Міністерство освіти і науки України
Національний університет
«Львівська політехніка»
Кафедра електронних обчислювальних машин

КУРСОВИЙ ПРОЄКТ

з дисципліни “Системне програмне забезпечення”

на тему: “Розробка програмного забезпечення для управління службами Windows. Розробка програми, що дозволяє керувати різними службами Windows, зупиняти, запускати та перезапускати їх.”

Студента 3-го курсу групи КІ-306
123 «Комп’ютерна інженерія»
Савіцький Н. О.
Керівник
Олексів М. В.

Національна шкала: _____
Кількість балів: _____
Оцінка ECTS: _____

Члени комісії: _____ _____
(підпис) (прізвище та ініціали)

_____ _____
(підпис) (прізвище та ініціали)

_____ _____
(підпис) (прізвище та ініціали)

Львів 2024

ЗАВДАННЯ НА КУРСОВИЙ ПРОЕКТ

Основною метою курсового проекту "Windows Service Manager" є розробка та впровадження програмного забезпечення для ефективного управління службами операційної системи Windows. Програма буде забезпечувати користувачам зручний та інтуїтивно зрозумілий інтерфейс для зміни стану та способу запуску служб.

Завдання проекту

1. Аналіз існуючих рішень
2. Розробка архітектури програми
3. Реалізація функціональних можливостей
    a. Перегляд доступних служб
    b. Запуск, зупинка, перезапуск, призупинення, продовження роботи служб
    c. Вибір способу запуску служби
    d. Пошук служб
    e. Сортування служб
    f. Інформація про служби
4. Провести тестування та відлагодження, усунути знайдені проблеми
5. Створити чітку та детальну документацію про проект

Очікувані результати

Результатом виконання курсового проекту "Windows Service Manager" має бути функціональна програма, яка надасть користувачам зручний та ефективний інструмент для управління службами операційної системи Windows. Програма повинна бути стабільною, надійною та забезпечувати необхідний функціонал для успішного управління службами операційної системи.

АНОТАЦІЯ

У цій курсовій роботі презентується розробка програмного забезпечення для управління службами в операційній системі Windows. Основною метою цього програмного засобу є забезпечення користувачам зручного та ефективного інструменту для контролю за роботою служб, що запускаються в операційній системі.

У роботі досліджено основні методи та інструменти, що використовуються для роботи зі службами в ОС Windows. Описано процес розробки програми, включаючи вибір мови програмування, структуру програмного забезпечення та реалізацію його функціональних можливостей.

Розроблене програмне забезпечення має зручний графічний інтерфейс, який дозволяє користувачам легко переглядати та керувати різними службами в операційній системі Windows. Основними принципами, на яких базується робота програми, є забезпечення ефективності та надійності.

Результати тестування показують, що програмне забезпечення успішно виконує поставлені завдання та може бути корисним інструментом для адміністраторів систем, технічних спеціалістів та інших користувачів, які потребують зручного та ефективного інструменту для управління службами в операційній системі Windows.

ЗМІСТ

ВСТУП

В сучасному світі комп'ютерні системи стають все більш складними і важливими для різноманітних аспектів життя, починаючи від домашнього використання до великих корпоративних середовищ. Для надійної та ефективної роботи цих систем вимагається регулярний моніторинг та управління їхнім станом. Це особливо актуально для операційних систем Windows, які використовуються в різних областях і забезпечують функціональність для великої кількості застосунків.

Моніторинг та управління службами Windows є однією з важливих складових ефективного функціонування комп'ютерних систем. Це не лише дозволяє вчасно виявляти та усувати проблеми, але й оптимізує використання ресурсів, забезпечуючи високу продуктивність та ефективність.

Цей курсовий проект спрямований на розробку програмного забезпечення - менеджера служб Windows, який надає користувачам зручний та ефективний інструмент для моніторингу та управління різними службами, що запускаються в операційній системі Windows. Це програмне забезпечення дозволить користувачам легко взаємодіяти зі службами, перевіряти їхній стан, запускати, призупиняти, перезавантажувати та змінювати параметри запуску.

Метою цієї роботи є створення ефективного та зручного інструменту для користувачів Windows, який дозволить їм ефективно управляти службами та забезпечувати контрольовану роботу операційної системи. У процесі розробки будуть використані сучасні методи та технології програмування, а також звернута увага на зручний та інтуїтивно зрозумілий інтерфейс користувача.

Ця курсова робота розглядає теоретичні та практичні аспекти розробки програмного забезпечення для управління службами Windows, а також може служити важливим внеском у сферу системного адміністрування та оптимізації роботи комп'ютерних систем під управлінням операційної системи Windows.

1. АНАЛІТИЧНИЙ ОГЛЯД

Служба Windows – це програма, яка працює у фоновому режимі на операційній системі Windows і надає певні функції або послуги. Служби можуть бути автоматично запущені при завантаженні системи або запускатися відповідно до запиту користувача або інших програм. Вони можуть виконувати різноманітні завдання, включаючи моніторинг, обробку даних, комунікацію з іншими програмами або обслуговування конкретного пристрою чи послуги. Розробка програми для управління сервісами Windows потребує знання специфічних інструментів і технологій, що дозволяють взаємодіяти з Диспетчером управління службами операційної системи.

1.1.Аналіз методів для керування службами Windows

Для отримання необхідної інформації про стан сервісів ОС Windows та керування ними використовуються системні API, що дозволяють доступ до низькорівневих даних:

- Service Control Manager (SCM): набір функцій, які забезпечують доступ до основних функцій керування службами операційної системи Windows. Вони дозволяють відкривати доступ до менеджера керування службами, відкривати та закривати дескриптори служб, керувати станом служби, таким як запуск, зупинка, пауза, і змінювати конфігурацію служб.
- Windows Management Instrumentation (WMI): набір технологій, який надає доступ та керування інформацією та операціями в операційній системі Windows. Він дозволяє виконувати різноманітні завдання адміністрування, включаючи моніторинг, налаштування та керування компонентами системи, такими як служби, процеси, події та ресурси. Використовує SQL-подібну мову запитів для взаємодії з цією інформацією.

Використання даних методів надає можливість отримувати детальну інформацію про стан та управління службами, що є важливим для ефективного моніторингу та керування сервісами в операційній системі Windows.

Використання бібліотек та фреймворків

У сфері розробки програми для керування сервісами Windows існує дуже обмежена кількість готових бібліотек та фреймворків, які спрощують процес створення утиліт для управління службами Windows. Ці бібліотеки надають високорівневі інтерфейси для доступу до системних функцій, дозволяючи

розробникам зосередитися на функціональних можливостях своєї програми, а не на деталях роботи з операційною системою.

.NET System.ServiceProcess:

- **Опис**: Набір класів у середовищі .NET дозволяє взаємодіяти зі службами Windows, надаючи зручний інтерфейс для керування та моніторингу служб.
- **Переваги**: Простота використання в середовищі .NET, вбудована підтримка для роботи зі службами Windows, можливість отримання детальної інформації про стан та параметри служб.

Використання цих бібліотек та фреймворків забезпечує розробникам зручні інструменти для зручної взаємодії з сервісами Windows, що економить розробникам багато часу при імплементації необхідних функцій.

