```python
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')
```