Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні систем та технологій

Лабораторна робота № 3

із дисципліни «Технології розроблення програмного забезпечення»

Тема: «Основи проектування розгортання»

Виконав
Студент групи ІА-31: Перевірив:
Олея М. С. Мягкий М. Ю.

Київ 2025

Зміст

1. **Мета:**

Навчитися проєктувати діаграми розгортання та компонентів для системи що проєктується, а також розробляти діаграми взаємодії, а саме діаграми послідовностей, на основі сценаріїв зроблених в попередній лабораторній роботі.

2. **Хід роботи:**

Варіант - 8

Powershell terminal (strategy, command, abstract factory, bridge, interpreter,

client-server)

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)

1) Ознайомитись з короткими теоретичними відомостями.
2) Проаналізувати діаграми створені в попередній лабораторній роботі а також тему системи та спроєктувати діаграму розгортання використання відповідно до обраної теми лабораторного циклу

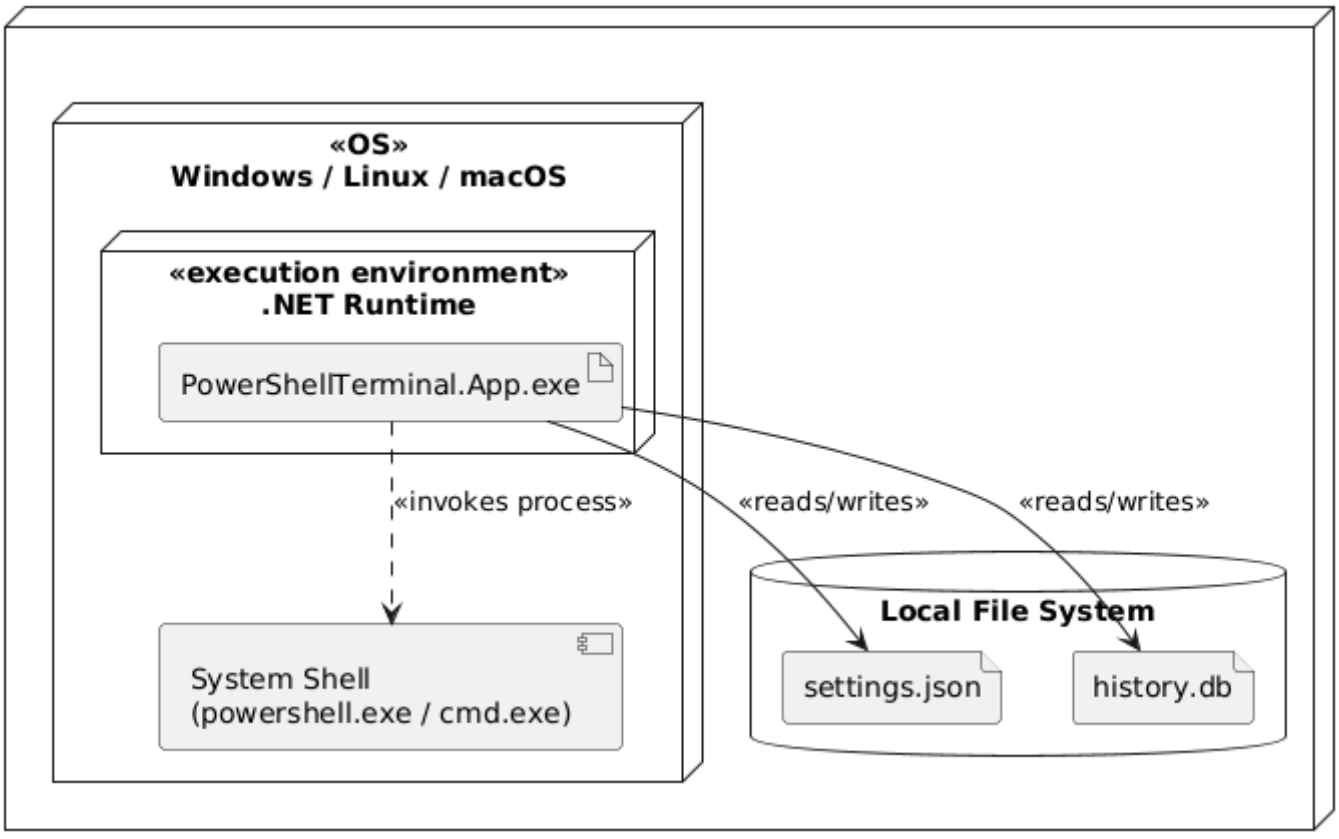

Рис.1 – **Діаграма розгортання**

На діаграмі (Рис. 1) зображено фізичну архітектуру програмного засобу «PowerShell Terminal». Система розгортається як настільний додаток (Desktop Application) на робочій станції користувача. Основними вузлами діаграми є:

1. User Workstation: Фізичний пристрій (ПК або ноутбук), на якому виконується програма.
2. Execution Environment (.NET Runtime): Середовище виконання, необхідне для роботи додатку, написаного мовою C#.
3. PowerShellTerminal.App.exe: Виконуваний файл самого термінала.
4. System Shell: Зовнішній компонент операційної системи (реальний powershell.exe або cmd.exe), до якого звертається наш додаток для безпосереднього виконання системних команд. Це реалізує патерн *Client-Server* на рівні процесів, де GUI є клієнтом, а системне ядро — сервером виконання.
5. Local File System: Використовується для збереження конфігураційних файлів (settings.json) та бази даних історії команд (history.db), доступ до яких здійснюється через рівень абстракції репозиторіїв.