1.2. Відомі розробки

Системні утиліти Windows

1. **Services.msc**:
   - Опис: Services.msc є вбудованою системною утилітою в операційній системі Windows, яка надає графічний інтерфейс для керування сервісами. Вона дозволяє переглядати список установлених служб, змінювати їх параметри, запускати, призупиняти або вимикати.
   - Переваги: Легкий доступ до керування сервісами через графічний інтерфейс, можливість зміни параметрів служб безпосередньо.
2. **Net.exe**:
   - Опис: Net.exe є утилітою з консольним інтерфейсом в операційній системі Windows, яка дозволяє керувати мережевими ресурсами, включаючи служби. За допомогою net.exe можна запускати, призупиняти, вимикати та переглядати інформацію про статус служб.
   - Переваги: Зручний інтерфейс командного рядка для керування службами, можливість автоматизації операцій зі службами за допомогою скриптів.
3. **Sc.exe**:
   - Опис: Sc.exe є утилітою з консольним інтерфейсом в операційній системі Windows, яка надає розширені можливості для керування службами. Вона дозволяє налаштовувати різноманітні параметри служб, включаючи тип запуску, залежності та інші.

- Переваги: Розширені можливості налаштування служб через командний рядок, можливість автоматизувати складні операції зі службами.

Комерційні та відкриті утиліти

1. **FireDaemon (Комерційний)**:
   - Опис: FireDaemon - це набір інструментів для створення, управління та моніторингу служб Windows. Він дозволяє запускати будь-яку програму як службу, а також забезпечує ряд додаткових функцій для керування сервісами.
   - Переваги: FireDaemon має інтуїтивний інтерфейс, зручні функції моніторингу та керування, а також можливість автоматичного перезапуску служби в разі відмови.
2. **WinSW (Відкритий)**:
   - Опис: Windows Service Manager (WinSW) - це відкрите програмне забезпечення, яке дозволяє створювати, запускати та керувати службами Windows звичайних .NET додатків. WinSW використовує конфігураційні файли XML для налаштування служб.
   - Переваги: WinSW має відкритий вихідний код, що дозволяє розширювати його функціональність, простий у використанні та налаштуванні, інтеграцію з .NET додатками.

1.3. Цільова аудиторія

Програма "Windows Services Manager" спрямована на різні категорії користувачів операційної системи Windows, які мають потребу в управлінні та моніторингу служб. Нижче наведено детальний опис кожної цільової аудиторії:

Системні адміністратори:

• **Опис:** Ця категорія користувачів відповідає за адміністрування та підтримку операційних систем Windows у великих мережах або на серверах. Вони здійснюють управління та моніторинг служб для забезпечення безперебійної роботи IT-інфраструктури.

• **Потреби:** Централізоване керування та моніторинг служб, виявлення та виправлення проблем, планування роботи та поновлення.

• **Функції, що задовольняють потреби:** Налаштування робочих параметрів та перевірка їх статусу, можливість зупинки, запуску та перезавантаження служб, моніторинг подій та стану роботи служб.

Розробники програмного забезпечення:

• **Опис:** Професіонали, які розробляють програмне забезпечення для операційних систем Windows. Вони використовують утиліти управління службами для тестування, налагодження та підтримки власних додатків.

• **Потреби:** Забезпечення стабільної роботи власних служб, виявлення та виправлення помилок у роботі додатків.

• **Функції, що задовольняють потреби:** Можливість керування статусом та конфігурацією власних служб.

Досвідчені користувачі:

• **Опис:** Користувачі, які мають достатню обізнаність в сервісах Windows та бажають контролювати та налаштовувати роботу служб у своїй операційній системі.

• **Потреби:** Розширені можливості управління та моніторингу служб, гнучка настройка параметрів роботи, інформація про стан служб.

• **Функції, що задовольняють потреби:** Детальний перегляд та керування службами, налаштування автозапуску, аналіз статусів служб.

Програма "Windows Services Manager" розроблена з урахуванням потреб кожної з цих груп користувачів, надаючи їм інструменти для ефективного управління та моніторингу служб у їх операційних системах Windows.

Висновки

Управління сервісами Windows є важливим аспектом забезпечення стабільної та ефективної роботи операційної системи. Існує широкий спектр методів та інструментів для керування сервісами на платформі Windows. Використання Service Control Manager (SCM) API, WMI, а також готових бібліотек, таких як .NET System.ServiceProcess, дозволяє розробникам створювати потужні утиліти для управління службами.

Розробка власної утиліти для керування сервісами на базі цих методів та інструментів дозволяє отримати гнучкий та зручний інструмент для адміністрування служб Windows. Це може бути корисним як для системних адміністраторів, які керують великою кількістю серверів та робочих станцій, так і для розробників програмного забезпечення, які потребують інтеграції управління сервісами у свої додатки, а також для досвідчених користувачів, які бажають мати більше контролю над своїми системами.

2. ПРОЕКТУВАННЯ ПРОГРАМИ

2.1.Обґрунтування засобів для розробки

Розробка утиліти для управління службами Windows включає використання двох мов програмування: C++ для розробки динамічної бібліотеки (DLL) та Python для створення графічного інтерфейсу користувача (GUI). Ця комбінація дозволяє максимально ефективно використовувати можливості обох мов і забезпечити зручність та функціональність утиліти.

Вибір технологій

C++ для розробки DLL було обрано мову C++, яка забезпечує високу продуктивність та доступ до низькорівневих функцій Windows API. Очевидно, що саме C++ дозволить найефективніше керувати сервісами Windows.

WinAPI пропонує засоби для роботи з Service Control Manager (SCM) у бібліотеці windows.h, які надають доступ до функцій, що дозволяють взаємодіяти з системними службами в операційній системі Windows. SCM є важливою складовою для управління різними аспектами служб, такими як їх стан, тип запуску, запуск, зупинка, пауза та відновлення.

Python було обрано як основну мову програмування для створення GUI завдяки її зручності, швидкості та простоті написання. Велика кількість як вбудованих так і сторонніх бібліотек дає повну можливість реалізувати ефективну взаємодію між GUI написаним на Python, та DLL розробленою на C++.

ctypes використовується для роботи з розробленою DLL на C++. Вона дозволяє викликати функції з DLL безпосередньо з коду Python, що забезпечує високу продуктивність та гнучкість у взаємодії з системними службами Windows. Це дозволяє використовувати можливості Windows API для управління службами операційної системи.

tkinter використовується для створення графічного інтерфейсу користувача (GUI). Це стандартна бібліотека Python для створення десктопних додатків, яка забезпечує простоту у використанні та широкі можливості для створення інтуїтивно зрозумілих інтерфейсів: з її використанням можна легко створювати різноманітні компоненти інтерфейсу, такі як кнопки, поля вводу, таблиці, що забезпечують зручність користування утилітою.

2.2. Високорівневий огляд програми

"Windows Services Manager" - це програма, що призначена для керування службами операційної системи Windows. Основною метою цієї програми є надання користувачам зручного інструменту для управління різними службами, що працюють на їхньому комп'ютері. Програма надає детальну інформацію про стан служб, дозволяючи користувачам здійснювати такі операції, як запуск,

зупинка, перезапуск, пауза та відновлення служб. Нижче наведено детальний опис архітектури та функціональних можливостей програми.

