```
import json
import secrets
import os
import random
import base64
from getpass import getpass
from typing import Dict, List, Optional
from dataclasses import dataclass, asdict
from datetime import datetime
from pyfiglet import Figlet
from colorama import Fore, Style, init
from cryptography.fernet import Fernet
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from argon2 import PasswordHasher
from enum import Enum
import sys

init(autoreset=True)

ph = PasswordHasher(
    time_cost=16,
    memory_cost=2**16,
    parallelism=2,
    hash_len=32,
    salt_len=32
)

def print_banner(message: str):
    fonts = ['chunky','bolger', 'cosmic', 'rowancap']
    f = Figlet(font=random.choice(fonts))
    print(Fore.BLUE + f.renderText(message) + Style.RESET_ALL)

def derive_key(password: str, salt: bytes) -> bytes:
    """Derive a key from the password using PBKDF2"""
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=100000,
    )
    key = base64.urlsafe_b64encode(kdf.derive(password.encode()))
    return key

class TaskStatus(Enum):
    PENDING = "Pending"
    COMPLETED = "Completed"
    OTHER = "Other"

def get_status_color(status: str) -> str:
    """Get the color code for a task status"""
    status = status.lower()
    if status == "completed":
        return Fore.GREEN
    elif status == "pending":
        return Fore.RED
    else:
        return Fore.YELLOW

@dataclass
class Task:
    id: int
    description: str
```

```
    status: str
    created_at: str
    user_id: str

    def get_color(self) -> str:
        """Get the color code for this task's status"""
        return get_status_color(self.status)

class TaskManager:
    def __init__(self):
        self.users_file = "users.json"
        self.tasks_file = "tasks.json"
        self.current_user: Optional[str] = None
        self._load_data()

    def _load_data(self):
        """Load users and tasks from files"""
        if os.path.exists(self.users_file):
            with open(self.users_file, 'r') as f:
                self.users = json.load(f)
        else:
            self.users = {}

        if os.path.exists(self.tasks_file):
            with open(self.tasks_file, 'r') as f:
                self.tasks = json.load(f)
        else:
            self.tasks = {}

    def _save_data(self):
        """Save users and tasks to files"""
        with open(self.users_file, 'w') as f:
            json.dump(self.users, f)
        with open(self.tasks_file, 'w') as f:
            json.dump(self.tasks, f)

    def _hash_password(self, password: str, salt: bytes = None) -> str:
        """Hash password using Argon2"""
        # Argon2 will generate its own salt if none provided
        return ph.hash(password)

    def _verify_password(self, password: str, hash_str: str) -> bool:
        """Verify password using Argon2"""
        try:
            ph.verify(hash_str, password)
            return True
        except:
            return False

    def _encrypt_task(self, task: Task, password: str) -> str:
        """Encrypt a task using the user's password with AES-256"""
        try:
            salt = base64.b64decode(self.users[self.current_user]['salt'])
            key = derive_key(password, salt)
            f = Fernet(key)
            task_dict = asdict(task)
            task_json = json.dumps(task_dict)
            encrypted_data = f.encrypt(task_json.encode())
            return base64.b64encode(encrypted_data).decode()
        except Exception as e:
            print(f"Encryption error: {str(e)}")
            raise

    def _decrypt_task(self, encrypted_task: str, password: str) -> Task:
        """Decrypt a task using the user's password with AES-256"""
```

```
            try:
                salt = base64.b64decode(self.users[self.current_user]['salt'])
                key = derive_key(password, salt)
                f = Fernet(key)
                encrypted_data = base64.b64decode(encrypted_task.encode())
                decrypted_data = f.decrypt(encrypted_data)
                task_dict = json.loads(decrypted_data.decode())
                return Task(**task_dict)
            except Exception as e:
                print(f"Decryption error: {str(e)}")
                raise

        def _get_next_task_id(self) -> int:
            """Get the next available task ID for the current user"""
            if not self.tasks[self.current_user]:
                return 1

            # Decrypt all tasks to get their IDs
            max_id = 0
            for encrypted_task in self.tasks[self.current_user]:
                try:
                    task = self._decrypt_task(encrypted_task, self.current_password)
                    max_id = max(max_id, task.id)
                except Exception:
                    continue
            return max_id + 1

        def register(self, username: str, password: str) -> bool:
            """Register a new user"""
            if username in self.users:
                print("Username already exists!")