3) Розробити діаграму компонентів для проєктованої системи

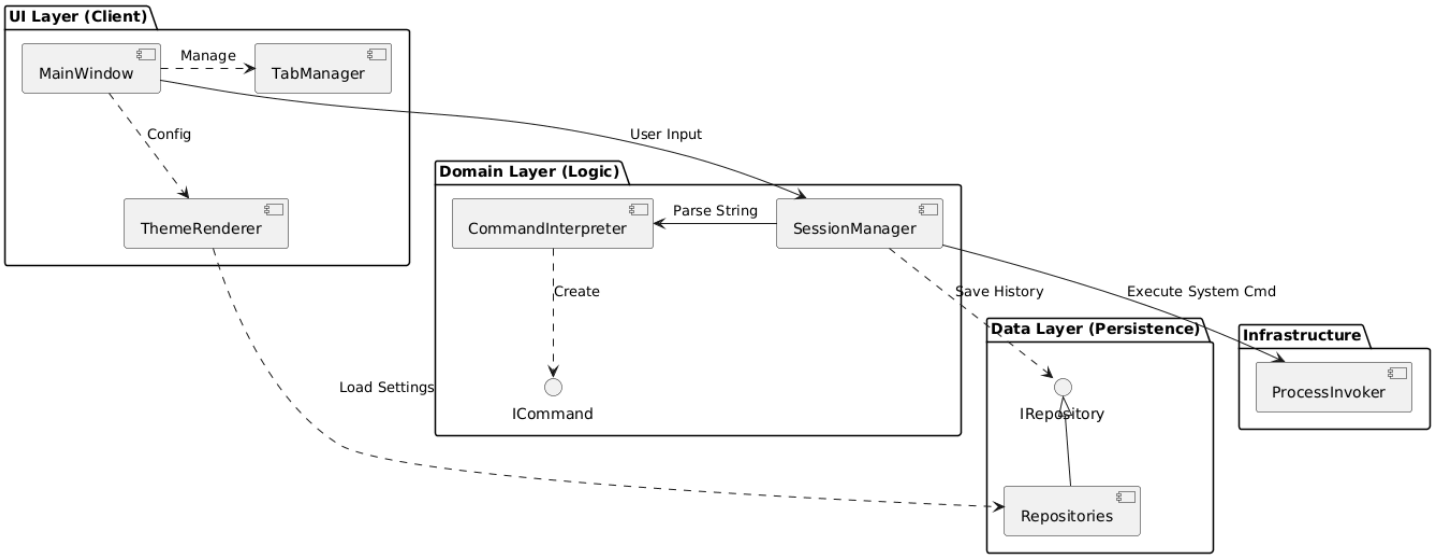

Рис.2 – Діаграма компонентів

На діаграмі компонентів (Рис. 2) відображено модульну структуру системи, яка розділена на три логічні шари:

1. UI Layer (Рівень представлення): Відповідає за взаємодію з користувачем. Компонент MainWindow приймає ввід, TabManager керує вкладками, а ThemeRenderer відповідає за візуальний стиль (Strategy pattern).
2. Domain Layer (Бізнес-логіка): Центральний шар обробки. SessionManager керує станом активної сесії. CommandInterpreter реалізує патерн Interpreter для розбору текстових команд і перетворення їх на об'єкти, що реалізують інтерфейс ICommand.
3. Data Layer (Дані): Містить компоненти доступу до даних (Repositories), які через інтерфейс IRepository забезпечують збереження історії та налаштувань, приховуючи деталі роботи з файловою системою.
4. Infrastructure: Компонент ProcessInvoker відповідає за низькорівневий запуск процесів ОС.

4) Розробити як мінімум дві діаграми послідовностей для сценаріїв прописаних в попередній лабораторній роботі

Сценарії використання:

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.

Табл. 1 Виконання команди

Основний потік:

1. Користувач вводить текст команди в інтерфейсі.
2. MainWindow передає рядок у поточну TerminalSession.
3. Сесія звертається до Interpreter для аналізу синтаксису.

4. Після успішного парсингу ініціюється запуск системного процесу (SystemProcess).
5. Результат виконання повертається у сесію та відображається користувачеві.
6. Сесія асинхронно зберігає запис про команду в HistoryRepository.