2.3.Архітектура програми

Програма "Windows Services Manager" складається з 2 ключових компонентів, які взаємодіють між собою для забезпечення ефективної взаємодії користувача з сервісами Windows:

1. **Інтерфейс користувача (GUI)**:
    - **Опис**: Інтерфейс користувача розроблений з використанням засобів доступних у бібліотеці tkinter для створення зручного та інтуїтивно зрозумілого інтерфейсу користувача. Використовує модуль ctypes для взаємодії з DLL.
    - **Функції**: Відображення списку служб у реальному часі, відображення їх стану та способу запуску, надання можливості керування присутніми службами, їх сортування.
2. **Модуль безпосередньої взаємодії зі службами (DLL)**:
    - **Опис**: Цей модуль відповідає за безпосередню взаємодію з сервісами Windows через Services Control Manager засобами WinAPI.
    - **Функції**: Отримання усіх сервісів з інформацією про їх стан та спосіб запуску, запуск, зупинка, перезапуск, призупинення, продовження роботи, зміна типу запуску, перевірка можливості зупинки та призупинення служби.

2.4.Функціональні можливості

1. **Перегляд доступних служб**
   Опис: Ця функція дозволяє користувачам переглядати список усіх доступних служб операційної системи Windows.
   Вимоги:
    - Відображення повного списку служб Windows.
    - Таблиця з відповідними стовпцями та смугою прокрутки.
    - Оновлення при фільтрації, сортуванні, після виконання дії.
    - Можливість обрати конкретну службу з таблиці для виконання подальших дій над нею.
    - Збереження вибору служби в таблиці при оновленні таблиці.
    - Зміна активності кнопок дій над сервісом залежно від стану служби (додатково перевірка можливості зупинки та паузи для запущених служб).
2. **Запуск, зупинка, перезапуск, призупинення, продовження роботи служб**

Опис: Можливість запуску вибраної служби для забезпечення її функціонування.

Вимоги:

- Можливість запуску окремих служб з інтерфейсу користувача.
- Відображення поточного стану служби перед та після виконання дії.
- Наявність відповідної кнопки з підказкою в інтерфейсі.
- Повідомлення в рядку стану про успішність або неуспішність виконання дії.

3. **Вибір способу запуску служби**

Опис: Можливість вибору способу запуску служби (автоматичний, при запуску системи, вручну, вимкнена).

Вимоги:

- Можливість зміни способу запуску окремих служб з інтерфейсу користувача.
- Відображення поточного способу запуску перед та після його зміненням.
- Наявність відповідної кнопки з підказкою в інтерфейсі.
- Повідомлення в рядку стану про успішність або неуспішність виконання дії.
- Окреме вікно для вибору способу запуску служби.

4. **Пошук служб**

Опис: Вбудований пошук для швидкого знаходження конкретних служб.

Вимоги:

- Поле для введення пошукового запиту.
- Наявність відповідної кнопки з підказкою в інтерфейсі.
- Оновлення списку служб відповідно до пошукового запиту при натисненні кнопки пошуку.
- Залишення лише тих служб, які в своїй назві містять введені в пошукове поле літери.

5. **Сортування служб**

Опис: Можливість сортування служб за різними критеріями для зручного перегляду.

Вимоги:

- Сортування служб за ім'ям.
- Сортування служб за станом.
- Сортування служб за способом запуску.
- Можливість реверсивності сортування служб за алфавітним порядком.
- Збереження вибраного стовпця та порядку сортування служб для забезпечення сортування при подальших оновленнях таблиці (при пошуку, після виконання дій)

6. **Інформація про служби**

Опис: Відображення детальної інформації про кожну службу, включаючи назву, стан та спосіб запуску.

Вимоги:

- Відображення назви служби.
- Відображення поточного стану служби (запущена, зупинена, призупинена).
- Відображення способу запуску служби (автоматичний, при запуску системи, вручну, вимкнений).

Висновки

Програма "Windows Services Manager" є ефективним інструментом для управління службами операційної системи Windows. Вона надає користувачам зручний інтерфейс для перегляду, запуску, зупинки, перезапуску та управління службами. Програма дозволяє швидко знаходити необхідні служби, налаштовувати способи їх запуску та отримувати детальну інформацію про кожну службу. Використання сучасних бібліотек та технологій робить "Windows Services Manager" надійним, гнучким та зручним у використанні, сприяючи стабільній та ефективній роботі комп'ютера.

3. РЕАЛІЗАЦІЯ

Для реалізації програми "Windows Services Manager" було обрано комбінацію мов програмування C++ та Python, щоб ефективно скористатися перевагами кожної з них. Цей підхід забезпечує високий рівень продуктивності та зручності використання.

Для безпосередньої взаємодії з сервісами Windows в C++ було використано:

- WinAPI: За допомогою бібліотеки windows.h та Service Control Manager (SCM), програма може взаємодіяти з системними службами Windows, включаючи їх запуск, зупинку, паузу та відновлення. Це забезпечує ефективне управління службами операційної системи.

Для реалізації GUI було використано такі стандартні та відкриті бібліотеки:

- ctypes: Ця бібліотека використовується для роботи з DLL, розробленою на C++. Вона дозволяє викликати функції з DLL безпосередньо з коду Python, забезпечуючи високу продуктивність та гнучкість у взаємодії з системними службами Windows.
- tkinter: Використовується для створення графічного інтерфейсу користувача. Це стандартна бібліотека Python для створення десктопних додатків, яка забезпечує простоту у використанні та широкі можливості для створення інтуїтивно зрозумілих інтерфейсів. З її допомогою можна легко створювати різноманітні компоненти інтерфейсу, такі як кнопки, поля вводу, таблиці тощо.

Завдяки поєднанню мов програмування C++ та Python, програма забезпечує високу продуктивність та зручність використання. Використання WinAPI для взаємодії з системними службами через C++ дозволяє ефективно керувати сервісами Windows. Інтеграція з Python за допомогою бібліотеки ctypes забезпечує безперебійну взаємодію між низькорівневими функціями C++ та графічним інтерфейсом користувача, створеним за допомогою tkinter. Цей підхід дозволяє створити інтуїтивно зрозумілий та функціональний інтерфейс, який робить процес управління службами зручним та ефективним. Завдяки використанню сучасних технологій та бібліотек, програма є надійною, гнучкою та зручною у використанні, що робить її корисною як для звичайних користувачів, так і для професійних системних адміністраторів.

3.1. Реалізація графічного інтерфейсу користувача

В програмі «Windows Services Manager» графічний інтерфейс користувача реалізовано з використанням можливостей вбудованої Python-бібліотеки tkinter.

Даний модуль дозволяє гнучко розміщувати віджети необхідні для оформлення GUI, а також підтримує використання як вбудованих так і користувацьких тем. Використано таку структуру розміщення елементів tkinter: основне вікно програми, фрейми які знаходяться в ньому, віджети які розміщені в фреймах.

