НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Кафедра інформаційних систем та технологій

Тема: PowerShell terminal

Курсова робота
З дисципліни «Технології розроблення програмного забезпечення»

Керівник

доц. Амонс О.А.

«Допущений до захисту»

_____

(Особистий підпис керівника)

« » _____2025р.

Захищений з оцінкою

_____

(оцінка)

Члени комісії:

_____ _____

(особистий підпис) (розшифровка підпису)

_____ _____

(особистий підпис) (розшифровка підпису)

Виконавець

ст. Олея М. С.

залікова книжка № ___ – _____

гр. ІА-31

_____

(особистий підпис виконавця)

« »_____2025р.

Київ – 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
(назва навчального закладу)

Кафедра ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Дисципліна «Технології розроблення програмного забезпечення»

Курс 3 Група ІА-31 Семестр 5

ЗАВДАННЯ

на курсову роботу студента

Олеї Михайла Сергійовича

(прізвище, ім’я, по батькові)

1.Тема роботи: PowerShell terminal

2. Строк здачі студентом закінченої роботи

3. Вихідні дані до роботи:

Термінал для powershell повинен нагадувати типовий термінал з можливістю налаштування кольорів синтаксичних конструкцій, розміру вікна, фону вікна, а також виконання команд powershell і виконуваних файлів, а також працювати в декількох вікнах терміналу (у вкладках або одночасно шляхом розділення вікна).

4. Зміст розрахунково – пояснювальної записки (перелік питань, що підлягають розробці)

1. Аналіз предметної області, загальний опис проєкту та огляд існуючих рішень. 2. Формулювання функціональних та нефункціональних вимог до системи. 3. Розробка сценаріїв використання (Use Case) та концептуальної моделі системи. 4. Проєктування архітектури системи, вибір технологічного стеку та баз даних.

Додатки:

Додаток А – режим доступу до репозиторію

Додаток Б - проектування паттернів

5.Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Рис. 1.1 — Діаграма варіантів використання; Рис. 1.2–1.4 — Діаграми послідовності; Рис. 1.5 – 1.7 — Діаграми класів; Рис. 1.8 — Діаграма компонентів; Рис. 1.9 — Діаграма розгортання; Рис. 2.1 – Проектування бази даних; Рис. 2.2 – 2.6 – структура паттернів програмування

6. Дата видачі завдання 17.09.2025

КАЛЕНДАРНИЙ ПЛАН

№, п/п	Назва етапів виконання курсової роботи	Строк виконання етапів роботи	Підписи або примітки
1.	Підбір та вивчення літератури	30.09.2025	
2.	Проектування та написання розділу 1	31.10.2025	
3.	Розробка та написання розділу 2	20.11.2025	
4.	Подання курсової роботи на перевірку	25.11.2025	
5.	Захист курсової роботи	08.12.2025	
6.			
7.			
8.			
9.			
10.			
11.			
12.			
13.			
14.			
15.			
16.			
17.			
18.			

Студент _____ _____Михайло ОЛЕЯ_____
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник_____ _Олександр АМОНС___
(підпис) (Ім’я ПРІЗВИЩЕ)

«____»_____ 20___ р.

ЗМІСТ

ВСТУП

Сучасне системне адміністрування та розробка програмного забезпечення неможливі без ефективних інструментів командного рядка (CLI). Термінали та командні оболонки займають центральне місце в управлінні операційними системами, автоматизації процесів та роботі з серверною інфраструктурою. Незважаючи на наявність стандартних рішень, таких як Windows PowerShell або CMD, існує постійний попит на створення універсальних, кастомізованих середовищ, що поєднують зручність графічного інтерфейсу з потужністю консольних команд.

Актуальність даної роботи полягає у необхідності розробки гнучкого, розширюваного програмного засобу для роботи з командним рядком, який інтегрує різні середовища виконання в єдиному інтерфейсі. Проєктування таких систем вимагає вирішення складних архітектурних завдань: забезпечення ізольованості процесів, підтримки віддаленого виконання команд, збереження стану сесії та реалізації механізмів безпеки. Неефективна архітектура в такому проєкті може призвести до неможливості масштабування функціоналу, проблем із багатопотоковістю та складнощів у підтримці коду.

Метою даного курсового проєкту є проєктування та розробка програмної системи "PowerShell Terminal". Система має виконувати функції багатофункціонального емулятора терміналу з підтримкою вкладок, забезпечувати виконання команд різних оболонок (PowerShell, CMD), вести персистентну історію команд користувача, а також надавати можливості для візуального налаштування інтерфейсу (теми, підсвітка синтаксису) та розмежування прав доступу (адміністратор/користувач).

Для досягнення поставленої мети особливу увагу буде приділено архітектурі системи та застосуванню сучасних патернів проєктування. Це дозволить створити слабкозв'язану (loosely coupled) систему, готову до майбутніх модифікацій. Проєкт буде реалізовано з використанням платформи .NET та технології Windows Forms, а також клієнт-серверної архітектури для реалізації віддаленого виконання команд.

У даній роботі буде проведено аналіз предметної області, розроблено архітектуру системи з використанням UML-діаграм, описано реалізацію ключових функціональних можливостей та продемонстровано практичне застосування таких патернів проєктування, як Strategy (для зміни тем), Command (для інкапсуляції запитів), Abstract Factory (для створення рольового інтерфейсу), Bridge (для підтримки різних рушіїв виконання), Interpreter (для обробки власних команд) та Client-Server, для вирішення конкретних завдань у межах проєкту.

1 ПРОЄКТУВАННЯ СИСТЕМИ

1.1. Огляд існуючих рішень

Сфера інструментів командного рядка (CLI) та емуляторів терміналів сьогодні є надзвичайно розвиненою, оскільки це основний робочий інструмент для системних адміністраторів, DevOps-інженерів та розробників програмного забезпечення. Ринок представлений як вбудованими системними рішеннями, так і потужними сторонніми продуктами, що розширюють базовий функціонал ОС. Серед основних категорій існуючих рішень можна виділити:

- Стандартні системні консолі: До цієї групи належать класичні Windows Console Host (conhost.exe), CMD та оригінальний інтерфейс Windows PowerShell. Це базові інструменти, що постачаються разом з операційною системою.

Переваги: Максимальна стабільність, доступність на будь-якому ПК без встановлення, низьке споживання ресурсів.

Недоліки: Застарілий інтерфейс, відсутність підтримки вкладок (tabs), обмежені можливості кастомізації (шрифти, теми, прозорість) та незручна робота з буфером обміну.

- Сучасні емулятори терміналів: Найяскравішим представником є Windows Terminal, а також такі проєкти, як Cmder, ConEmu та Alacritty. Ці програми є оболонками, що дозволяють запускати різні рушії (PowerShell, WSL, CMD) в єдиному сучасному інтерфейсі.

Переваги: Підтримка вкладок та розділення екрана (split panes), апаратне прискорення рендерингу тексту, широкі можливості налаштування через JSON-файли, підтримка Unicode та емодзі.

Недоліки: Часто мають високі системні вимоги порівняно з консоллю, складну конфігурацію для пересічного користувача та надлишкову функціональність для простих завдань.

- Інтегровані термінали (IDE): Термінали, вбудовані в середовища розробки, такі як Visual Studio Code або Visual Studio.

Переваги: Тісна інтеграція з файловою системою проєкту, автоматичне визначення оточення.

Недоліки: Прив'язка до конкретного редактора коду, неможливість використання як окремого легкого інструменту для адміністрування.

Аналіз існуючих рішень показує, що хоча сучасні термінали (як Windows Terminal) вирішують більшість проблем UI, існує брак навчально-демонстраційних систем, які б поєднували гнучкість налаштування з прозорою архітектурою. Більшість існуючих рішень не надають вбудованих інструментів для збереження історії команд у реляційну базу даних (для аудиту та аналітики) або можливості симуляції рольового доступу (Admin/User) без втручання в налаштування ОС. Саме реалізація такої системи з використанням сучасних патернів проєктування і є метою даного проєкту.

1.2. Загальний опис проєкту

Проєкт присвячений розробці програмної системи "PowerShell Terminal", призначеної для організації універсального середовища виконання команд та скриптів. На відміну від стандартних системних консолей (як CMD) або важковагових середовищ розробки (IDE), дана система проєктується як гнучке модульне рішення з розширеними можливостями візуалізації, збереженням стану сесій та рольовою моделлю доступу.

Система базується на багатошаровій архітектурі з використанням елементів клієнт-серверної взаємодії.

- Ядро системи (Domain Logic): Побудоване на базі платформи .NET, виступає центральним вузлом, який реалізує ключові патерни проєктування. Воно відповідає за інтерпретацію введених команд (Interpreter), вибір рушія виконання (Bridge), керування темами оформлення (Strategy) та взаємодію з базою даних.
- Підсистема даних (Data Layer): Використовує локальну реляційну базу даних (SQLite) через ORM Entity Framework Core. Вона забезпечує персистентне

збереження історії команд, профілів користувачів та налаштувань між перезапусками програми.

- Інтерфейс (UI Layer): Реалізований на Windows Forms із використанням кастомізованих компонентів (RichTextBox) для забезпечення функцій "живої" підсвітки синтаксису та підтримки багатовіконності через вкладки.

Проєкт чітко розмежовує функціонал та права доступу на основі двох ключових ролей, інтерфейс для яких формується динамічно (патерн Abstract Factory):

1. Адміністратор (Admin): Володіє повними правами на керування системою. Інтерфейс адміністратора візуально відрізняється (кольорова схема, системні повідомлення) для запобігання помилкам.
    - Доступ до налаштувань: Адміністратор має ексклюзивне право змінювати глобальні теми оформлення терміналу.
    - Необмежене виконання: Має доступ до виконання всіх системних команд, включно з потенційно деструктивними операціями.
    - Аудит: Має доступ до повної історії команд.
2. Користувач (User): Стандартна роль для повсякденної роботи. Функціонал користувача спрямований на безпечне виконання завдань:
    - Обмежене середовище: Система автоматично блокує виконання небезпечних команд (наприклад, видалення файлів або форматування дисків) через механізм проксі-валідації.
    - Ізольовані налаштування: Користувач працює у встановленій темі без права зміни конфігурації інтерфейсу.

Основний сценарій роботи передбачає повний цикл обробки команди: від введення тексту з "живою" підсвіткою синтаксису до його лексичного розбору внутрішнім інтерпретатором. Система автоматично визначає, чи є команда внутрішньою (наприклад, навігація cd або перегляд історії history), чи її слід передати зовнішньому рушію (PowerShell/CMD/Remote Server) через механізм моста (Bridge). Результат виконання автоматично логується в базу даних та виводиться користувачеві у відформатованому вигляді.