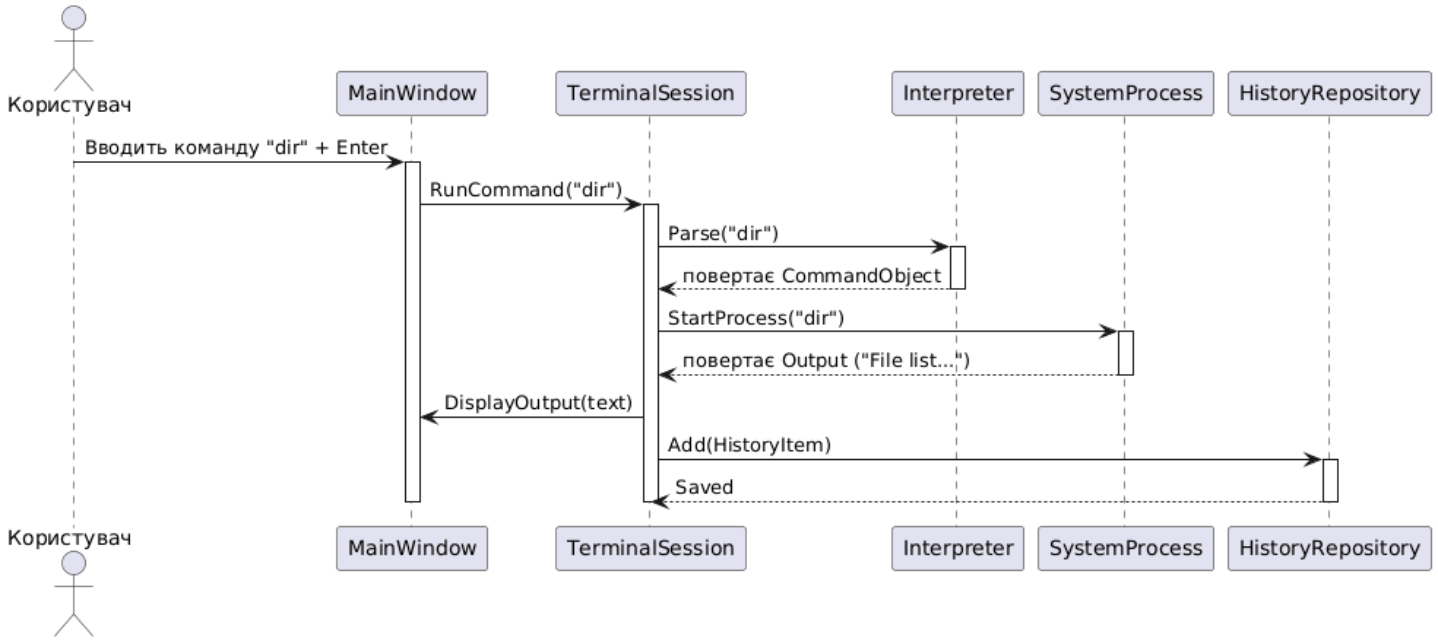

Рис. 3 – Діаграма послідовності Виконання команди

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.

Табл. 2 Налаштування теми

Основний потік:

1. Користувач обирає нову тему зі списку в налаштуваннях.

2. MainWindow викликає метод SetTheme у менеджера тем.
3. ThemeManager миттєво застосовує нові кольори до інтерфейсу (оновлення View).
4. Менеджер викликає SettingsRepository для збереження вибору користувача.
5. Репозиторій записує зміни у конфігураційний файл, щоб налаштування збереглися при наступному запуску.