Основне вікно програми складається з 4 фреймів (див. рис. 3.1):

1. Операції над службою
2. Пошук служб
3. Таблиця зі службами
4. Рядок стану

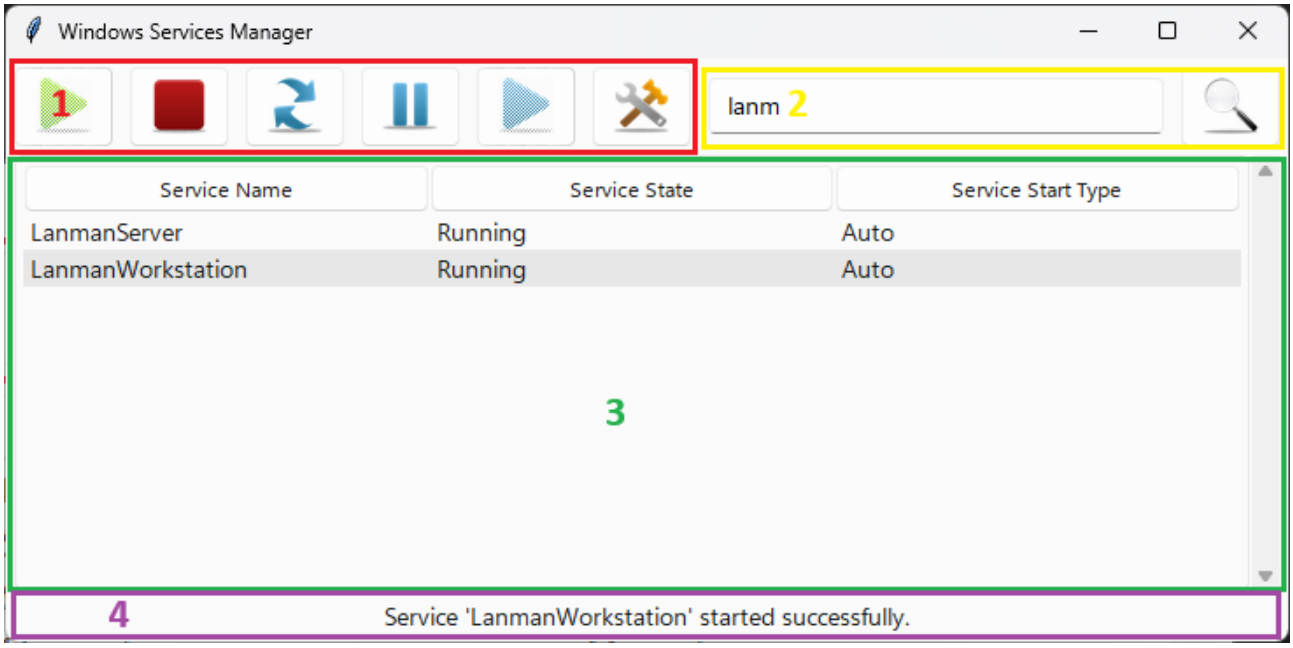
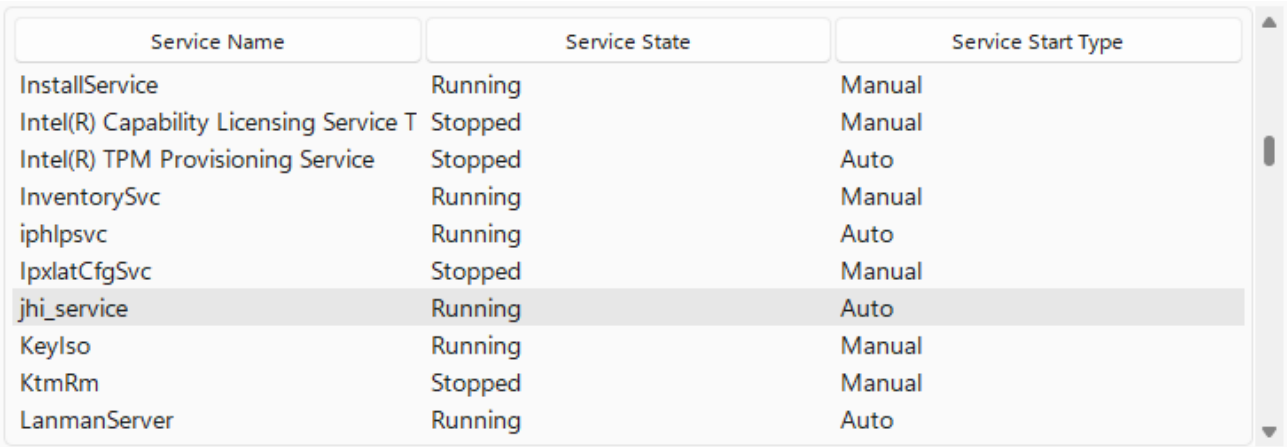


Рис.3.1. Поділ основного вікна програми на фрейми.

В програмі використана користувацька тема sv_ttk яка робить вигляд елементів tkinter більш сучасним та приємним для ока користувача.

Перегляд доступних служб



Service Name	Service State	Service Start Type
InstallService	Running	Manual
Intel(R) Capability Licensing Service T	Stopped	Manual
Intel(R) TPM Provisioning Service	Stopped	Auto
InventorySvc	Running	Manual
iphlpsvc	Running	Auto
IpxlatCfgSvc	Stopped	Manual
jhi_service	Running	Auto
KeyIso	Running	Manual
KtmRm	Stopped	Manual
LanmanServer	Running	Auto

Рис.3.2. Перегляд доступних служб.

Доступні служби можна переглянути в таблиці, яка містить відповідні стовпці, та має смугу прокрутки, розташовану праворуч. В таблиці можна обрати сервіс для подальших дій над ним, обраний сервіс зберігається при оновленні таблиці.

Запуск, зупинка, перезапуск, призупинення, продовження роботи служб

Рис.3.3. Панель кнопок дій над службою.

Зверху, над таблицею яка містить служби, знаходиться панель кнопок дій над службою (див рис. 3.3).

Для кнопки запуску служби було використано піктограму у вигляді повернутого зеленого трикутника (рис. 3.4), адже вона асоціюється з дією початку, або запуску.

Рис.3.4. Піктограма запуску служби.

Зупинка роботи служби позначена червоним квадратом (рис. 3.5), який також цілком асоціюється з виконанням дії зупинки.

Рис.3.5. Піктограма запуску служби.

Піктограма зображена на рис. 3.6 позначає кнопку перезапуску обраної служби.

Рис.3.6. Піктограма запуску служби.

Для призупинення роботи служби обрано піктограму яку зазвичай асоціюють з «паузою» (див. рис. 3.7).

Рис.3.7. Піктограма запуску служби.

Продовження роботи служби позначено піктограмою зображеною на рис. 3.8, вона цілком добре підходить для позначення дії продовження.

Рис.3.8. Піктограма запуску служби.

Вибір способу запуску служби

Дія вибору способу запуску служби позначається піктограмою зображеною на рис. 3.9, яка розташовується праворуч від інших дій над службою.

Рис.3.9. Піктограма запуску служби.

Також дана функція передбачає вікно для вибору типу запуску служби, в якому розташовані радіокнопки з варіантами вибору, а також кнопку підтвердження дії зміни способу запуску служби.

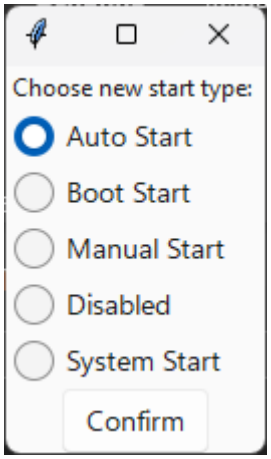


Рис.3.10. Вікно вибору способу запуску служби.

Пошук служб

Пошук служб розташований в окремому фреймі (рис. 3.11).

Рис.3.11. Пошук служб.