1.3. Вимоги до застосунків системи

1.3.1. Функціональні вимоги до системи

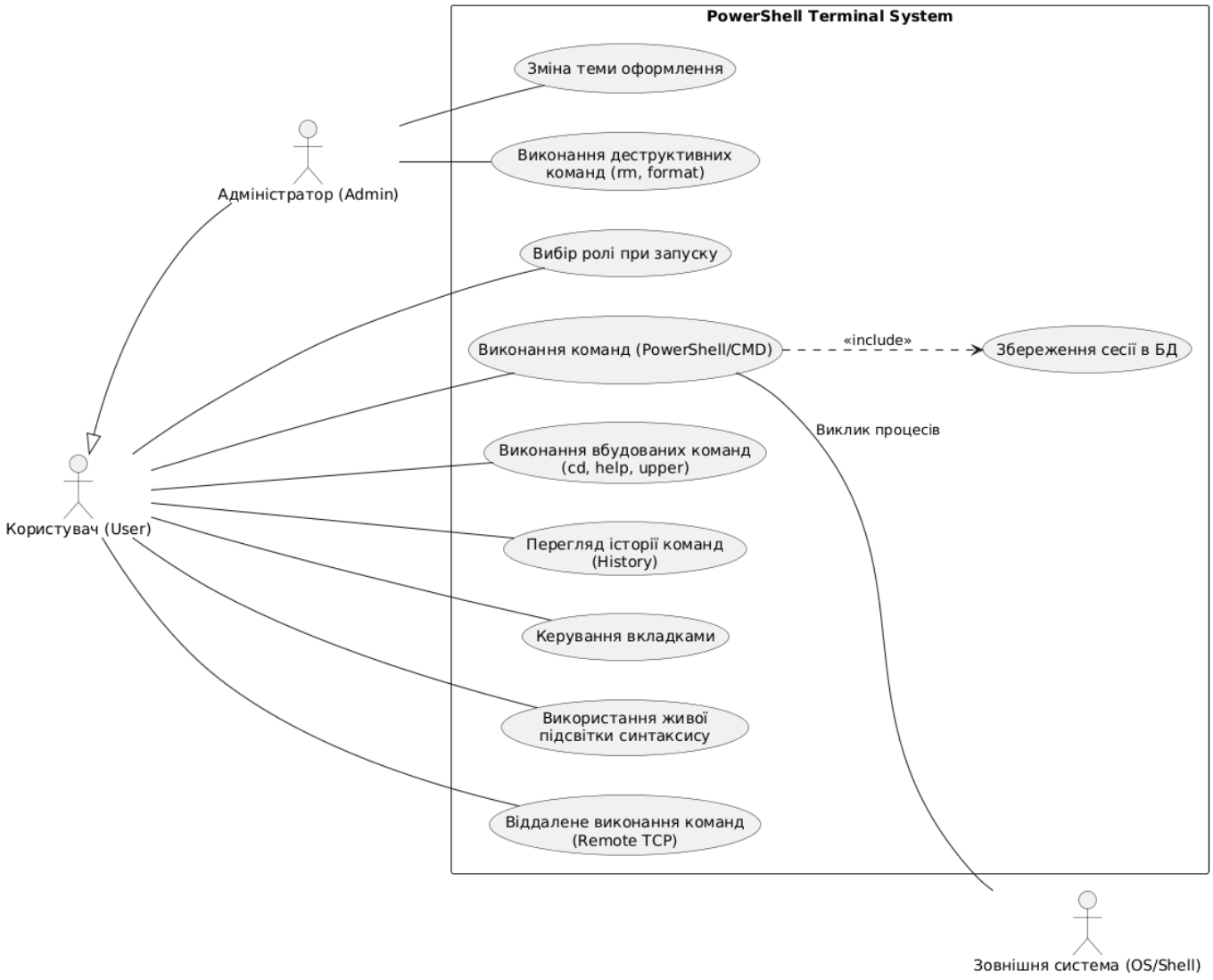

Рис. 1.1 - Діаграма варіантів використання

Табл. 1.1. Функціональні вимоги

№	Актор	Назва вимоги	Короткий опис
1	Користувач	Вибір ролі	Можливість при запуску програми обрати контекст роботи ("User" або "Admin"), що визначає доступний функціонал та інтерфейс (Abstract Factory).

2	Користувач	Введення команд	Текстове введення команд у консоль із підтримкою "живої" підсвітки синтаксису (ключові слова виділяються кольором у реальному часі).
3	Користувач	Навігація (cd)	Зміна поточної робочої директорії за допомогою внутрішньої команди cd. Стан директорії зберігається окремо для кожної вкладки.
4	Користувач	Виконання зовнішніх команд	Запуск системних процесів (PowerShell, CMD, executable files) через патерн Bridge, з отриманням результату (stdout/stderr).
5	Користувач	Робота з історією	Автоматичне збереження введених команд у локальну базу даних (SQLite) та навігація по історії за допомогою клавіш стрілок (Up/Down).
6	Користувач	Багатовіконність	Створення, перемикання та закриття вкладок терміналу. Кожна вкладка має ізольований процес та стан.
7	Користувач	Пайплайн команд (Pipe)	Підтримка передачі результату виконання однієї команди в іншу за допомогою символу | (наприклад, echo text | upper).
8	Користувач	Віддалене виконання	Можливість перемикання на рушій "Remote" для відправки команд на віддалений сервер через TCP-сокет.
9	Адміністратор	Зміна теми оформлення	Доступ до меню налаштувань для зміни глобальної теми інтерфейсу (Matrix, Ubuntu, Blue) через патерн Strategy.
10	Адміністратор	Виконання привілейованих команд	Можливість виконання потенційно небезпечних команд (rm, del, format), які заблоковані для звичайного користувача.

1.3.2. Нефункціональні вимоги до системи

Табл. 1.2. Нефункціональні вимоги

№	Назва	Опис
1	Продуктивність	Мінімальна затримка при введенні тексту (реакція на TextChanged для підсвітки) та швидкий старт нових процесів (PowerShell/CMD).
2	Персистентність даних	Збереження історії команд та налаштувань користувача у базі даних SQLite повинно відбуватися транзакційно, щоб уникнути втрати даних при аварійному закритті.
3	Розширюваність	Архітектура (Bridge, Strategy, Command) повинна дозволяти додавання нових рушіїв виконання (наприклад, Bash) або нових тем без модифікації існуючого коду.
4	Безпека (Role-Based)	Система повинна програмно блокувати виконання деструктивних команд для ролі "User" на рівні валідації запиту, не покладаючись лише на права ОС.
5	Юзабіліті (Usability)	Інтерфейс повинен візуально та функціонально нагадувати звичні термінали (моноширинний шрифт, чорний фон, стандартні гарячі клавіші).
6	Супроводжуваність	Код повинен бути структурований згідно з принципами SOLID та використовувати патерни GoF для полегшення розуміння та доопрацювання.
7	Сумісність	Програма повинна коректно працювати в середовищі Windows (через залежність від powershell.exe та cmd.exe) з встановленим .NET Runtime.

1.4. Сценарії використання системи

Табл. 1.3. Сценарій використання «Виконання команди»

Характеристика	Опис
Назва	Виконання PowerShell команди
Передумови	Термінал запущено, курсор знаходиться в рядку вводу.
Постумови	Команда виконана, результат виведено на екран, команду збережено в історії.
Взаємодіючі сторони	Користувач, Інтерпретатор команд.
Основний потік подій	1. Користувач вводить команду (наприклад, Get-Process). 2. Користувач натискає Enter. 3. Система парсить текст (Interpreter). 4. Система створює об'єкт команди (Command). 5. Система виконує команду і повертає текст результату. 6. Система відображає результат.
Винятки	Команду не знайдено або помилка синтаксису. Система виводить повідомлення про помилку червоним кольором.

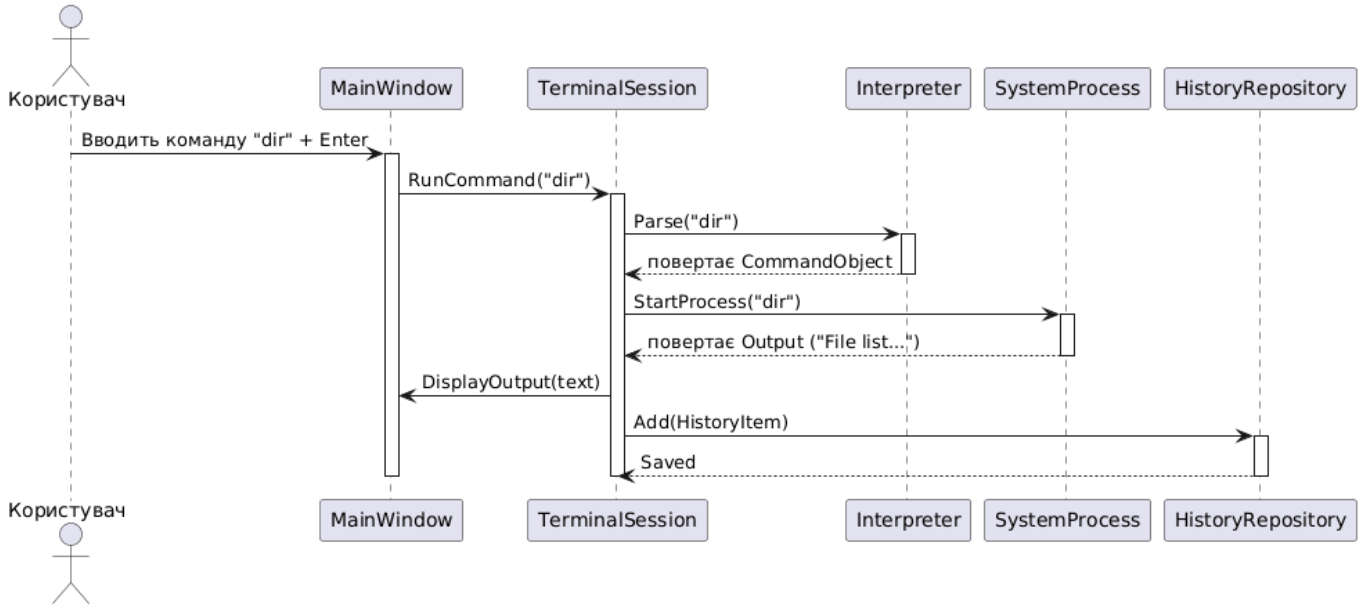

Рис. 1.2 - Діаграма послідовності «Виконання команди»

Табл. 1.4. Сценарій використання «Налаштування теми»

Характеристика	Опис
Назва	Зміна колірної схеми терміналу
Передумови	Термінал запущено, відкрито меню налаштувань.
Постумови	Інтерфейс терміналу змінив кольори фону та тексту. Конфігурація збережена в БД/файлі.
Взаємодіючі сторони	Користувач, Менеджер конфігурацій.
Основний потік подій	1. Користувач обирає опцію "Налаштування". 2. Користувач обирає нову тему зі списку (наприклад, "Dark Matrix"). 3. Система застосовує параметри теми (колір фону, шрифт) до активного вікна. 4. Система зберігає вибір у профіль користувача.
Винятки	Файл конфігурації пошкоджено. Завантажується тема за замовчуванням.