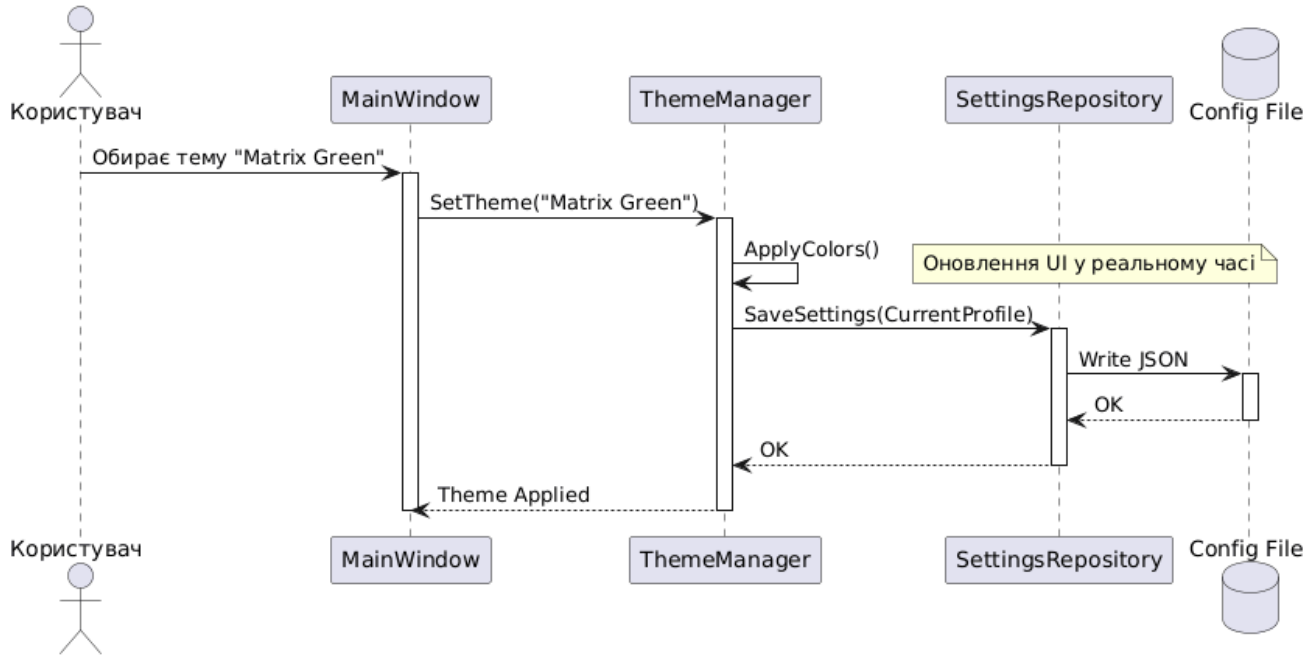

Рис. 4 – Діаграма послідовності Зміна колірної схеми

5) На основі спроєктованих діаграм розгортання та компонентів доопрацювати програмну частину системи. Реалізація системи, додатково до попередньої реалізації, повинна містити як мінімум дві візуальні форми. В системі вже повинен бути повністю реалізована архітектура (повний цикл роботи з даними від вводу на формі до збереження їх в БД і подальшій виборці з БД та відображенням на UI).

Програмна реалізація

a) LoginForm.cs

```
using System;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using PowerShellTerminal.App.Data;
using PowerShellTerminal.App.Domain.Entities;
```

```
namespace PowerShellTerminal.App.UI.Forms
{
    public class LoginForm : Form
    {
        private TextBox _txtUsername;
        private Button _btnLogin;
        public UserProfile? LoggedInUser { get; private set; }

        public LoginForm()
        {
            this.Text = "Вхід у термінал";
            this.Size = new Size(300, 150);
            this.StartPosition = FormStartPosition.CenterScreen;

            var lblName = new Label() { Text = "Ім'я профілю:", Top = 20, Left = 20, Width =
240 };
            _txtUsername = new TextBox() { Top = 45, Left = 20, Width = 240 };
            _btnLogin = new Button() { Text = "Увійти / Створити", Top = 75, Left = 20, Width
= 240, BackColor = Color.LightGray };
            _btnLogin.Click += OnLoginClick;

            this.Controls.Add(lblName);
            this.Controls.Add(_txtUsername);
            this.Controls.Add(_btnLogin);
        }

        private void OnLoginClick(object? sender, EventArgs e)
        {
            var name = _txtUsername.Text;
