AUTHENTICATION and AUTHORIZATION

USER MANAGER

This document outlines the features of authentication and authorization of a user and stores the user data if it is valid

CONSTANTS.PY

Three constants have been defined for storing the user details

```
#----------Authentication and Authorization----
USERS = ['maheshk', 'yaswanthp', 'vigneshl']
ADMINS = ['prasad']
USER_DATA = {}
```

USER_MANAGER_UTILS.PY

Logger and re module is imported for storing the logs and matching the name pattern's that user has sent

```
from logger import *
import re
```

authenticate_user function is defined which takes name from the user and and constant user's in which user and admin details are stored. The function authenticate's the user details and return's TRUE if the user is valid else it raises the exception

```
def authenticate_user(name, users, admins):
    '''
    This function take's name, users and admins as argument and validates
    param : {name : str, users : dict, admins : dict}
    return : bool
    '''
    logging.debug(f"Authenticating for the user :- {name}")
    if (name in users) or (name in admins):
        logging.debug(f"User Authenticated succesfully")
        return True
    return False
```

is_valid_user take's the user_name, users and admins as argument's and validate's the user_name based on the below mentioned conditions and return's True if the user_name is valid else raises exception

```
def is_valid_user_name(user_name, users, admins):
    '''
    This function take's user_name as argument and validate's
    param : {name : str, users : dict, admins : dict}
```

```
    retunr : bool
    '''
    if (user_name in users)  or (user_name in admins):
        suggestions = [f"{user_name}{i}{user_name[:1]}" for i in range(6)]
        logging.error(f"user name {user_name} already exists. suggestions: {', '. join
        raise ValueError(f"user name {user_name} already exists.\nsuggestions: {', '.

    if len(user_name) < 8:
        logging.error(f"User name must me 8 character's long")
        raise ValueError(f"User name must me 8 character's long")

    if not re.search(r'[!@#$%^&*(),.?":{}|<>]', user_name):
        logging.error("Username must contain at least one special character")
        raise ValueError("Username must contain at least one special character")
    if not re.search(r'[0-9]', user_name):
        logging.error("Username must contain at least one number")
        raise ValueError("Username must contain at least one number")
    if not re.search(r'[A-Z]', user_name):
        logging.error("Username must contain at least one uppercase letter")
        raise ValueError("Username must contain at least one uppercase letter")

    return True
```

is_valid_name function takes takes name as argument and check's the pattern and if the pattern match's then return's True else raise ValueError

```
def is_valid_name( name):
    """
    This function takes name as argument
    param : name : str
    return : bool
    """
    if not name or not isinstance(name, str) or not re.match("^[a-zA-Z ]*$", name):
        raise ValueError("Name must be a non-empty alphabetic string")
    return True
```

is_valid_dept takes dept as argument and check's the if the dept is string or not and returns True is it is valid else raises exception

```
def is_valid_dept(dept):
    """
    This function takes dept as argument and validates
    param : dept : str
    return : bool
    """
    if not dept or not isinstance(dept, str):
        raise ValueError("Department must be a non-empty string")
    return True
```

is_valid_dob takes date of birth as argument and check's if it is in the DD-MON-YYYY format or not and returns True or valueerror based on the dob

```
def is_valid_dob(dob):
    """
    This function takes dept as argument and validates
    param : dob : str
    return : bool
    """
    if not re.match(r"\d{1,2}-\d{1,2}-\d{4}", dob):
        raise ValueError("DOB must be in the format DD-MM-YYYY")
    return True
```

AUTHENTICATION_AUTHORIZATION.PY

required constants are imported for validating and storing log's

```
from logger import *
from constants import USERS, ADMINS
from user_manager_utils import *
```

A class named **UserManager** is defined and constructor is defined for stroing the constants

```
class UserManager():
    '''
    This classs manage's the user's authentication and authorization
    '''
    def __init__(self):
        '''
        Constants are defined in the constructor
        '''
        self.users = USERS
        self.admins = ADMINS
```

create_user_role method take's the **user_name** and **user_role** as argument's initially and after successful validation, takes remaining user details as input's one by one and validates all the details and after successful validation , stores the user details in the dict format and returns the dict

```
def create_user_role(self, user_name, user_role):
        '''
        This method takes the user information and validates it
        param : user_role : str
        return : None
```

```
        '''
        logging.info(f"creating user details with the admin access {user_name}")
        user_name = input("Enter the User name here :-")
        is_valid_user_name(user_name, self.users, self.admins)

        firstname = input("Enter first name: ")
        is_valid_name(firstname)

        middlename = input("Enter middle name (optional): ")
        if middlename:
            is_valid_name(middlename)

        lastname = input("Enter last name (optional): ")
        if lastname:
            is_valid_name(lastname)

        dept = input("Enter the department: ")
        is_valid_dept(dept)

        dob = input("Enter the DOB (DD-MM-YYYY): ")
        is_valid_dob(dob)

        if user_role == 'a':
            is_admin = True
        else:
            is_admin = False

        user_info = {
            "username": user_name,
            "name": f"{firstname} {middlename} {lastname}".strip(),
            "dept": dept,
            "dob": dob,
            "isadmin": is_admin
        }

        return user_info
```