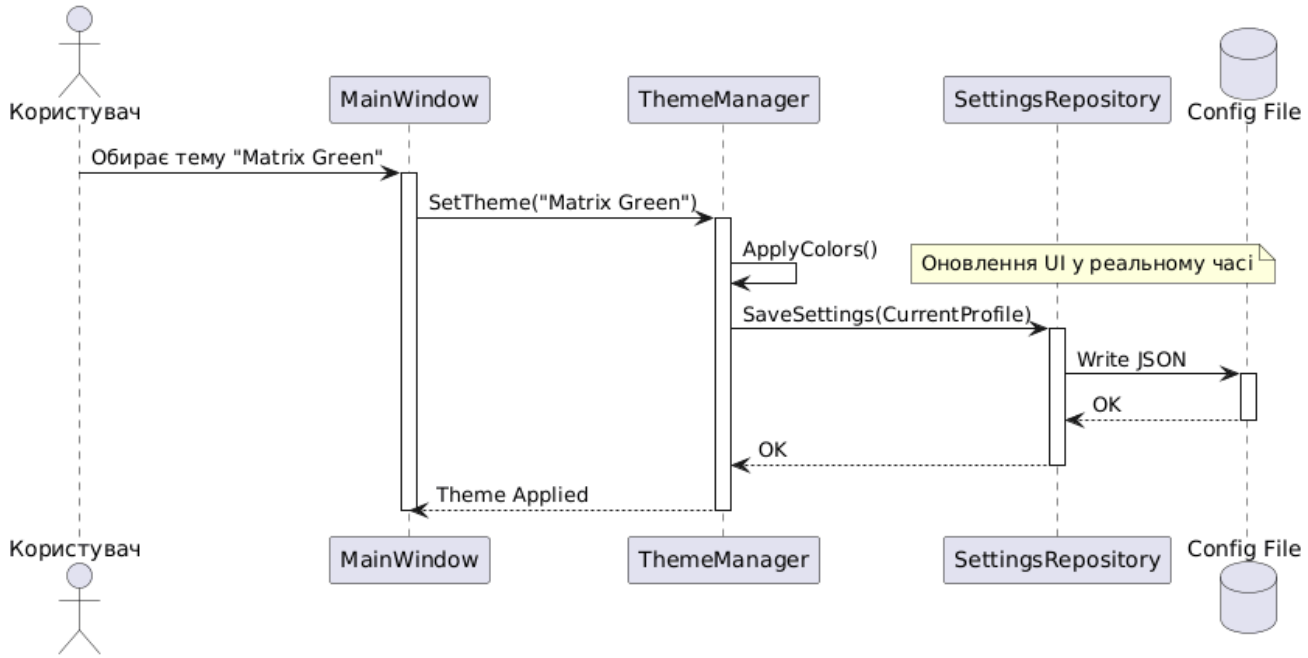

Рис. 1.3 - Діаграма послідовності «Налаштування теми»

Табл. 1.5. Сценарій використання «Робота з вкладками»

Характеристика	Опис
Назва	Відкриття нової вкладки
Передумови	Запущено хоча б одне вікно терміналу.
Постумови	Створено нову незалежну сесію PowerShell в новій вкладці.
Взаємодіючі сторони	Користувач, Менеджер вікон.
Основний потік	1. Користувач натискає кнопку "+" або комбінацію клавіш (Ctrl+T). 2. Система ініціалізує нову сесію (Abstract Factory). 3. Система додає вкладку в панель вкладок. 4. Фокус перемикається на нову вкладку.
Винятки	Досягнуто ліміту пам'яті. Система видає попередження.

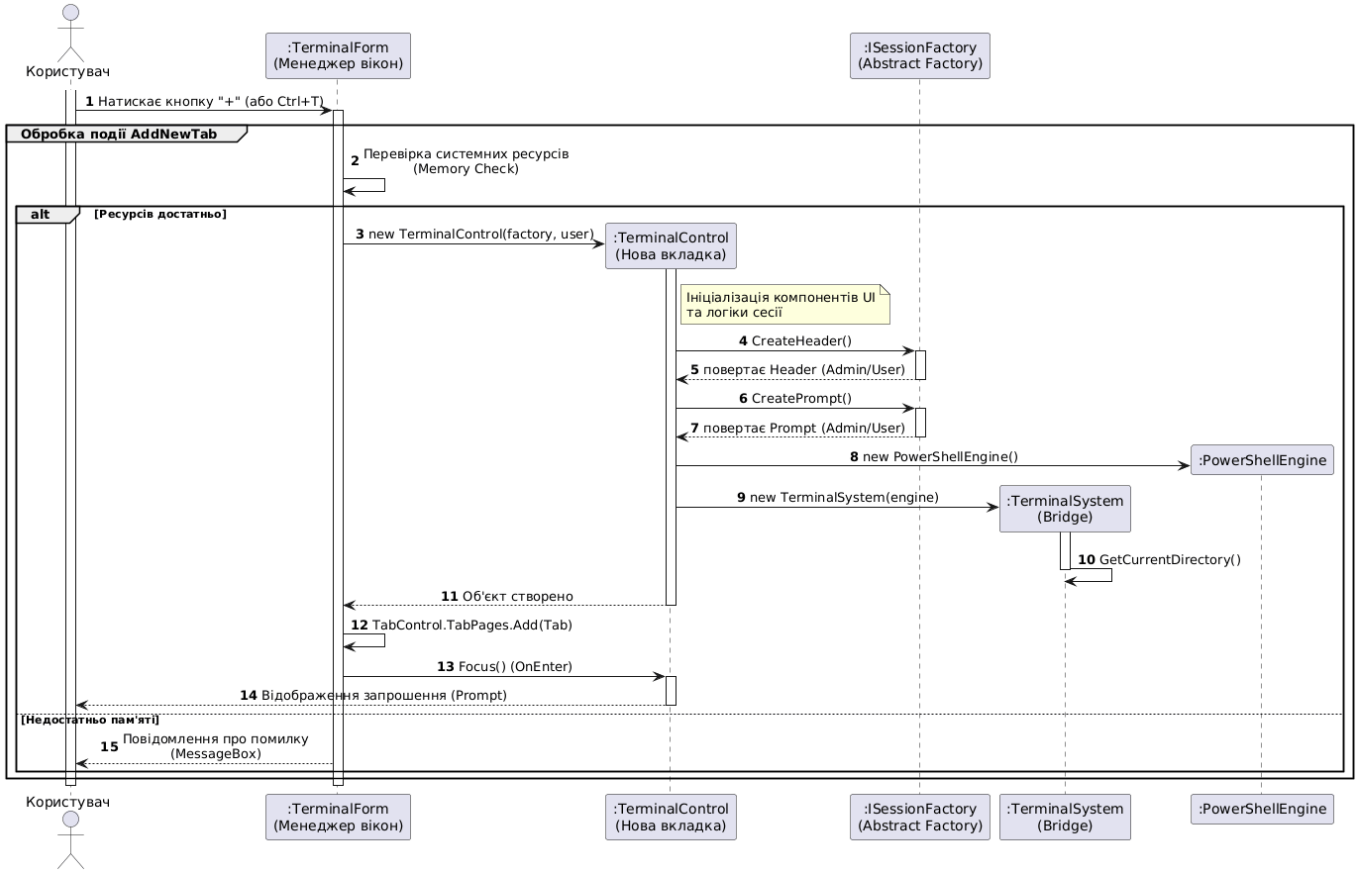

Рис. 1.4 - Діаграма послідовності «Робота з вкладками»

1.5. Концептуальна модель системи

Концептуальна модель відображає статичну структуру програмної системи, визначаючи ключові класи, їх атрибути та відношення між ними. Для повноти картини модель розбита на три рівні: рівень даних (сутності), рівень бізнес-логіки (архітектурні зв'язки) та рівень доступу до даних (репозиторії).

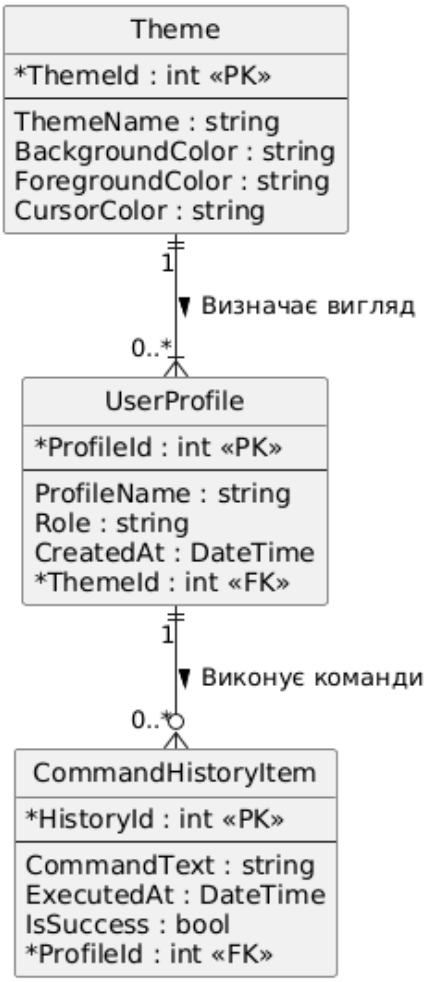

Рис. 1.5 - Діаграма сутностей даних

На цьому рівні (рис. 1.5) зображено класи, що відображають структуру бази даних (Code-First Entities). Ці класи використовуються Entity Framework Core для створення таблиць у SQLite.

- UserProfile: Центральна сутність, що зберігає інформацію про користувача та його роль (Admin/User).