            if (string.IsNullOrWhiteSpace(name)) return;

            using (var db = new AppDbContext())
            {
                var user = db.UserProfiles.FirstOrDefault(u => u.ProfileName == name);

                if (user == null)
                {
                    user = new UserProfile { ProfileName = name, CreatedAt = DateTime.Now,
ThemeId = 1 };
                    db.UserProfiles.Add(user);
                    db.SaveChanges();
                    MessageBox.Show("Створено новий профіль!");
                }
                else
                {
                    MessageBox.Show($"З поверненням, {user.ProfileName}!");
                }

                LoggedInUser = user;
                this.DialogResult = DialogResult.OK;
```

```
            }
        }
    }
}
```

б) SettingsForm.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;
using PowerShellTerminal.App.Data;
using PowerShellTerminal.App.Domain.Entities;

namespace PowerShellTerminal.App.UI.Forms
{
    public class SettingsForm : Form
    {
        private UserProfile _user;
        private ComboBox _cmbThemes;
        private Button _btnSave;

        public SettingsForm(UserProfile user)
        {
            _user = user;
            this.Text = $"Налаштування: {_user.ProfileName}";
            this.Size = new Size(300, 200);
            this.StartPosition = FormStartPosition.CenterScreen;

            var lblTheme = new Label() { Text = "Оберіть тему:", Top = 20, Left = 20 };

            _cmbThemes = new ComboBox() { Top = 45, Left = 20, Width = 240 };
            _cmbThemes.Items.AddRange(new object[] { "Dark Matrix", "PowerShell Blue", "Ubuntu
Purple" });
            _cmbThemes.SelectedIndex = 0;