Для введення назви шуканрого сервісу присутнє поле для її вводу (див. рис. 3.12). При відсутності введеного в нього тексту, дане поле містить напис-підказку «Filter by name…» що додає функції пошуку інтуітивності.

Рис.3.12. Піктограма запуску служби.

Кнопка пошуку розміщується праворуч поля для вводу пошукового запиту, зображена піктограмою лупи (рис. 3.13), яка часто використовується для позначення пошуку.

Рис.3.13. Піктограма запуску служби.

Сортування служб

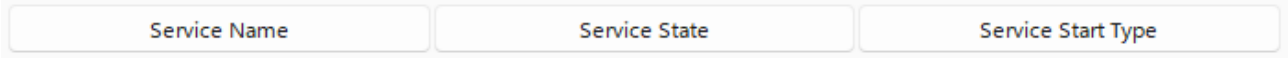


Рис.3.14. Заголовки таблиці служб.

Сортування реалізовано натисненням на заголовки (назви стовпців) в таблиці служб (див. рис. 3.14). Заголовки оформлено в кнопкоподібному вигляді, що візуально підказує користувачу про можливість натиснення на них для виконання якоїсь дії. Відповідно, можна побачити 3 види сортування: за назвою, за станом, за типом запуску служби.

Обране сортування зберігається при оновленні таблиці.

Інформація про служби

Графічно відображення інформації про служби реалізовується розподілом таблиці служб на 3 стовпці: назва, стан, тип запуску служби (див. рис. 3.15). Стан може мати такі значення: Stopped, Start Pending, Stop Pending, Running, Continue Pending, Pause Pending, Paused. Типи запуску: Auto Start, Boot Start, Manual Start, Disabled, System Start.

Service Name	Service State	Service Start Type
AdobeARMservice	Running	Auto
AJRouter	Stopped	Manual
ALG	Stopped	Manual
AppIDSvc	Stopped	Manual
Appinfo	Running	Manual
AppMgmt	Stopped	Manual
AppReadiness	Stopped	Manual
AppVClient	Stopped	Disabled
AppXSvc	Running	Manual
aspnet_state	Stopped	Manual

Рис.3.15. Інформація про служби.

Варто зазначити, що при наведенні на кожну з присутніх в програмі кнопок, висвічується підказка з написом про дію яку позначає кнопка (рис. 3.16).

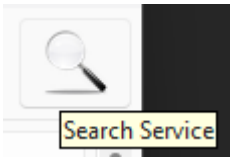


Рис.3.16. Підказка при наведенні на кнопку.

Також, слід зауважити відображення повідомлення про успішність або неуспішність кожної з дій в рядок стану, який знаходиться нижче таблиці служб (див. рис. 3.17).

Service 'LanmanWorkstation' started successfully.

Рис.3.17. Рядок стану.

Реалізований дизайн програми «Windows Services Manager» є цілком зручним та інтуітивним в використанні. Завдяки простому, та водночас зрозумілому дизайну, програма працює максимально продуктивно, не затрачаючи ресурси комп’ютера на роботу дизайну.

3.2. Високорівневий опис реалізації програмного рішення згідно розроблених вимог

Перегляд доступних служб

В Python засобами tkinter створюється підготовлена для заповнення службами таблиця, зі смугою прокрутки розташованою праворуч. За допомогою ctypes викликається функція DLL, яка засобами WinAPI звертається до Services Control Manager, отримує список усіх служб та повертає його в масив в Python код. Таблиця заповнюється отриманими службами, відповідно користувач може їх візуально сприймати.

Запуск, зупинка, перезапуск, призупинення, продовження роботи служб
В Python засобами tkinter створюються кнопки з відповідними піктограмами та підказками при наведенні на них. При натисненні однієї з кнопок (запуску, зупинки, перезапуску, призупинення, продовження), викликається відповідна функція DLL за допомогою ctypes, в яку передається назва обраного в таблиці сервісу. Ця функція засобами WinAPI звертається до Services Control Manager зі спробою відповідного контролю роботи служби, та повертає в Python код значення яке означає успішність або неуспішність виконання відповідного контролю. В рядку стану GUI виводиться відповідне повідомлення, в разі успішності виконання дії оновлюється таблиця зі службами.

Вибір способу запуску служби
В Python засобами tkinter створюється кнопка з відповідною піктограмою та підказкою при наведенні на неї. При натисненні цієї кнопки, відкривається вікно, яке містить радіокнопки з типами запуску (Auto Start, Boot Start, Manual Start, Disabled, System Start) та кнопка підтвердження зміни типу запуску. При виборі радіокнопки з типом запуску, відмінним від поточного, та натисненні кнопки підтвердження, за допомогою ctypes викликається функція з DLL, в яку передаються назва обраних служби та нового типу запуску. Ця функція засобами WinAPI звертається до Services Control Manager зі спробою змінення способу запуску в конфігурації відповідної служби, та повертає в Python код значення яке означає успішність або неуспішність виконання відповідного контролю. В рядку стану GUI виводиться відповідне повідомлення, в разі успішності виконання дії оновлюється таблиця зі службами.

Пошук служб
В Python засобами tkinter створюється поле для вводу назви служби, кнопка з відповідною піктограмою. та підказкою при наведенні на неї. При натисненні кнопки пошуку та наявності введеного в пошукове поле тексту, за допомогою ctypes викликається функція DLL, яка засобами WinAPI звертається до Services Control Manager, отримує список усіх служб та повертає його в масив в Python код. Таблиця заповнюється лише тими службами, які в своїй назві містять введений в пошукове поле текст.

Сортування служб
При натисненні на заголовки стовпців таблиці, здійснюється сортування усіх її рядків за обраним стовпцем. Стовпець та напрямок за яким здійснювалось

останнє сортування зберігаються в відповідних змінних, для того щоб забезпечити сортування в прямому та зворотному алфавітних порядках, а також збереженні сортування при оновленні таблиці.

Інформація про служби

При отриманні списку служб в функції DLL разом з назвою кожної служби додатково повертаються стан та спосіб запуску, які в подальшому відображаються в окремих стовпцях таблиці, що дозволяє користувачу наглядно бачити інформацію про служби.

3.3. Реалізація функцій керування службами Windows, їх призначення

1. **Функція** const char** EnumerateServicesWithInfo()

Призначення: отримує список усіх сервісів Windows, повертає їх в форматі «назва,стан,тип_запуску».

2. **Функція** int StartServiceC(const char* serviceName)

Призначення: запускає сервіс з назвою переданою в параметрі, повертає цілочисельне значення яке позначає успішність виконання операції.

3. **Функція** int StopService(const char* serviceName)

Призначення: зупиняє сервіс з назвою переданою в параметрі, повертає цілочисельне значення яке позначає успішність виконання операції.

4. **Функція** int PauseService (const char* serviceName)

Призначення: призупиняє сервіс з назвою переданою в параметрі, повертає цілочисельне значення яке позначає успішність виконання операції.

5. **Функція** int ContinueService (const char* serviceName)

Призначення: продовжує роботу сервісу з назвою переданою в параметрі, повертає цілочисельне значення яке позначає успішність виконання операції.

6. **Функція** int ChangeStartType(const char* serviceName, DWORD startType)

Призначення: змінює тип запуску сервісу, назва сервісу та новий тип запуску передаються через параметри, повертає цілочисельне значення яке позначає успішність виконання операції.

