

            raise PermissionError(username)

        acl = []

        with open(ACL\_PICKLE, 'rb') as f:

            acl = (pickle.load(f)) #load objects from pickle

        if action == '1':

            '''delete user permission'''

            for acl\_entry in acl:

                if acl\_entry.username == username:

                    print(acl\_entry.user\_permissions)

                    try:

                        acl\_entry.user\_permissions.remove(perm\_name)

                        print('Permission removed')

                        with open(ACL\_PICKLE, 'wb') as f:

                            pickle.dump(acl, f) #store objects in pickle

                    except:

                        print("this user does not have this permission")

                        print("No changes were made")

        elif action == '2':

            for acl\_entry in acl:

                if acl\_entry.username == username:

                    acl\_entry.user\_permissions.append(perm\_name)

                    with open(ACL\_PICKLE, 'wb') as f:

                        pickle.dump(acl, f) #store objects in pickle

                    print('Permission added')

    def print\_user\_permissions (self, username):

        '''This method prints permissions of a given user. The method takes

        the name of the user as an input. It has no return value. It raises an

        exception, if the user is not known.

        '''

        acl = []

        with open(ACL\_PICKLE, 'rb') as f:

            acl = (pickle.load(f)) #load objects from pickle

        user\_exists = False

        for acl\_entry in acl:

            if acl\_entry.username == username:

                print('The permission of ', username, " are:")

                print(acl\_entry.user\_permissions)

                user\_exists = True

                break

        if user\_exists == False:

            raise UserUnknown(username)

    def is\_authorized (self, user, task):

        '''This mehtods checks, if a given user authorized to do a certain

        task. The method takes the name of the task and the name of the user

        as input argument. The method will return True if the user is

        privileged to do the given task. The method will return False, if the

        user is not privilege to do the given task'''

        return\_value = False

        acl = []

        with open(ACL\_PICKLE, 'rb') as f:

            acl = (pickle.load(f)) #load objects from pickle

        for acl\_entry in acl:

            if acl\_entry.username == user:

                if task in acl\_entry.user\_permissions:

                    return\_value = True

        return return\_value

#-----------------------

# Some Auxiliary Functions

#-----------------------

# ----------------------

# clear command line window

# ----------------------

def screen\_clear():

   # for mac and linux(here, os.name is 'posix')

   if os.name == 'posix':

      \_ = os.system('clear')

   else:

      # for windows platfrom

      os.system('cls')