- Theme: Довідкова сутність, що містить параметри кольорових схем. Зв'язок "один-до-багатьох" показує, що одну тему можуть використовувати декілька користувачів.
- CommandHistoryItem: Журнал подій. Зв'язок з UserProfile забезпечує можливість фільтрації історії для конкретного користувача.

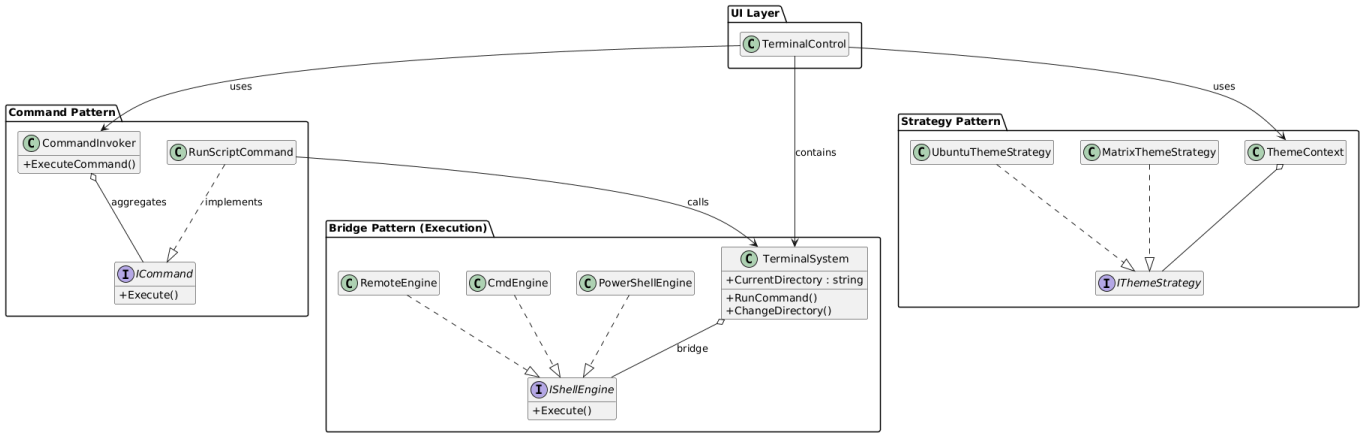

Рис. 1.6 - Діаграма архітектурних зв'язків

Це ключова діаграма (рис. 1.6), яка демонструє взаємодію основних компонентів системи через патерни проєктування GoF. Вона показує, як інтерфейс (UI) відокремлений від логіки виконання.

- Bridge: Клас TerminalSystem виступає абстракцією, що делегує виконання інтерфейсу IShellEngine. Це дозволяє змінювати рушії (PowerShell/CMD) без зміни коду терміналу.
- Command: CommandInvoker не знає деталей команди, він лише викликає метод Execute() інтерфейсу ICommand. Конкретна команда RunScriptCommand зв'язує систему виконання з текстом скрипта.
- Strategy: ThemeContext дозволяє підміняти алгоритм розфарбовування форми (IThemeStrategy) під час виконання програми.

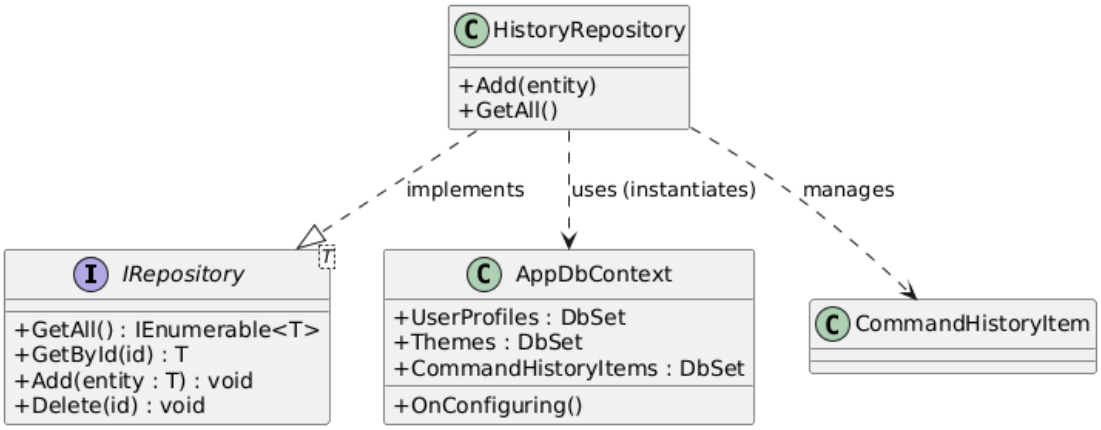

Рис. 1.7 - Діаграма класів репозиторіїв

На рис. 1.7 зображено організацію шару доступу до даних. Використання патерну Repository дозволяє абстрагувати бізнес-логіку від прямої роботи з контекстом Entity Framework.

- IRepository<T>: Узагальнений інтерфейс, що визначає контракт CRUD-операцій (Create, Read, Update, Delete).
- HistoryRepository: Конкретна реалізація для роботи з історією команд. Цей клас інкапсулює створення екземпляра AppDbContext та керування транзакціями, забезпечуючи чистоту коду у TerminalControl.
- AppDbContext: Клас контексту, що успадковується від DbContext (EF Core) та відповідає за з'єднання з файлом SQLite.

1.6. Вибір бази даних

Для забезпечення коректного функціонування системи та збереження стану між сеансами роботи необхідно забезпечити надійне зберігання даних. У межах проєкту база даних використовується для збереження ключових сутностей системи, зокрема:

- Профілі користувачів (UserProfile): зберігання налаштувань та рольової приналежності (Admin/User).
- Теми оформлення (Theme): налаштування кольорових схем інтерфейсу.
- Історія команд (CommandHistoryItem): журнал виконаних операцій з прив'язкою до часу та статусу виконання.

На початковому етапі проєктування було проведено порівняльний аналіз поширених систем управління базами даних (СУБД): PostgreSQL, MySQL, Microsoft SQL Server та SQLite.

Спочатку як основний варіант розглядалася СУБД Microsoft SQL Server, оскільки вона є стандартом для корпоративної розробки на платформі .NET і має низку переваг:

1. Нативна інтеграція: Ідеальна сумісність з екосистемою Microsoft та Entity Framework Core.
2. Інструментарій: Наявність SQL Server Management Studio (SSMS) для зручного проєктування схеми.
3. Масштабованість: Можливість обробки величезних масивів даних, що було б корисно при реалізації централізованого логування для сотень користувачів.

Однак, зважаючи на специфіку розроблюваного застосунку (Desktop-додаток "товстий клієнт"), використання клієнт-серверної СУБД виявилося архітектурно надлишковим. Вимога встановлювати та налаштовувати локальний сервер SQL для запуску простого терміналу значно ускладнила б розгортання програми (Deployment).

Тому для фінальної реалізації було прийнято рішення використати вбудовану реляційну СУБД SQLite. Це рішення обґрунтоване наступними перевагами:

- Zero-configuration та портативність: База даних зберігається у вигляді єдиного локального файлу (terminal.db), що дозволяє переносити програму простим копіюванням папки без необхідності встановлення серверного ПЗ.
- Сумісність з EF Core: SQLite повністю підтримує підхід Code-First, що дозволило автоматично генерувати структуру таблиць при першому запуску програми на основі C#-класів (Entities).
- Продуктивність: Для локального зберігання історії та налаштувань швидкість SQLite є вищою за мережеві запити до великих серверів.

Єдиним недоліком SQLite є нижчий рівень захисту від одночасного запису (concurrency) порівняно з SQL Server, проте для однокористувацького режиму роботи терміналу цей фактор не є критичним.

1.7. Вибір мови програмування та середовища розробки

Для реалізації програмної системи «PowerShell Terminal» було обрано мову програмування C# та платформу .NET 9 з використанням бібліотеки графічного інтерфейсу Windows Forms.

Мова програмування C# є сучасною, об’єктно-орієнтованою мовою зі строгою типізацією. Для даного проєкту C# стала безальтернативним вибором завдяки кільком ключовим факторам:

1. Робота з системними процесами: Потужний простір імен System.Diagnostics дозволяє ефективно керувати зовнішніми процесами (powershell.exe, cmd.exe), перехоплювати їхні потоки введення/виведення та керувати їхнім життєвим циклом, що є основою архітектури проєкту (патерн Bridge).
2. Реалізація патернів: C# має розвинені механізми ООП (інтерфейси, абстрактні класи, поліморфізм), що дозволило чисто та академічно правильно реалізувати складні архітектурні патерни, такі як Abstract Factory, Strategy та Interpreter.
3. Подійно-орієнтована модель: Зручна робота з подіями (events), що критично важливо для реалізації "живої" підсвітки синтаксису та обробки користувацького вводу в реальному часі.

В якості графічного фреймворку було обрано Windows Forms (у складі .NET 9). Його ключові переваги для цього проєкту:

- Нативні компоненти: Доступ до компонента RichTextBox, який забезпечує необхідні можливості для форматування тексту та зміни кольорів окремих слів (Syntax Highlighting).
- Легковаговість: WinForms забезпечує швидкий старт програми та менше споживання оперативної пам'яті порівняно з WPF або MAUI, що важливо для утиліти, яка має працювати у фоновому режимі.
- Простота розгортання: Додатки WinForms легко компілюються в один виконуваний файл, що спрощує перенесення програми між комп'ютерами.

Для роботи з даними використано Entity Framework Core. Це дозволило застосувати підхід Code-First, автоматично генеруючи структуру бази даних SQLite

на основі класів C#, що значно пришвидшило розробку модуля історії команд та профілів.

Як середовище розробки було обрано Visual Studio Code (VS Code). Це легкий, швидкий та кросплатформений редактор коду. Завдяки офіційному розширенню C# Dev Kit, він надає всі необхідні можливості для повноцінної розробки, включаючи IntelliSense, налагодження (Debugging) та рефакторинг. Важливою перевагою є наявність інтегрованого терміналу, який активно використовувався для керування пакетами NuGet та роботи з системою контролю версій Git.

1.8. Проєктування розгортання системи

Для повного розуміння архітектури системи необхідно розглянути її з двох точок зору: логічної та фізичної. Логічна архітектура визначає, з яких програмних частин складається додаток, тоді як фізична архітектура показує, як вони розміщуються на реальному обладнанні.