3.4. Реалізація функцій інтерфейсу для взаємодії з користувачами

Клас WindowsServicesManager

Призначення: Клас для створення графічного інтерфейсу користувача.

Функції:

- setup_dll(): підключає C++ DLL з функціями керування сервісами Windows, визначає типи значень аргументів та параметрів які повертають функції бібліотеки.
- setup_gui(): створює вікно програми, ініціює виклики інших функцій для створення та заповнення віджетами фреймів.
- setup_buttons(): додає в вікно кнопки виконання дій над обраним сервісом.
- load_icons(): завантажує та підготовлює для використання в якості іконок відповідних файлів зображень.
- setup_search_entry(): додає в вікно поле для вводу назви служби, кнопку здійснення пошуку.
- setup_services_tree(): додає в вікно таблицю зі службами, зі смугою прокрутки праворуч, оголошує змінні для збереження обраного сортування.
- sort_treeview_column(col, reverse): здійснює обране сортування таблиці.
- setup_status_label(): додає в низ вікна рядок стану.
- update_buttons_state(): змінює активність кнопок дій, залежно від стану обраної служби.
- start_service(): викликає в DLL запуск для обраної служби.
- stop_service(): викликає в DLL зупинку для обраної служби.
- restart_service(): здійснює перезапуск для обраної служби.
- pause_service(): викликає в DLL запуск для обраної служби.
- continue_service(): викликає в DLL запуск для обраної служби.
- change_start_type(service_name, start_type): викликає в DLL зміну типу запуску для обраної служби.
- update_services_tree(): оновлює таблицю служб.
- on_change_start_type_button_click(): викликає створення вікна вибору типу запуску служби.
- show_start_type_selection_dialog(service_name, current_start_type): створює вікно вибору типу запуску служби.
- get_services(): отримує з DLL актуальний список служб.
- wait_until_service_state(service_name, state): чекає поки служба набуде очікуваного стану, після чого викликає оновлення таблиці.
- run(): запускає цикл обробки подій графічного інтерфейсу.

3.5. Опис алгоритму роботи графічного інтерфейсу

Першим кроком, програма підключає всі необхідні для її роботи модулі: tkinter, ctypes, os.

Далі, створюється об’єкт класу, який безпосередньо відповідає за графічний інтерфейс програми. Конструктор даного об’єкту ініціює дві основних масштабних дії:

1. Налагодження роботи з DLL. Створюється об’єкт ctypes.CDLL для DLL розташованого за заздалегідь визначеним шляхом. Задаються очікувані типи аргументів та результатів для усіх необхідних функцій DLL. Це надає можливість в подальшому викликати дані функції з Python коду.
2. Налагодження GUI. За допомогою tkinter, власне, створюється основне вікно програми, а також фрейми в ньому, які заповнюються відповідними віджетами: кнопками дій, полем та кнопкою для пошуку, таблицею служб, текстовим рядком який виконує роль рядку стану. Перед створенням кнопок завантажуються відповідні піктограми. Після створення віджету таблиці служб, ініціюється її заповнення отриманими з DLL службами.

Після того як робота з DLL та GUI налагоджені, запускається цикл обробки подій графічного інтерфейсу, тобто програма очікує на виконання дій користувачем, та обробляє їх, допоки її процес запущений.

4. ТЕСТУВАННЯ

Для того щоб переконатись в успішності розробки програми "Windows Services Manager" необхідно перевірити чи вона відповідає усім розробленим під час проектування вимогам, для цього протестуємо усі функції програми.

1. Перегляд доступних служб

Як можна побачити з рис. 4.1 функція перегляду доступних служб дійсно відображає повний список служб Windows, розмістивши їх в таблиці з інформацією про них в відповідних стовпцях, праворуч від таблиці бачимо смугу прокрутки таблиці. Також можемо переконатись в можливості обрати конкретну службу для виконання над нею подальших дій.

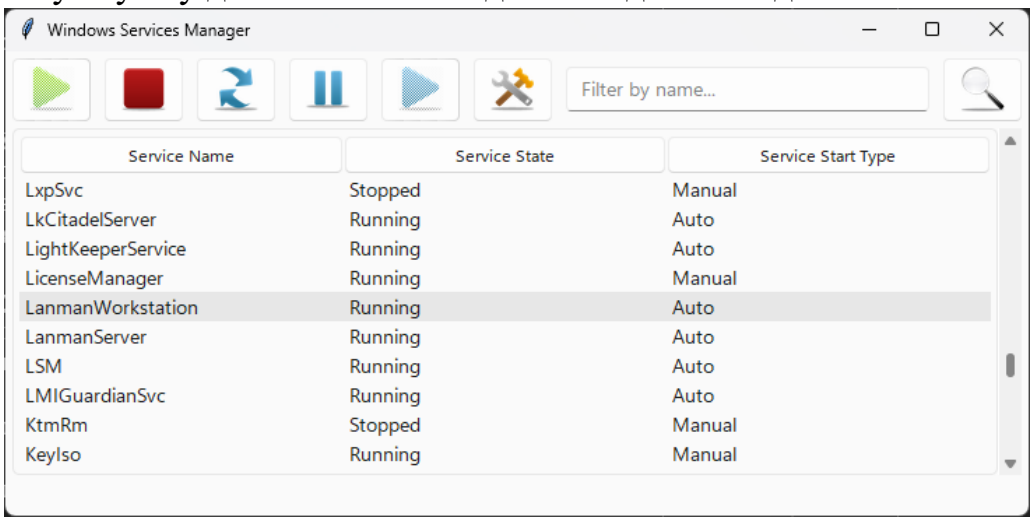


Рис.4.1. Перегляд доступних служб №1.

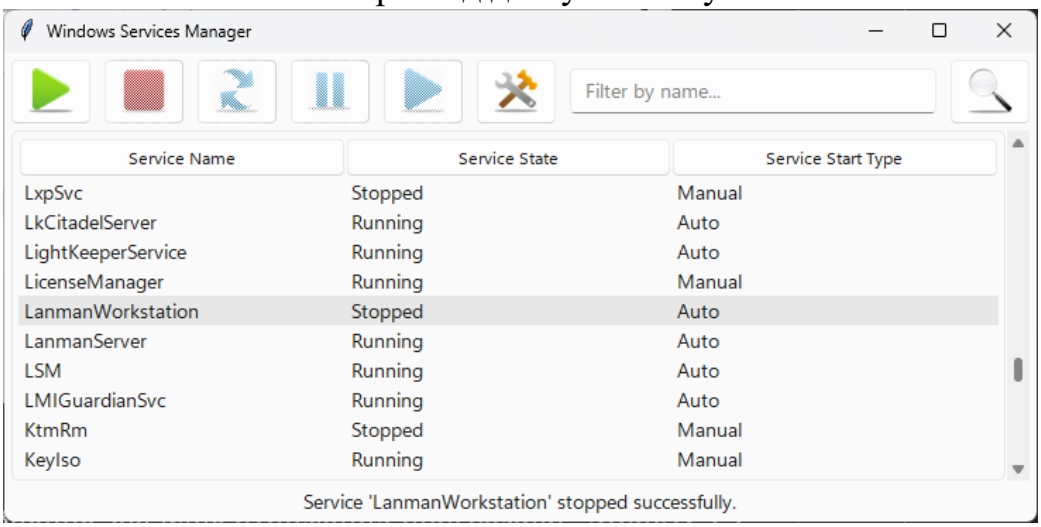


Рис.4.2. Перегляд доступних служб №2.