                return False

            # Generate a strong salt for encryption key derivation (separate from Argon2's salt)
            salt = secrets.token_bytes(32)
            salt_b64 = base64.b64encode(salt).decode()

            # Hash the password with Argon2
            password_hash = self._hash_password(password)

            self.users[username] = {
                'password_hash': password_hash,
                'salt': salt_b64  # This salt is used for task encryption, not password hashing
            }
            self.tasks[username] = []
            self._save_data()

            # Automatically log in the user after registration
            self.current_user = username
            self.current_password = password
            print(Fore.GREEN + "Registration successful!" + Style.RESET_ALL)
            print(f"Welcome, {username}!")
            return True

        def login(self, username: str, password: str) -> bool:
            """Login a user"""
            if username not in self.users:
                print("Username not found!")
                return False

            if not self._verify_password(password, self.users[username]['password_hash']):
                print("Invalid password!")
                return False

            self.current_user = username
```

```
        self.current_password = password  # Store password temporarily for
encryption/decryption
        print("Login successful!")
        return True

    def logout(self):
        """Logout current user"""
        self.current_user = None
        self.current_password = None
        print("Logged out successfully!")

    def add_task(self, description: str) -> bool:
        """Add a new task"""
        if not self.current_user:
            print("Please login first!")
            return False

        task_id = self._get_next_task_id()
        task = Task(
            id=task_id,
            description=description,
            status="Pending",
            created_at=datetime.now().isoformat(),
            user_id=self.current_user
        )

        encrypted_task = self._encrypt_task(task, self.current_password)
        self.tasks[self.current_user].append(encrypted_task)
        self._save_data()
        print(f"Task added successfully with ID: {task_id}")
        return True

    def view_tasks(self) -> bool:
        """View all tasks for current user"""
        if not self.current_user:
            print("Please login first!")
            return False

        if not self.tasks[self.current_user]:
            print("No tasks found!")
            return False

        print("\nYour Tasks:")
        print("-" * 50)
        for encrypted_task in self.tasks[self.current_user]:
            try:
                task = self._decrypt_task(encrypted_task, self.current_password)
                color = task.get_color()
                print(color + f"""ID: {task.id}
Description: {task.description}
Status: {task.status}
Created: {task.created_at}""" + Style.RESET_ALL)
                print("-" * 50)
            except Exception as e:
                print(f"Error decrypting task: {e}")
                continue
        print("\nPress any key to continue...")
        wait_key()
        return True

    def show_task_list(self) -> bool:
        """Show a condensed list of tasks"""
        if not self.current_user:
            print("Please login first!")
            return False
```

```
        if not self.tasks[self.current_user]:
            print("No tasks found!")
            return False

        print("\nAvailable Tasks:")
        for encrypted_task in self.tasks[self.current_user]:
            try:
                task = self._decrypt_task(encrypted_task, self.current_password)
                color = task.get_color()
                print(color + f"{task.id} - {task.description}" + Style.RESET_ALL)
            except Exception as e:
                print(f"Error decrypting task: {e}")
                continue
        return True

    def mark_completed(self, task_id: int) -> bool:
        """Mark a task as completed"""
        if not self.current_user:
            print("Please login first!")
            return False

        print()
        for i, encrypted_task in enumerate(self.tasks[self.current_user]):
            try:
                task = self._decrypt_task(encrypted_task, self.current_password)
                if task.id == task_id:
                    task.status = "Completed"
                    self.tasks[self.current_user][i] = self._encrypt_task(task,
self.current_password)
                    self._save_data()
                    print(f"Task {task_id} marked as completed!")