create_user takes name as initial argument and authenticates the user name and then after successful authentication, then the user role is taken as input and creates user details after successful validation and stores it to the users or admins based on the user role

```
def create_user(self, name):
        '''
        This method take's the argument name in the str format and create's user
        param : name : str
        return : None
        '''
        if not authenticate_user(name, self.users, self.admins):
            logging.error(f"Permission denied for the user {name}")
            raise Exception (f"Unauthorized user {name}, trying to access the applicat

        if name in self.admins:
```

```
            logging.debug(f"User {name} Authorized successfully")

            user_role = input("Enter the role ('a' for admin and 'n' for normal) for t
            if user_role in ['n','a']:
                user_info = self.create_user_role(name, user_role)
                if user_role == 'n':
                    self.users.append(user_info['username'])
                    logging.debug(f"{user_info['username']} added as normal user")
                else:
                    self.admins.append(user_info['username'])
                    logging.debug(f"{user_info['username']} added as an admin user")
                print(f"Final user info: {user_info}")
            else:
                logging.debug("Incorrect role chosen - available options are n and a c
                raise ValueError("Incorrect role chosen - available options are n and
        else:
            logging.debug(f"User {name} does not have create profile permission")
            raise PermissionError(f"User {name} does not have create profile permissio
```

LOGS

2024-06-27 17:26:44,672 - DEBUG - Authenticating for the user :- prasad
2024-06-27 17:26:44,672 - DEBUG - User Authenticated succesfully
2024-06-27 17:26:44,673 - DEBUG - User prasad Authorized successfully
2024-06-27 17:26:50,482 - INFO - creating user details with the admin access prasad
2024-06-27 17:27:49,751 - DEBUG - Authenticating for the user :- prasad
2024-06-27 17:27:49,752 - DEBUG - User Authenticated succesfully
2024-06-27 17:27:49,753 - DEBUG - User prasad Authorized successfully
2024-06-27 17:27:52,003 - INFO - creating user details with the admin access prasad
2024-06-27 17:28:48,506 - DEBUG - Authenticating for the user :- prasad
2024-06-27 17:28:48,507 - DEBUG - User Authenticated succesfully
2024-06-27 17:28:48,507 - DEBUG - User prasad Authorized successfully
2024-06-27 17:28:50,836 - INFO - creating user details with the admin access prasad
2024-06-27 17:29:14,167 - DEBUG - Mahesh@1989 added as an admin user
2024-06-27 17:54:53,425 - DEBUG - Authenticating for the user :- prasad
2024-06-27 17:54:53,425 - DEBUG - User Authenticated succesfully
2024-06-27 17:54:53,425 - DEBUG - User prasad Authorized successfully
2024-06-27 17:54:56,456 - INFO - creating user details with the admin access prasad
2024-06-27 17:55:03,366 - ERROR - user name maheshk already exists. suggestions: maheshk0m, maheshk1m, maheshk2m, maheshk3m, maheshk4m, maheshk5m
2024-06-27 19:27:53,337 - DEBUG - Authenticating for the user :- prasad
2024-06-27 19:27:53,337 - DEBUG - User Authenticated succesfully
2024-06-27 19:27:53,337 - DEBUG - User prasad Authorized successfully
2024-06-27 19:27:55,424 - INFO - creating user details with the admin access prasad
2024-06-27 19:28:00,273 - ERROR - Username must contain at least one special character
2024-06-27 19:28:04,641 - DEBUG - Authenticating for the user :- prasad
2024-06-27 19:28:04,641 - DEBUG - User Authenticated succesfully
2024-06-27 19:28:04,641 - DEBUG - User prasad Authorized successfully
2024-06-27 19:28:07,529 - INFO - creating user details with the admin access prasad
2024-06-27 19:28:25,069 - DEBUG - Authenticating for the user :- prasad

2024-06-27 19:28:25,069 - DEBUG - User Authenticated succesfully
2024-06-27 19:28:25,069 - DEBUG - User prasad Authorized successfully
2024-06-27 19:28:26,364 - INFO - creating user details with the admin access prasad
2024-06-27 19:28:52,809 - DEBUG - Yaswanth@1997 added as an admin user