            _btnSave = new Button() { Text = "Зберегти в БД", Top = 80, Left = 20, Width =
240, BackColor = Color.LightBlue };
            _btnSave.Click += OnSaveClick;

            this.Controls.Add(lblTheme);
            this.Controls.Add(_cmbThemes);
            this.Controls.Add(_btnSave);
        }

        private void OnSaveClick(object? sender, EventArgs e)
        {
            using (var db = new AppDbContext())
            {
                var userToUpdate = db.UserProfiles.Find(_user.ProfileId);
                if (userToUpdate != null)
                {
```

```
                    userToUpdate.CreatedAt = DateTime.Now;

                    db.SaveChanges();
                    MessageBox.Show($"Тему '{_cmbThemes.SelectedItem}' збережено для
{userToUpdate.ProfileName}!");
                }
            }
            this.Close();
        }
    }
}
```

в) AppDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using PowerShellTerminal.App.Domain.Entities;

namespace PowerShellTerminal.App.Data
{
    public class AppDbContext : DbContext
    {
        public DbSet<UserProfile> UserProfiles { get; set; }
        public DbSet<Theme> Themes { get; set; } // <--- ДОДАЙ ЦЕЙ РЯДОК

        protected override void OnConfiguring(DbContextOptionsBuilder options)
            => options.UseSqlite("Data Source=terminal.db");
    }
}
```

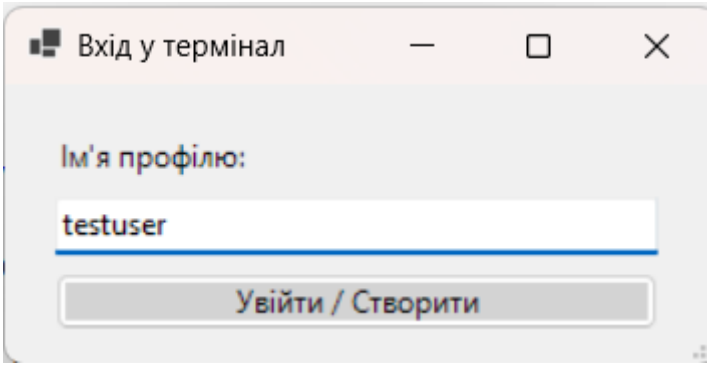

Рис. 5 Форма авторизації/реєстрації

ProfileId	ProfileName	*ThemeId*	CreatedAt
Фільтр	Фільтр	Фільтр	Фільтр
1	testuser	1	2025-12-19 23:01:04.0527302

Рис. 6 Створено запис в БД(SQLite)

Id	Username	PasswordHash	Role	SettingsJson	CreatedAt
Фільтр	Фільтр	Фільтр	Фільтр	Фільтр	Фільтр
11111111-1111-1111-1111-111111111111	admin	240be518fabd2724ddb6f04eeb1da5967448…	Admin	*NULL*	2025-12-11 19:03:09.415
22222222-2222-2222-2222-222222222222	user	e606e38b0d8c19b24cf0ee3808183162ea7c…	User	*NULL*	2025-12-11 19:03:09.415
6C709AAE-1894-4CFC-B924-89426C23B836	Vladislav	2052a537688420f2375cc3a708448ac628e5…	User	*NULL*	2025-12-11 19:19:13.638