Для того щоб переконатись в тому що програма задовольняє решту вимог даної функції, виконаємо дію над обраною службою (рис. 4.2), а саме її зупинку. Цим ми остаточно переконались в можливості виконання дій над обраною службою, а також, оскільки оновився стан сервісу, і в оновленні таблиці після виконання операції, відповідно й в збереженні вибору служби

при оновленні таблиці. І, нарешті, бачимо що змінилась активність кнопок дій, після виконання операції над службою.

2. Запуск, зупинка, перезапуск, призупинення, продовження роботи служб

З рис. 4.3-4.7 в інтерфейсі програми можна побачити наявність відповідних кнопок дій з підказками. В низу програми знаходиться рядок стану, який відображає повідомлення про успішність виконання дій. З рисунків видно, що програма показує поточний стан перед та після виконанням дії над обраною службою (стовпець «Service State»). Також, дані рисунки демонструють працездатність наявних кнопок, тобто підтверджуть можливість виконання дій над окремою службою з інтерфейсу користувача.

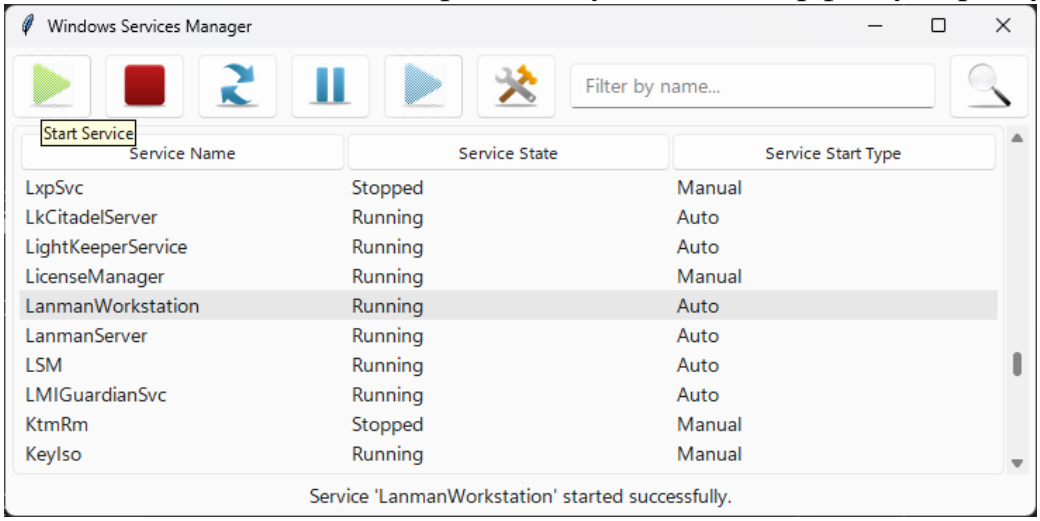


Рис.4.3. Програма після запуску служби.

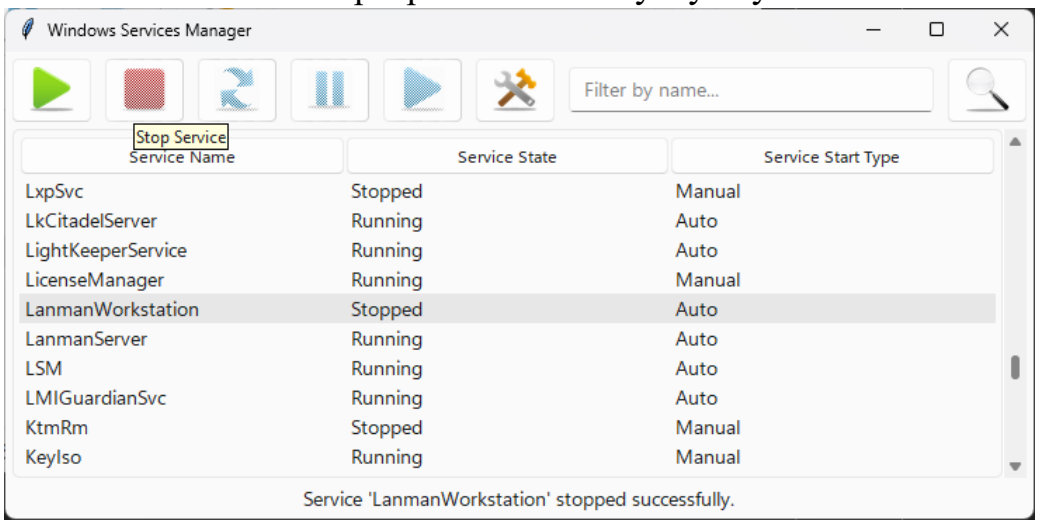


Рис.4.4. Програма після зупинки служби.

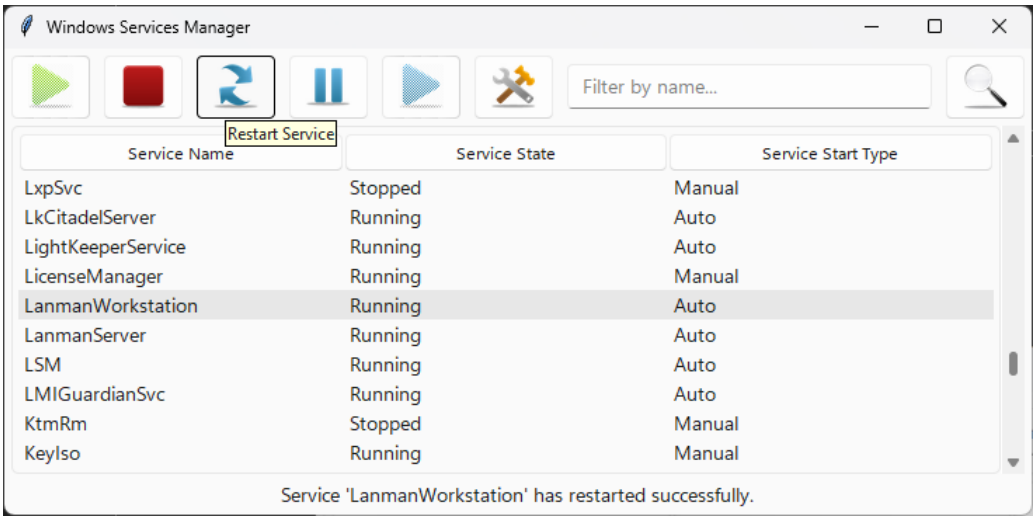


Рис.4.5. Програма після перезапуску служби.

