import json

import getpass

import hashlib

from cryptography.fernet import Fernet

import logging

class PasswordManager:

def \_\_init\_\_(self, filename, master\_key):

self.filename = filename

self.passwords = {}

self.master\_key = master\_key

self.logger = self.\_setup\_logger()

def \_setup\_logger(self):

logger = logging.getLogger('PasswordManager')

logger.setLevel(logging.INFO)

formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')

file\_handler = logging.FileHandler('audit.log')

file\_handler.setFormatter(formatter)

logger.addHandler(file\_handler)

return logger

def create\_master\_password(self):

while True:

password = getpass.getpass("Create a master password: ")

confirm\_password = getpass.getpass("Confirm master password: ")

if password == confirm\_password:

self.master\_key = hashlib.sha256(password.encode()).digest()

self.logger.info('Master password created.')

print("Master password created successfully.")

break

else:

print("Passwords do not match. Please try again.")

def authenticate(self):

password = getpass.getpass("Enter master password: ")

hashed\_password = hashlib.sha256(password.encode()).digest()

if hashed\_password == self.master\_key:

return True

else:

print("Incorrect master password.")

return False

def create\_password(self, website, username, password):

if self.authenticate() and self.\_has\_permission('create\_password'):

if website in self.passwords:

print("Password already exists for this website.")

else:

if self.\_is\_strong\_password(password):

self.passwords[website] = {

'username': username,

'password': password

}

self.\_save\_passwords()

self.logger.info('Password created for website: %s', website)

print("Password created successfully.")

else:

print("Password does not meet the requirements.")

else:

print("Access denied. You do not have permission to create a password.")

def get\_password(self, website):

if self.authenticate() and self.\_has\_permission('get\_password'):

if website in self.passwords:

password = self.passwords[website]['password']

self.logger.info('Password retrieved for website: %s', website)

print("Password:", password)

else:

print("Password not found for this website.")

else:

print("Access denied. You do not have permission to retrieve passwords.")

def delete\_password(self, website):

if self.authenticate() and self.\_has\_permission('delete\_password'):

if website in self.passwords:

confirm = input("Are you sure you want to delete the password for {}? (yes/no): ".format(website))

if confirm.lower() == 'yes':

del self.passwords[website]

self.\_save\_passwords()

self.logger.info('Password deleted for website: %s', website)

print("Password deleted successfully.")

else:

print("Deletion cancelled.")

else:

print("Password not found for this website.")

else:

print("Access denied. You do not have permission to delete passwords.")

def reset\_master\_password(self):

if self.authenticate() and self.\_has\_permission('reset\_master\_password'):

while True:

new\_password = getpass.getpass("Enter a new master password: ")

confirm\_password = getpass.getpass("Confirm new master password: ")

if new\_password == confirm\_password:

self.master\_key = hashlib.sha256(new\_password.encode()).digest()

self.logger.info('Master password reset.')

print("Master password reset successfully.")

break

else:

print("Passwords do not match. Please try again.")

else:

print("Access denied. You do not have permission to reset the master password.")

def \_has\_permission(self, action):

# Implement RBAC logic to check if the user has permission for the action

# Return True if the user has permission, False otherwise

# You can customize this method to implement RBAC based on user roles and permissions

# Example implementation:

# if user\_role == 'admin' and action in ['create\_password', 'get\_password', 'delete\_password', 'reset\_master\_password']:

# return True

# elif user\_role == 'user' and action in ['get\_password']:

# return True

# else:

# return False

return True

def \_is\_strong\_password(self, password):

# Implement your own password strength requirements here

# For example, check length, complexity, and other criteria

return len(password) >= 8

def \_save\_passwords(self):

encrypted\_passwords = self.\_encrypt\_data(json.dumps(self.passwords))

with open(self.filename, 'wb') as file:

file.write(encrypted\_passwords)

def \_load\_passwords(self):

try:

with open(self.filename, 'rb') as file:

encrypted\_passwords = file.read()

self.passwords = json.loads(self.\_decrypt\_data(encrypted\_passwords))

except FileNotFoundError:

self.passwords = {}

def \_encrypt\_data(self, data):

cipher = Fernet(self.master\_key)

encrypted\_data = cipher.encrypt(data.encode())

return encrypted\_data

def \_decrypt\_data(self, encrypted\_data):

cipher = Fernet(self.master\_key)

decrypted\_data = cipher.decrypt(encrypted\_data)

return decrypted\_data.decode()

def main():

filename = 'passwords.dat'

master\_key = None

password\_manager = PasswordManager(filename, master\_key)

password\_manager.create\_master\_password()

password\_manager.\_load\_passwords()

while True:

print("\nPassword Manager")

print("1. Create a new password")

print("2. Retrieve a password")

print("3. Delete a password")

print("4. Reset master password")

print("5. Quit")

choice = input("Enter your choice: ")

if choice == '1':

website = input("Enter the website: ")

username = input("Enter the username: ")

password = getpass.getpass("Enter the password: ")

password\_manager.create\_password(website, username, password)

elif choice == '2':

website = input("Enter the website: ")

password\_manager.get\_password(website)

elif choice == '3':

website = input("Enter the website: ")

password\_manager.delete\_password(website)

elif choice == '4':

password\_manager.reset\_master\_password()

elif choice == '5':

break

else:

print("Invalid choice. Please try again.")

if \_\_name\_\_ == '\_\_main\_\_':

main()