Рис. 7 Таблиця SQLite

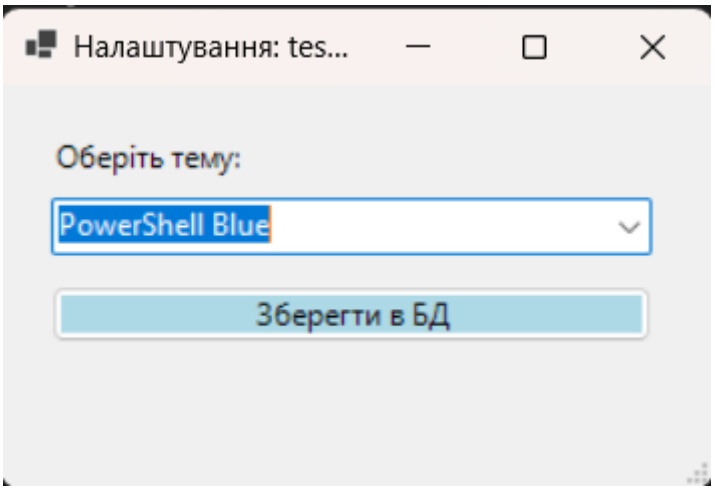

Рис. 8 Успішна авторизація

3. Висновок

У ході виконання лабораторної роботи було спроєктовано фізичну та логічну архітектуру програмного засобу «PowerShell Terminal». Розроблено UML-діаграми розгортання та компонентів, що визначили розміщення програмних модулів на робочій станції користувача та їх взаємодію з операційною системою і локальним сховищем даних. Створено діаграми послідовності для ключових сценаріїв використання, що деталізують логіку виконання команд та керування конфігурацією. Практична частина роботи полягала у реалізації програмного прототипу з використанням технологій Windows Forms та Entity Framework Core, що демонструє повний цикл роботи з базою даних (SQLite) — від авторизації користувача та створення профілю до збереження і зчитування налаштувань інтерфейсу, підтверджуючи працездатність запропонованої архітектури.

4. Контрольні питання

1. Що собою становить діаграма розгортання?

Діаграма розгортання — це UML-схема, яка показує фізичну конфігурацію системи: на яких пристроях або середовищах виконання розташовані компоненти програмного забезпечення та як ці елементи взаємодіють у реальних умовах.

2. Які бувають види вузлів на діаграмі розгортання?

На діаграмі розгортання розрізняють два типи вузлів:

- Апаратні вузли — конкретні фізичні пристрої (сервери, ПК, смартфони).
- Програмні вузли — платформи або середовища виконання (ОС, віртуальні машини, контейнери).

3. Які бувають зв’язки на діаграмі розгортання?

Основні зв’язки:

- Асоціація — показує фізичне або логічне з’єднання між вузлами.
- Залежність — відображає, що один вузол чи розміщений на ньому компонент потребує ресурси або функції іншого.

4. Які елементи присутні на діаграмі компонентів?

На діаграмі компонентів зазвичай присутні:
компоненти, інтерфейси, порти, залежності між компонентами, пакети та підсистеми.

5. Що становлять собою зв’язки на діаграмі компонентів?

Зв’язки відображають, як один компонент використовує інтерфейси іншого або залежить від нього, тобто показують функціональні та структурні взаємозв’язки між компонентами системи.

6. Які бувають види діаграм взаємодії?

До діаграм взаємодії належать:

- діаграми послідовностей,
- діаграми комунікацій,
- діаграми часу,
- діаграми огляду взаємодії.

7. Для чого призначена діаграма послідовностей?

Діаграма послідовностей використовується для відображення порядку передачі повідомлень і викликів між об’єктами під час виконання конкретного сценарію або функціонального процесу.

8. Які ключові елементи можуть бути на діаграмі послідовностей?

На ній можуть бути присутні: актори, об’єкти, лінії життя, повідомлення (синхронні та асинхронні), а також області активації, що показують виконання операцій.

9. Як діаграми послідовностей пов’язані з діаграмами варіантів використання?

Вони деталізують кожен варіант використання, показуючи покрокову взаємодію між учасниками сценарію та системою, яка реалізує цей варіант.

10. Як діаграми послідовностей пов’язані з діаграмами класів?

Об’єкти, що взаємодіють на діаграмі послідовностей, є конкретними екземплярами класів, а повідомлення між ними відповідають викликам методів, описаних у діаграмі класів.