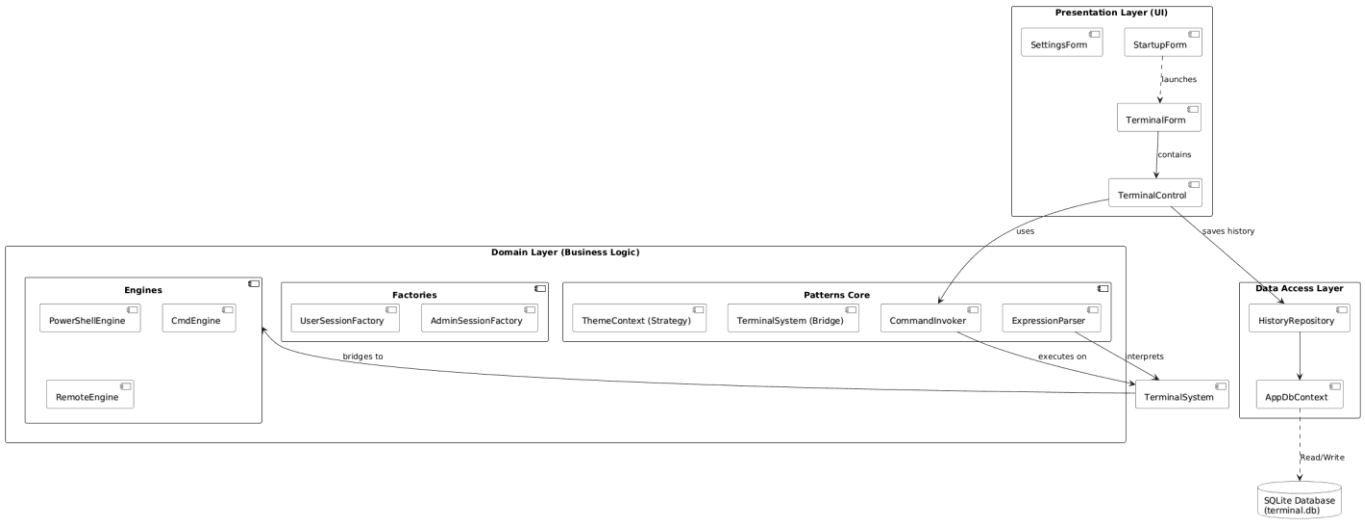

Рис. 1.8 - Діаграма компонентів

На діаграмі компонентів (рис. 1.8) зображено логічну архітектуру системи, поділену на три основні шари:

Шар представлення (Presentation Layer): Містить компоненти інтерфейсу користувача (TerminalForm, SettingsForm), реалізовані на Windows Forms. Цей шар

відповідає за відображення даних, обробку введення користувача (включно з "живою" підсвіткою синтаксису) та керування вкладками.

Шар бізнес-логіки (Domain Layer): Це ядро системи, де реалізовано основні архітектурні патерни. Тут розташовані:

- Interpreter (ExpressionParser): Розбирає текстові команди та пайплайни.
- Bridge (TerminalSystem): Абстрагує виконання команд від конкретного рушія.
- Strategy & Factory: Відповідають за динамічне створення інтерфейсу та зміну тем.

Шар доступу до даних (Data Access Layer): Містить контекст бази даних (AppDbContext) та репозиторії (HistoryRepository). Цей шар інкапсулює роботу з ORM Entity Framework Core та ізолює бізнес-логіку від деталей збереження даних.

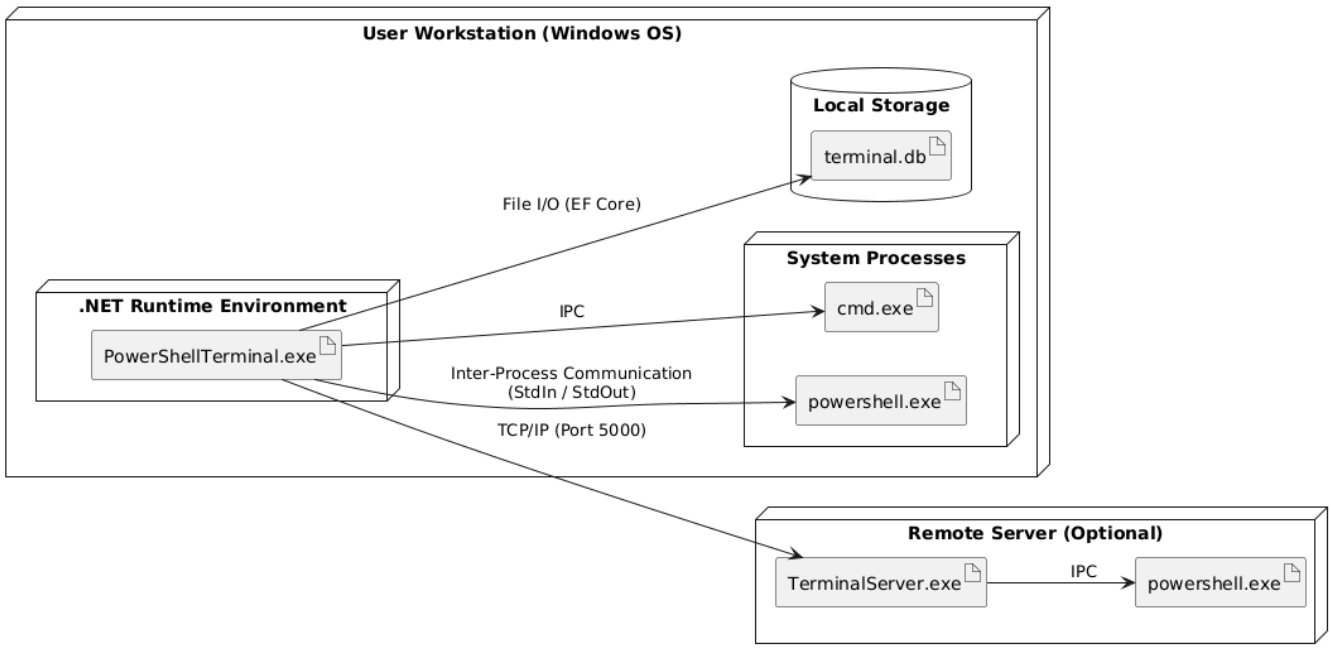

Рис. 1.9 - Діаграма розгортання

На діаграмі розгортання (рис. 1.9) зображено фізичну архітектуру настільного застосунку "Powershell terminal". Система включає такі вузли:

Робоча станція користувача (User Workstation): Персональний комп'ютер під керуванням ОС Windows із встановленим середовищем .NET Runtime. Тут виконується основний файл додатка PowerShellTerminal.exe.

- Взаємодія з ОС: Додаток запускає дочірні процеси (powershell.exe, cmd.exe) і спілкується з ними через перенаправлення потоків введення/виведення (IPC).
- Збереження даних: База даних terminal.db (SQLite) фізично розміщується в тій самій директорії, що й виконуваний файл, забезпечуючи принцип "Zero-configuration".

Віддалений сервер (Remote Server Node): Опціональний вузол, який може бути розгорнутий на іншій машині в мережі. На ньому працює слухач TerminalServer.exe.

- Протокол взаємодії: Клієнтський додаток з'єднується із сервером через протокол TCP/IP (порт 5000) для передачі текстових команд та отримання результату виконання.

2 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

2.1. Структура бази даних

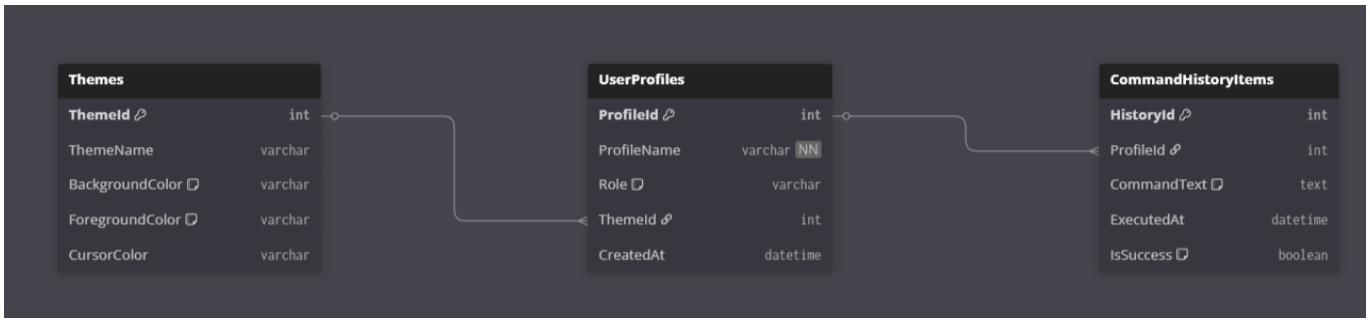

Рис. 2.1 – Проектування бази даних

Табл. 2.1 . Огляд сутностей

Назва таблиці	Атрибут (Поле)	Тип даних (SQLite)	Тип ключа	Опис призначення
Themes				Зберігає налаштування візуального оформлення терміналу.
	ThemeId	INTEGER	PK	Первинний ключ. Ідентифікатор теми.
	ThemeName	TEXT		Зрозуміла назва теми для відображення у випадаючому списку (ComboBox).
	BackgroundColor	TEXT		Колір фону форми та полів введення у

				форматі HEX (наприклад, #000000).
	ForegroundColor	TEXT		Колір основного тексту шрифту (наприклад, #00FF00 для Matrix).
	CursorColor	TEXT		Колір текстового курсору та виділення.
UserProfiles				Зберігає облікові записи та налаштування користувачів.
	ProfileId	INTEGER	PK	Первинний ключ. Унікальний ID користувача.
	ProfileName	TEXT		Системне ім'я користувача (використовується для відображення в Prompt).
	Role	TEXT		Роль користувача (Admin / User). Визначає права доступу та вигляд інтерфейсу (Abstract Factory).

	ThemeId	INTEGER	FK	Зовнішній ключ. Посилання на таблицю Themes. Зберігає вибір теми користувачем.
	CreatedAt	TEXT (ISO8601)		Дата та час першого створення профілю.
CommandHistoryItems				Журнал (лог) усіх виконаних команд.
	HistoryId	INTEGER	PK	Первинний ключ запису.
	ProfileId	INTEGER	FK	Зовнішній ключ. Посилання на таблицю UserProfiles. Вказує, хто виконав команду.
	CommandText	TEXT		Повний текст команди, яку ввів користувач (включно з аргументами).
	ExecutedAt	TEXT (ISO8601)		Часова мітка моменту виконання команди (для сортування історії).
	IsSuccess	INTEGER (0/1)		Прапорець статусу. 1 (True) — команда

				виконана успішно, 0 (False) — сталася помилка.