                    return True
            except Exception as e:
                print(f"Error decrypting task: {e}")
                continue

        print(f"Task with ID {task_id} not found!")
        return False

    def delete_task(self, task_id: int) -> bool:
        """Delete a task"""
        if not self.current_user:
            print("Please login first!")
            return False

        print()
        for i, encrypted_task in enumerate(self.tasks[self.current_user]):
            try:
                task = self._decrypt_task(encrypted_task, self.current_password)
                if task.id == task_id:
                    self.tasks[self.current_user].pop(i)
                    self._save_data()
                    print(f"Task {task_id} deleted successfully!")
                    return True
            except Exception as e:
                print(f"Error decrypting task: {e}")
                continue

        print(f"Task with ID {task_id} not found!")
        return False

    def get_task_summary(self) -> tuple[int, int]:
        """Get summary of pending and completed tasks"""
        pending = 0
```

```
        completed = 0

        if not self.tasks[self.current_user]:
            return (0, 0)

        for encrypted_task in self.tasks[self.current_user]:
            try:
                task = self._decrypt_task(encrypted_task, self.current_password)
                if task.status == "Completed":
                    completed += 1
                elif task.status == "Pending":
                    pending += 1
            except Exception:
                continue

        return (pending, completed)

def wait_key():
    """Wait for a key press on Unix/Windows/MacOS"""
    try:
        # For Windows
        import msvcrt
        msvcrt.getch()
    except ImportError:
        # For Unix/Linux/MacOS
        import tty, termios
        fd = sys.stdin.fileno()
        old_settings = termios.tcgetattr(fd)
        try:
            tty.setraw(sys.stdin.fileno())
            sys.stdin.read(1)
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)

def main():
    task_manager = TaskManager()

    while True:
        if not task_manager.current_user:
            print_banner("Task Manager")
            print(Fore.CYAN + "A Simple CLI Task Manager" + Style.RESET_ALL)
            print("Main Menu:")
            print("1. Register")
            print("2. Login")
            print("3. Exit")
            choice = input("Enter your choice (1-3): ")
            if choice == "1":
                username = input("Enter username: ")
                password = getpass("Enter password: ")
                task_manager.register(username, password)
            elif choice == "2":
                username = input("Enter username: ")
                password = getpass("Enter password: ")
                task_manager.login(username, password)
            elif choice == "3":
                print("Goodbye!")
                break
            else:
                print("Invalid choice! Please try again.")
        else:
            print_banner(f"Hi, {task_manager.current_user}!")
            print(f"\nHi, {Fore.LIGHTCYAN_EX}{task_manager.current_user}{Style.RESET_ALL},
lets get started on some tasks!")
            pending, completed = task_manager.get_task_summary()
            total = pending + completed
```

```
            print(f"You have {total} tasks")
            print(f"In Progress: {Fore.RED}{pending}{Style.RESET_ALL} Completed:
{Fore.GREEN}{completed}{Style.RESET_ALL}")
            print("\nTask Manager Menu:")
            print("1. Add Task")
            print("2. View Tasks")
            print("3. Mark Task as Completed")
            print("4. Delete Task")
            print("5. Logout")
            choice = input("Enter your choice (1-5): ")

            if choice == "1":
                description = input("Enter task description: ")
                task_manager.add_task(description)
            elif choice == "2":
                print_banner("Tasks")
                task_manager.view_tasks()
            elif choice == "3":
                task_manager.show_task_list()
                task_id = int(input("Enter task ID to mark as completed: "))
                task_manager.mark_completed(task_id)
            elif choice == "4":
                task_manager.show_task_list()
                task_id = int(input("Enter task ID to delete: "))
                task_manager.delete_task(task_id)
            elif choice == "5":
                task_manager.logout()
            else:
                print("Invalid choice! Please try again.")

if __name__ == "__main__":
    main()
```