import os

import json

import hashlib

from cryptography.fernet import Fernet

from datetime import datetime

# ================================

#  CONFIGURATION

# ================================

BASE\_DIR = os.path.expanduser("~")  # User's home folder

DATA\_FOLDER = os.path.join(BASE\_DIR, "MemoirData")

KEY\_FILE = os.path.join(DATA\_FOLDER, "secret.key")

DATA\_FILE = os.path.join(DATA\_FOLDER, "memories.json")

PIN\_FILE = os.path.join(DATA\_FOLDER, "pin.json")

# Ensure folder exists

os.makedirs(DATA\_FOLDER, exist\_ok=True)

# ================================

#  ENCRYPTION FUNCTIONS

# ================================

def load\_key():

    """Load or create encryption key"""

    if not os.path.exists(KEY\_FILE):

        key = Fernet.generate\_key()

        with open(KEY\_FILE, "wb") as f:

            f.write(key)

    else:

        with open(KEY\_FILE, "rb") as f:

            key = f.read()

    return Fernet(key)

fernet = load\_key()

# ================================

#  PIN HANDLING

# ================================

def hash\_pin(pin: str) -> str:

    """Hash PIN using SHA256 (never store plain PIN)"""

    return hashlib.sha256(pin.encode()).hexdigest()

def setup\_pin():

    """Setup master PIN on first run"""

    if not os.path.exists(PIN\_FILE):

        print("\nSet up your MEMOIR master PIN:")

        while True:

            pin = input("Enter a new PIN: ")

            confirm = input("Confirm your PIN: ")

            if pin == confirm and pin.strip():

                with open(PIN\_FILE, "w") as f:

                    json.dump({"pin\_hash": hash\_pin(pin)}, f)

                print("PIN set successfully!\n")

                break

            else:

                print("PINs do not match. Try again.")

    else:

        with open(PIN\_FILE, "r") as f:

            saved = json.load(f)

        for attempt in range(3):

            pin = input("Enter your MEMOIR PIN: ")

            if hash\_pin(pin) == saved["pin\_hash"]:

                print("Access granted!\n")

                return

            else:

                print("Wrong PIN.")

        print("Too many wrong attempts. Exiting...")

        exit()

# ================================

#  MEMORY FUNCTIONS

# ================================

def load\_memories():

    if not os.path.exists(DATA\_FILE):

        return []

    with open(DATA\_FILE, "r") as f:

        return json.load(f)

def save\_memories(memories):

    with open(DATA\_FILE, "w") as f:

        json.dump(memories, f, indent=4)

def add\_memory():

    text = input("Enter your memory: ")

    unlock\_date = input("Enter unlock date (YYYY-MM-DD): ")

    encrypted\_text = fernet.encrypt(text.encode()).decode()

    memories = load\_memories()

    memories.append({

        "memory": encrypted\_text,

        "unlock\_date": unlock\_date

    })

    save\_memories(memories)

    print("Memory saved!")

def view\_specific\_memory():

    memories = load\_memories()

    today = datetime.today().date()

    if not memories:

        print("No memories saved yet.")

        return

    # Show list of memories with unlock date

    print("\n===== Your Memories =====")

    for i, mem in enumerate(memories, start=1):

        unlock\_date = datetime.strptime(mem["unlock\_date"], "%Y-%m-%d").date()

        status = "Unlocked" if today >= unlock\_date else f"Locked until {unlock\_date}"

        print(f"{i}. Memory ({status}) - Unlock date: {unlock\_date}")

    try:

        choice = int(input("\nEnter the memory number to view: "))

        if 1 <= choice <= len(memories):

            mem = memories[choice - 1]

            unlock\_date = datetime.strptime(mem["unlock\_date"], "%Y-%m-%d").date()

            if today >= unlock\_date:

                decrypted = fernet.decrypt(mem["memory"].encode()).decode()

                print(f"\nMemory {choice}: {decrypted}")

            else:

                print(f"This memory is locked until {unlock\_date}.")

        else:

            print("Invalid choice.")

    except ValueError:

        print("Please enter a valid number.")

def delete\_memory():

    memories = load\_memories()

    if not memories:

        print("No memories to delete.")

        return

    # Show list of memories

    print("\n===== Delete a Memory =====")

    for i, mem in enumerate(memories, start=1):

        print(f"{i}. Unlock date: {mem['unlock\_date']}")

    try:

        choice = int(input("\nEnter the memory number to delete: "))

        if 1 <= choice <= len(memories):

            confirm = input(f"Are you sure you want to delete memory {choice}? (y/n): ").lower()

            if confirm == "y":

                deleted = memories.pop(choice - 1)

                save\_memories(memories)

                print("Memory deleted successfully.")

            else:

                print("Deletion cancelled.")

        else:

            print("Invalid choice.")

    except ValueError:

        print("Please enter a valid number.")

# ================================

#  MAIN MENU

# ================================

def main\_menu():

    while True:

        print("\n===== MEMOIR - Your Digital Diary =====")

        print("1. Add a memory")

        print("2. View a memory")

        print("3. Delete a memory")

        print("4. Exit")

        choice = input("Enter choice: ")

        if choice == "1":

            add\_memory()

        elif choice == "2":

            view\_specific\_memory()

        elif choice == "3":

            delete\_memory()

        elif choice == "4":

            print("Goodbye! Stay safe with your memories.")

            break

        else:

            print("Invalid choice, try again.")

# ================================

#  PROGRAM START

# ================================

if \_\_name\_\_ == "\_\_main\_\_":

    setup\_pin()

    main\_menu()