2.2. Архітектура системи

2.2.1. Специфікація системи

Розроблена система "PowerShell Terminal" є настільним програмним комплексом (Desktop Application), побудованим на базі платформи .NET (версія 9.0). Архітектура системи спроєктована за принципом багатошарової архітектури (N-Tier Architecture) з інтеграцією класичних патернів проєктування GoF. Такий підхід забезпечує слабку зв'язність (Loose Coupling) між модулями інтерфейсу, бізнес-логіки та доступу до даних, що робить систему гнучкою до розширення та легкою у супроводі.

Табл. 2.2. Технологічний стек

Категорія	Технологія / Інструмент	Опис використання в проєкті
Платформа	.NET 9	Середовище виконання (Runtime), що забезпечує кросплатформеність (в перспективі), високу продуктивність та доступ до новітніх можливостей мови C#.
Мова програмування	C# 12	Основна мова розробки. Використано сучасні можливості: record types, pattern matching, асинхронність (async/await) та події.
Графічний інтерфейс	Windows Forms (WinForms)	Фреймворк для побудови UI. Використано для створення головного вікна, системи вкладок (TabControl) та кастомізації RichTextBox для консольного виводу.

Доступ до даних (ORM)	Entity Framework Core	Об'єктно-реляційний мапер для роботи з базою даних. Реалізує підхід Code-First (генерація БД з класів C#).
База даних	SQLite	Вбудована реляційна СУБД. Зберігає історію команд, профілі користувачів та налаштування тем у локальному файлі terminal.db.
Взаємодія з ОС	System.Diagnostics.Process	API для запуску зовнішніх процесів (powershell.exe, cmd.exe), керування ними та перехоплення потоків введення/виведення (StdIn/StdOut/StdErr).
Обробка тексту	System.Text.RegularExpressions	Використовується для лексичного аналізу команд (Interpreter) та реалізації "живої" підсвітки синтаксису (Syntax Highlighting) в інтерфейсі.
Мережева взаємодія	System.Net.Sockets	Використання протоколу TCP/IP (класи TcpClient, TcpListener) для реалізації віддаленого виконання команд (Remote Engine).
Архітектура	GoF Patterns	Реалізація патернів: Bridge (для рушіїв), Strategy (для тем), Command (для дій), Abstract Factory (для ролей), Interpreter (для парсингу).

Структура рішення

Структура проєкту організована відповідно до принципів чистої архітектури, де бізнес-логіка відокремлена від інтерфейсу користувача та деталей реалізації бази даних. Рішення складається з одного основного проєкту PowerShellTerminal.App, який логічно розділений на папки (Namespaces).

PowerShellTerminal.App.Data Відповідає за персистентність даних. Тут реалізовано клас AppDbContext, який успадковує DbContext від Entity Framework Core і налаштовує з'єднання з файлом SQLite. HistoryRepository інкапсулює логіку запису та зчитування історії команд, забезпечуючи чистоту коду в контролерах.

PowerShellTerminal.App.Domain Центральна частина системи, що містить реалізацію всіх патернів проєктування:

- Entities: Містить класи-моделі (User, Theme), які відображаються на таблиці бази даних.
- AbstractFactory: Відповідає за створення візуальних компонентів (заголовок вікна, рядок запрошення) залежно від ролі користувача (Admin/User), не прив'язуючи основний код до конкретних класів.
- Bridge: Відокремлює абстракцію терміналу (яка пам'ятає поточну папку cd) від реалізації виконання (PowerShell, CMD, Remote). Це дозволяє змінювати "двигун" терміналу "на льоту".
- Interpreter: Містить логіку розбору введеного тексту. Визначає, чи є команда внутрішньою (history, help), чи її треба передати в ОС, а також обробляє пайплайни (|).
- Strategy: Визначає алгоритми зміни кольорової схеми інтерфейсу.

PowerShellTerminal.App.UI Містить графічний інтерфейс користувача (Windows Forms).

- Controls\TerminalControl: Ключовий компонент, що інкапсулює логіку однієї вкладки терміналу. Саме тут реалізовано обробку подій клавіатури та алгоритм "живої" підсвітки синтаксису (Regex Highlighting).
- Forms: Містить форми для авторизації (StartupForm), налаштувань та головний контейнер вкладок.

Program.cs Точка входу в додаток. Тут відбувається ініціалізація бази даних (створення таблиць, якщо вони відсутні), початкове наповнення даними (Seed Data) та запуск форми вибору ролі.

2.2.2. Вибір та обґрунтування патернів реалізації

А) Проблема: Система повинна підтримувати роботу з різними рушіями виконання команд (PowerShell, CMD, Remote TCP), але при цьому логіка самого терміналу (збереження поточної директорії cd, робота з вкладками) повинна залишатися незмінною. Жорстка прив'язка коду UI до класу Process унеможливлює легку заміну рушія.

Рішення: Патерн Bridge дозволив відокремити абстракцію (керування станом сесії) від її реалізації (фізичного виконання процесу). Структура наведена на рис. 2.2.

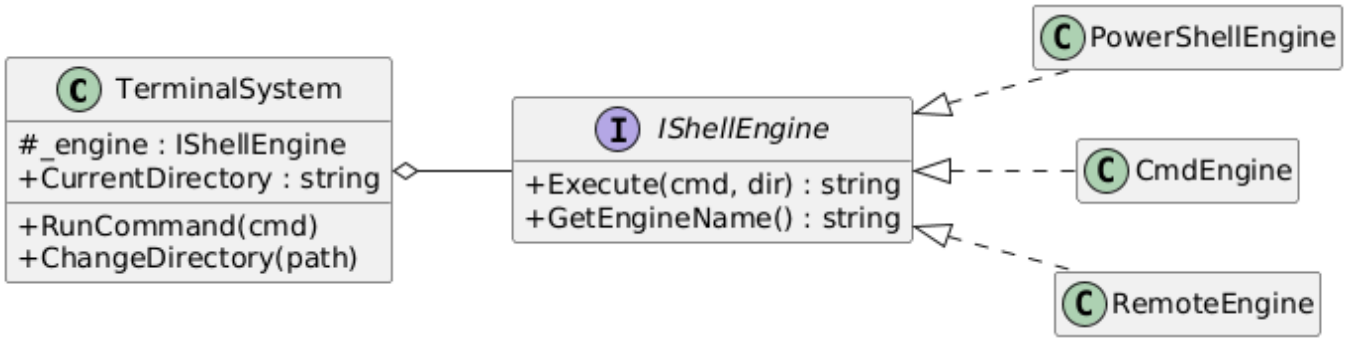

Рис. 2.2 - Структура патерну Bridge

- TerminalSystem (Абстракція): Відповідає за високорівневу логіку (навігація, збереження контексту).
- IShellEngine (Реалізатор): Інтерфейс, що визначає контракт виконання.
- PowerShellEngine / CmdEngine: Конкретні реалізації, що працюють із системними процесами.
- Перевага: Можна додати підтримку Bash (WSL) або SSH, створивши новий клас двигуна, не змінюючи код TerminalSystem.

Б) Проблема: Інтерфейс терміналу повинен підтримувати зміну тем оформлення (кольори фону, тексту, курсору) "на льоту". Використання умовних операторів (if-else

або switch) у коді форми призводить до заплутаності та порушує принцип відкритості/закритості (OCP).

Рішення: Використано патерн Strategy для винесення параметрів оформлення в окремі класи. Структура наведена на рис. 2.3.

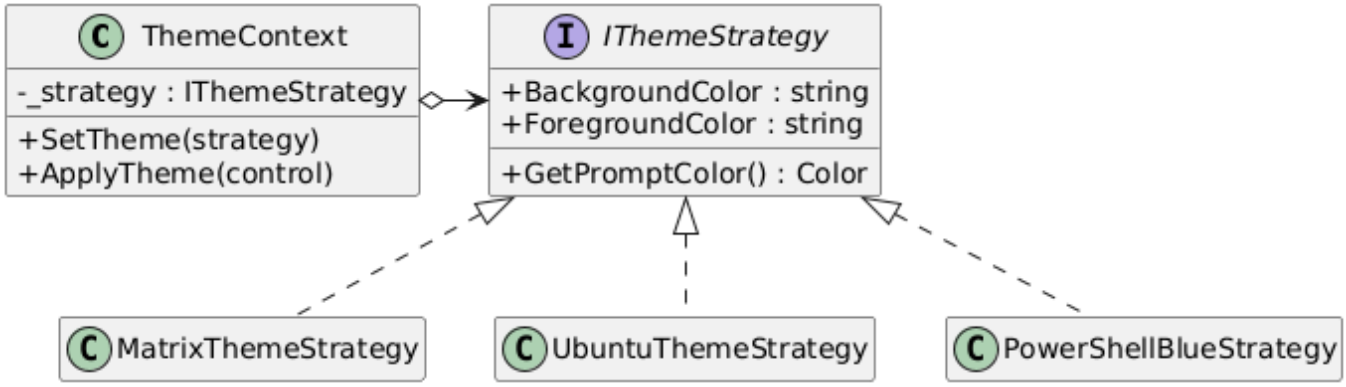

Рис. 2.3 - Структура патерну Strategy

- IThemeStrategy: Інтерфейс, що описує набір кольорів.
- ThemeContext: Клас у формі налаштувань, який застосовує обрану стратегію до елементів UI.
- Перевага: Додавання нової теми зводиться до створення нового класу без втручання в логіку форми.

В) Проблема: Необхідність відокремити об'єкт, що ініціює дію (кнопка Enter на клавіатурі), від об'єкта, що виконує дію (TerminalSystem). Також потрібна можливість логування операцій до їх виконання.

Рішення: Запит на виконання скрипта інкапсульовано в об'єкт RunScriptCommand (рис. 2.4).

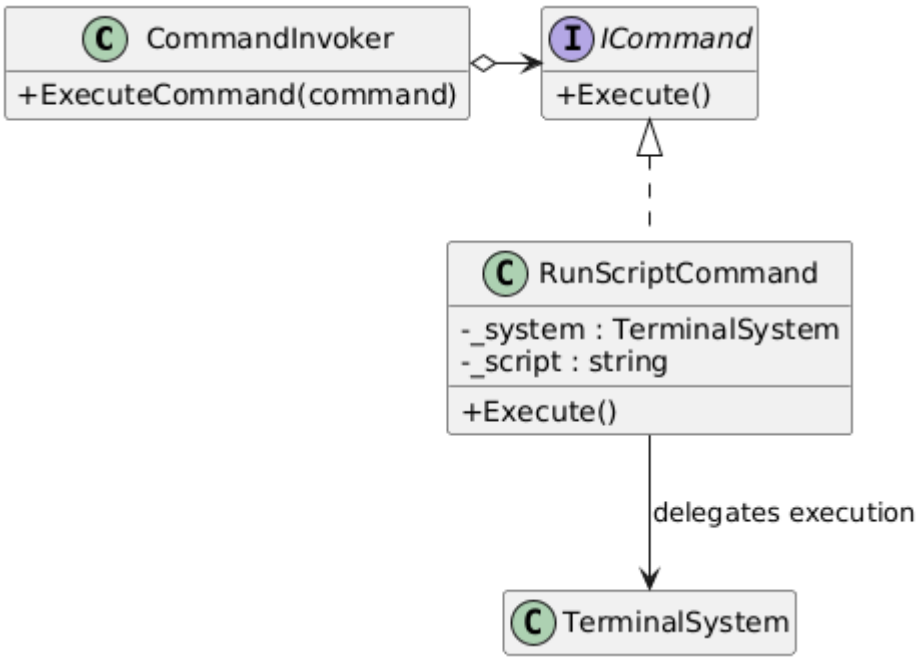

Рис. 2.4 - Структура патерну Command

Це дозволило реалізувати механізм "Proxy" (перевірка безпеки команд rm/format для користувача) всередині методу Execute() до того, як команда потрапить у двигун.

Г) Проблема: Система має два режими роботи: "Admin" та "User". Кожен режим вимагає створення узгодженого набору візуальних елементів (заголовок вікна, стиль запрошення prompt). Ручне створення цих об'єктів у коді призводить до ризику змішування стилів.

Рішення: Створено фабрики, які гарантують створення правильних сімейств об'єктів для кожної ролі. Структура наведена на рис. 2.5.

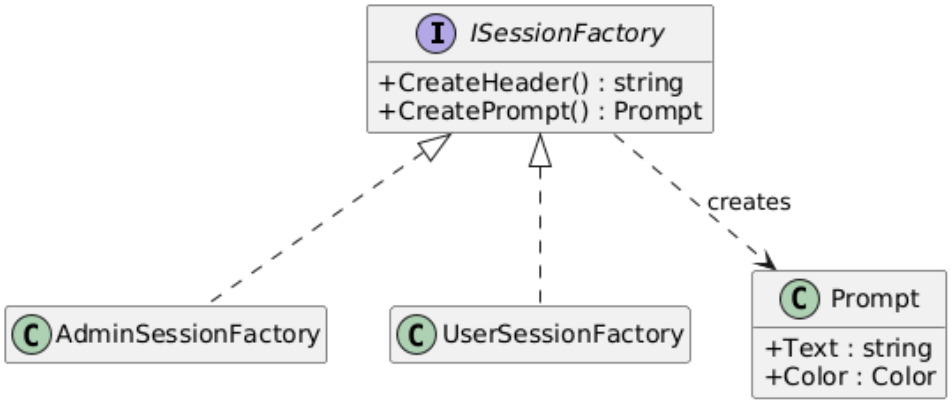

Рис. 2.5 - Структура патерну Abstract Factory

Гарантує, що Адміністратор завжди отримає червоний промпт і попереджувальний заголовок, а Користувач — зелений промпт.

Д) Проблема: Термінал отримує від користувача "сирий" рядок тексту. Необхідно розрізняти системні команди (dir, ipconfig), внутрішні команди програми (history, help, cd) та операції пайплайну (| upper).

Рішення: Створено граматику для розбору вхідного рядка та перетворення його на об'єкти-вирази (Expressions). Структура наведена на рис. 2.6.

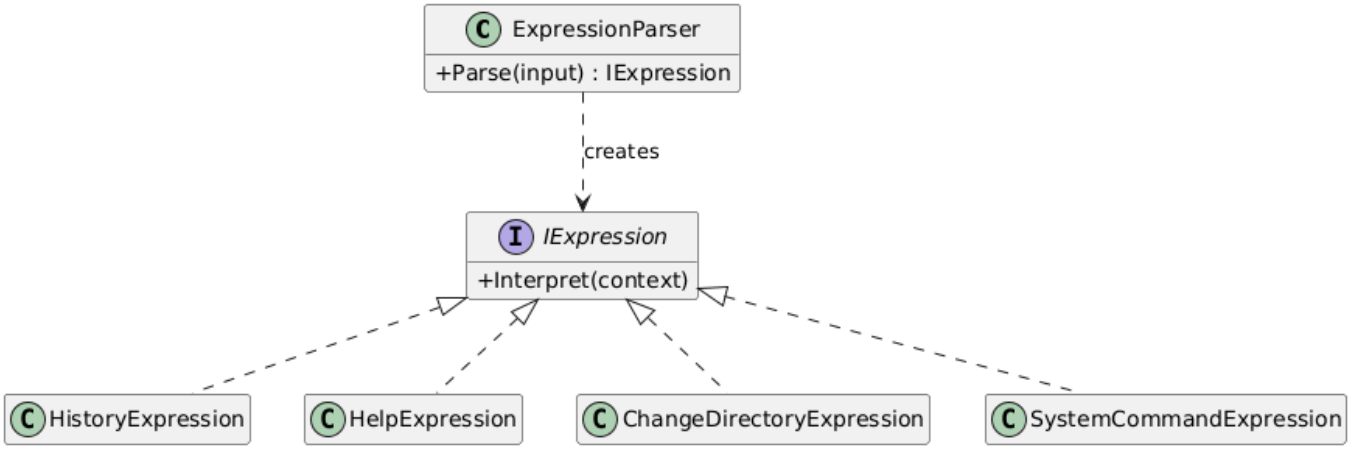

Рис. 2.6 - Структура патерну Interpreter

- ExpressionParser: Аналізує рядок. Якщо бачить cd, створює ChangeDirectoryExpression.
- IExpression: Визначає метод Interpret.

- Перевага: Дозволяє легко додавати нові ключові слова мови терміналу (наприклад, alias або clear), просто додаючи нові класи виразів.

2.3. Інструкція користувача

2.3.1. Запуск та вхід у систему

1. Запустіть файл PowerShellTerminal.exe.
2. При першому запуску система автоматично створить файл бази даних terminal.db та налаштує необхідні таблиці.
3. На екрані з'явиться вікно вибору режиму роботи . Оберіть необхідну роль:
    - Запуск як ADMIN (Червона кнопка): Надає повний доступ до налаштувань тем та дозволяє виконувати всі системні команди.
    - Запуск як USER (Зелена кнопка): Стандартний режим. Доступ до налаштувань заблоковано, небезпечні команди (видалення файлів, форматування) заборонені.

2.3.2. Інтерфейс головного вікна

Після вибору ролі відкривається головне вікно терміналу.

- Панель вкладок (вгорі): Відображає активні сесії. Кнопка "+" створює нову вкладку.
- Робоча область (центр): Чорне поле для виведення результатів.
- Рядок введення (внизу): Поле для набору команд. Підтримує "живу" підсвітку синтаксису (ключові слова, такі як dir, echo, if, автоматично підсвічуються золотим або синім кольором).

2.3.3. Робота з командами

Система підтримує три типи команд:

1. Вбудовані команди (Internal):
    - help — вивести довідку про доступні функції та гарячі клавіші.
    - history — показати повний журнал команд, збережений у базі даних.

- cd <шлях> — змінити поточну директорію (наприклад, cd Desktop або cd ..).
- exit — закрити поточну вкладку або програму.

2. Системні команди (System): Працюють усі стандартні команди Windows PowerShell та CMD:
    - dir / ls — перегляд файлів.
    - ipconfig — налаштування мережі.
    - ping <адреса> — перевірка зв'язку.
    - Запуск програм: notepad, calc, git status.
3. Команди конвеєра (Piping): Ви можете використовувати символ | для передачі тексту.
    - echo привіт | upper — виведе "ПРИВІТ" (конвертація у верхній регістр).

2.3.4. Керування вкладками та навігація

- Нова вкладка: Натисніть кнопку "+" у верхній частині вікна або комбінацію клавіш Ctrl+T. Кожна вкладка працює ізольовано: ви можете відкрити папку C:\Windows в одній вкладці та D:\Projects в іншій.
- Історія команд: Використовуйте клавіші ВГОРУ (↑) та ВНИЗ (↓) на клавіатурі для швидкого перебору раніше введених команд.

2.3.5. Налаштування оформлення (Themes)

Доступ до налаштувань регулюється правами доступу:

- Для Адміністратора: Натисніть кнопку "Settings" у правому верхньому куті. У вікні, що з'явиться, виберіть одну з тем:
    - Matrix (Чорний фон, яскраво-зелений текст).
    - Ubuntu (Фіолетовий фон, помаранчевий текст).
    - PowerShell Blue (Класичний синій фон). Після натискання "Apply" тема застосується миттєво до всіх елементів.

- Для Користувача: Кнопка налаштувань відображається як "locked" і є неактивною. Користувач працює у темі, встановленій за замовчуванням.

2.3.6. Обмеження безпеки (для режиму User)

Якщо ви увійшли як звичайний користувач, система автоматично блокує виконання деструктивних команд. При спробі ввести rm, del або format, термінал видасть повідомлення: [SECURITY ALERT] Command is restricted to Administrators only. Для виконання цих дій необхідно перезапустити програму в режимі Admin.

2.3.7. Можливі несправності

Табл. 2.3. Можливі несправності

Симптом	Можлива причина	Рішення
Команда cd не змінює папку	Введено неіснуючий шлях.	Перевірте правильність шляху командою dir.
Не працює git, python	Ці програми не встановлені в ОС.	Термінал використовує змінні середовища Windows. Встановіть необхідне ПЗ.
Помилка "Database locked"	Файл terminal.db відкритий іншою програмою.	Закрийте інші екземпляри терміналу або програми DB Browser.

ВИСНОВКИ

У ході виконання курсового проєкту було успішно спроєктовано та реалізовано програмну систему "PowerShell Terminal", яка є повнофункціональним багатоконтекстним емулятором командного рядка з розширеними можливостями налаштування інтерфейсу та адміністрування. Основною метою роботи було створення гнучкого інструменту, що поєднує потужність стандартних системних оболонок Windows із зручністю сучасного графічного інтерфейсу, і цю мету було повністю досягнуто завдяки використанню платформи .NET 9 та технології Windows Forms. Ключовим результатом проєкту стала побудова надійної та гнучкої архітектури системи на основі класичних патернів проєктування, що забезпечило додатку високу масштабованість, тестованість та легкість у подальшому супроводі. Зокрема, застосування патерну Bridge дозволило абстрагувати логіку роботи терміналу від конкретних рушіїв виконання, надаючи можливість безболісного перемикання між PowerShell, CMD та віддаленим сервером, патерн Strategy забезпечив механізм динамічної зміни візуального оформлення інтерфейсу, а

використання Abstract Factory дозволило реалізувати гнучку рольову модель безпеки з адаптацією функціоналу для адміністраторів та звичайних користувачів.

Важливим технічним досягненням стала інтеграція системи з локальною реляційною базою даних SQLite через ORM Entity Framework Core, що дозволило реалізувати надійне збереження історії команд, профілів користувачів та налаштувань між сеансами роботи без необхідності розгортання складного серверного оточення. Окрім архітектурних рішень, у проєкті було вирішено низку складних прикладних задач, таких як реалізація власного лексичного інтерпретатора для обробки внутрішніх команд і пайплайнів, синхронізація стану файлової системи між ізольованими вкладками та створення механізму "живої" підсвітки синтаксису в реальному часі. Практична цінність роботи полягає у створенні готового до використання інструменту, який не лише виконує функції стандартної консолі, але й пропонує покращений користувацький досвід та механізми захисту від деструктивних дій. Виконання даного курсового проєкту дозволило систематизувати знання з об'єктно-орієнтованого проєктування, поглибити навички роботи з системними процесами та базами даних, а також продемонструвати ефективність використання патернів проєктування для вирішення реальних інженерних задач.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Visual Studio Code Documentation. Documentation for Visual Studio Code [Електронний ресурс]. – Режим доступу: https://code.visualstudio.com/docs.
2. Microsoft .NET Documentation. .NET 9.0 and C# 12 Documentation [Електронний ресурс] // Microsoft Learn. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/.
3. Windows Forms Documentation. Desktop Guide (Windows Forms .NET) [Електронний ресурс] // Microsoft Learn. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/.
4. Entity Framework Core Documentation. Documentation for Entity Framework Core [Електронний ресурс] // Microsoft Learn. – Режим доступу: https://learn.microsoft.com/en-us/ef/core/.
5. SQLite Documentation. Official Documentation for SQLite [Електронний ресурс]. – Режим доступу: https://www.sqlite.org/docs.html.
6. Payette B., Siddaway R. Windows PowerShell in Action. 3rd Edition. – Manning Publications, 2018. – 897 p. [Електронний ресурс]. – Режим доступу: https://networkvt.com/wp-content/uploads/2018/06/Windows-PowerShell-in-Action-2018.pdf .
7. Troelsen A., Japikse P. C# 7.0 and .NET Core 2.0 – with Visual Studio 2017. – Apress, 2017. – 1372 p. [Електронний ресурс]. – Режим доступу: https://dl.ebooksworld.ir/motoman/Apress.Pro.Csharp.7.With.NET.and.NET.Core.www.EBooksWorld.ir.pdf .
8. Refactoring.Guru. Design Patterns [Електронний ресурс]. – Режим доступу: https://refactoring.guru/uk/design-patterns .
9. System.Diagnostics.Process Class. Process Class Documentation [Електронний ресурс] // Microsoft Learn. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.process.

10. Regular Expressions in .NET. .NET Regular Expressions Documentation [Електронний ресурс] // Microsoft Learn. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/standard/base-types/regular-expressions.
11. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Addison-Wesley Professional, 1994. – 395 p. [Електронний ресурс]. – Режим доступу: https://www.javier8a.com/itc/bd1/articulo.pdf .
12. PowerShell Documentation. Official PowerShell Documentation [Електронний ресурс] // Microsoft Learn. – Режим доступу: https://learn.microsoft.com/en-us/powershell/.
13. System.Management.Automation Namespace. PowerShell SDK Documentation [Електронний ресурс] // Microsoft Learn. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/api/system.management.automation.

ДОДАТКИ

Додаток А

Доступ до проєкту

Проєкт завантажено на github-репозиторій, доступний за наступним посиланням: https://github.com/Hoityk/powershellterminal

Додаток Б

Проектування паттернів

У даному додатку наведено фрагменти вихідного коду, що демонструють практичну реалізацію структурних, породжуючих та поведінкових патернів проєктування у системі "PowerShell Terminal".

Б.1. Патерн Bridge (Міст)

Призначення: Відокремлення високорівневої логіки терміналу (збереження стану, навігація) від низькорівневої реалізації виконання команд (PowerShell, CMD, Remote).

Лістинг Б.1 – Реалізація патерну Bridge

```
// 1. Implementor (Інтерфейс реалізації)
public interface IShellEngine
{
    string Execute(string command, string workingDirectory);
    string GetEngineName();
}

// 2. Concrete Implementor (Конкретна реалізація для PowerShell)
public class PowerShellEngine : IShellEngine
{
    public string Execute(string command, string workingDirectory)
    {
        var processInfo = new ProcessStartInfo
        {
            FileName = "powershell.exe",
            Arguments = $"-NoProfile -ExecutionPolicy Bypass -Command \"{command}\"",
            WorkingDirectory = workingDirectory,
```

```
            RedirectStandardOutput = true,
            UseShellExecute = false,
            CreateNoWindow = true
        };
    }
}

// 3. Abstraction (Абстракція)
public class TerminalSystem
{
    protected IShellEngine _engine; // Посилання на реалізацію (Bridge)
    public string CurrentDirectory { get; private set; }

    public TerminalSystem(IShellEngine engine)
    {
        _engine = engine;
        CurrentDirectory = Directory.GetCurrentDirectory();
    }

    // Делегування виконання через міст
    public string RunCommand(string cmd)
    {
        return _engine.Execute(cmd, CurrentDirectory);
    }
}
```

Б.2. Патерн Strategy (Стратегія)

Призначення: Забезпечення можливості динамічної зміни алгоритму візуального оформлення (теми) інтерфейсу без зміни логіки форми.

Лістинг Б.2 – Реалізація патерну Strategy

```
// 1. Strategy Interface
public interface IThemeStrategy
{
    string ThemeName { get; }
    string BackgroundColor { get; }
    string ForegroundColor { get; }
    string CursorColor { get; }
}

// 2. Concrete Strategy (Тема Matrix)
public class MatrixThemeStrategy : IThemeStrategy
```

```
{
    public string ThemeName => "Dark Matrix";
    public string BackgroundColor => "#000000";
    public string ForegroundColor => "#00FF00";
    public string CursorColor => "#00FF00";
}

// 3. Context (Контекст використання)
public class ThemeContext
{
    private IThemeStrategy _strategy;

    public void SetStrategy(IThemeStrategy strategy)
    {
        _strategy = strategy;
    }

    public void ApplyTheme(Control control)
    {
        control.BackColor = ColorTranslator.FromHtml(_strategy.BackgroundColor);
        control.ForeColor = ColorTranslator.FromHtml(_strategy.ForegroundColor);
    }
}
```

Б.3. Патерн Command (Команда)

Призначення: Інкапсуляція запиту на виконання скрипта в об'єкт для параметризації клієнтів, логування та реалізації перевірок безпеки.

Лістинг Б.3 – Реалізація патерну Command

```
// 1. Command Interface
public interface ICommand
{
    void Execute();
}

// 2. Concrete Command (Виконання скрипта)
public class RunScriptCommand : ICommand
{
    private readonly TerminalSystem _system;
    private readonly string _script;
    private readonly UserProfile _user;
```

```
    public RunScriptCommand(TerminalSystem system, string script, UserProfile user)
    {
        _system = system;
        _script = script;
        _user = user;
    }

    public void Execute()
    {
        // Proxy-логіка: Перевірка прав доступу перед виконанням
        if (_user.Role != "Admin" && IsDestructiveCommand(_script))
        {
            Console.WriteLine("[SECURITY ALERT] Command restricted.");
            return;
        }

        string result = _system.RunCommand(_script);
    }
}

// 3. Invoker (Ініціатор)
public class CommandInvoker
{
    public void ExecuteCommand(ICommand command)
    {
        command.Execute();
    }
}
```

Б.4. Патерн Abstract Factory (Абстрактна фабрика)

Призначення: Створення сімейств пов'язаних об'єктів (елементів інтерфейсу) для різних ролей користувачів (Admin vs User) без прив'язки до конкретних класів.

Лістинг Б.4 – Реалізація патерну Abstract Factory

```
// 1. Abstract Factory
public interface ISessionFactory
{
    string CreateHeader();
    Prompt CreatePrompt();
}

// 2. Concrete Factory (Фабрика для Адміністратора)
```

```
public class AdminSessionFactory : ISessionFactory
{
    public string CreateHeader() => "ADMINISTRATOR: PowerShell Terminal [ELEVATED]";

    public Prompt CreatePrompt()
    {
        return new Prompt { Text = "PS ADMIN # >", Color = Color.Red };
    }
}

// 3. Concrete Factory (Фабрика для Користувача)
public class UserSessionFactory : ISessionFactory
{
    public string CreateHeader() => "PowerShell Terminal [User Mode]";

    public Prompt CreatePrompt()
    {
        return new Prompt { Text = "PS User $ >", Color = Color.Green };
    }
}
```

Б.5. Патерн Interpreter (Інтерпретатор)

Призначення: Визначення граматики мови терміналу та інтерпретація введених рядків для розрізнення внутрішніх команд (cd, help) та зовнішніх викликів.

Лістинг Б.5 – Реалізація патерну Interpreter

```
// 1. Abstract Expression
public interface IExpression
{
    void Interpret(InterpreterContext context);
}

// 2. Terminal Expression (Команда Change Directory)
public class ChangeDirectoryExpression : IExpression
{
    private readonly TerminalSystem _system;
    private readonly string _path;

    public void Interpret(InterpreterContext context)
    {
        // Змінює стан системи (current directory)
```

```
        string result = _system.ChangeDirectory(_path);
        context.Output = result;
    }
}

// 3. Client / Parser (Лексичний аналізатор)
public class ExpressionParser
{
    public IExpression Parse(string input)
    {
        var trimmed = input.Trim();

        if (trimmed.StartsWith("cd "))
            return new ChangeDirectoryExpression(_system, trimmed.Substring(3));

        if (trimmed.Equals("help"))
            return new HelpExpression();

        if (trimmed.Equals("history"))
            return new HistoryExpression(_user.ProfileId);

        return new SystemCommandExpression(_system, trimmed);
    }
}
```