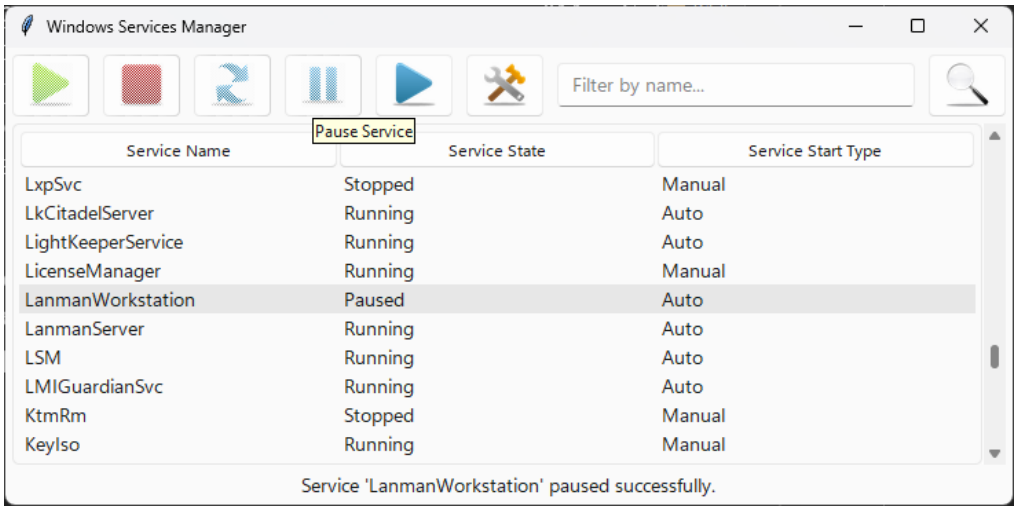


Рис.4.6. Програма після призупинення служби.

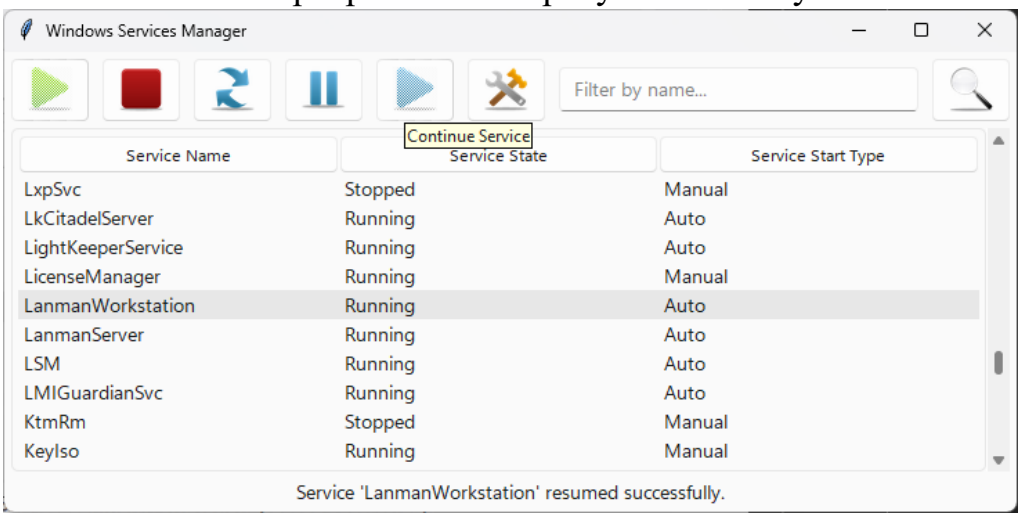


Рис.4.7. Програма після продовження робота служби.

3. Вибір способу запуску служби

Рис. 4.8 демонструє наявність в інтерфейсі програми кнопки з підказкою для зміни типу запуску обраної служби.

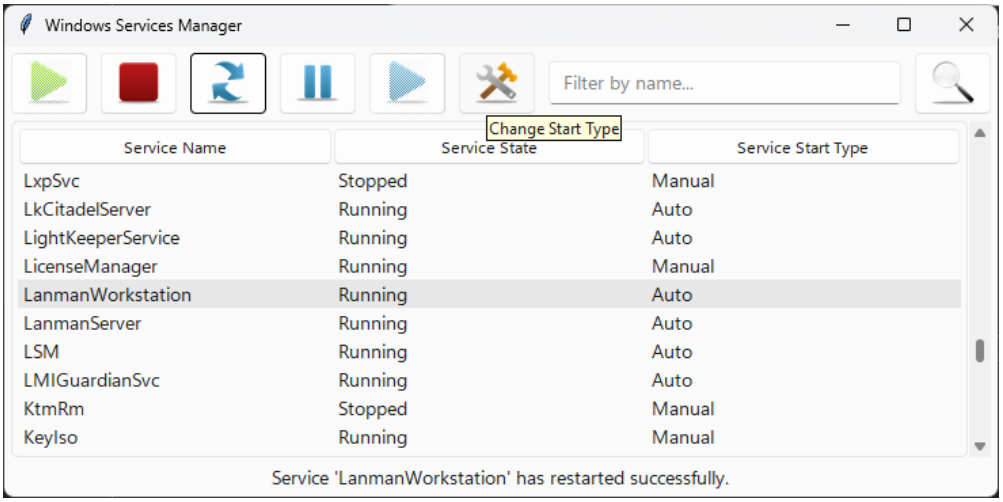


Рис.4.8. Кнопка зміни типу запуску служби.

Окреме вікно з радіокнопками для вибору нового типу запуску служби, можна побачити на рис. 4.9.

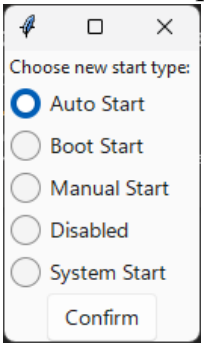


Рис.4.9. Вікно зміни типу запуску служби.

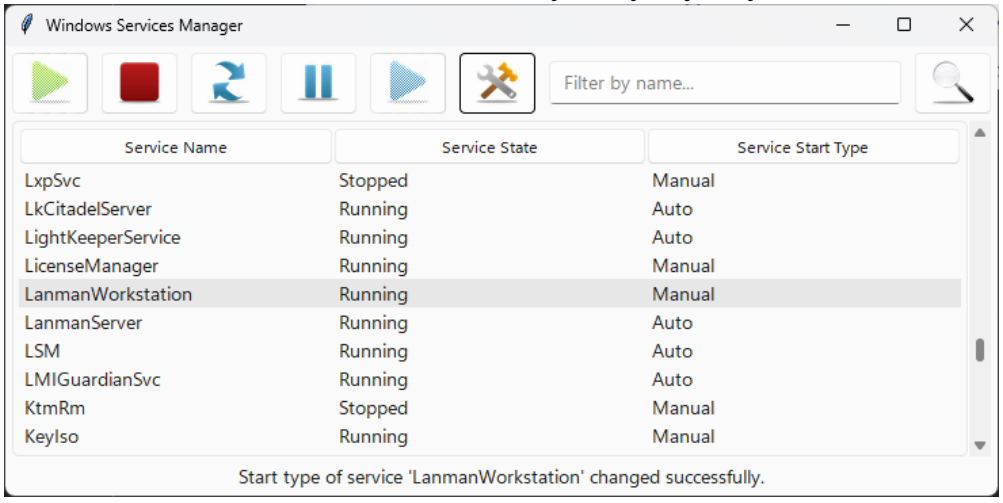


Рис.4.10. Програма після зміни типу запуску служби.

Задоволення вимог відображення поточного способу запуску служби перед та після його зміною, повідомлення в рядку стану про успішність або неуспішність виконання дії здійснює зображене на рис. 4.10. Рис. 4.8-4.10 доводять забезпечення програмою можливості зміни способу запуску окремих служб з інтерфейсу користувача.

4. Пошук служб

Присутність в інтерфейсі програми поля для введення пошукового запиту, наявність відповідної кнопки з підказкою, можемо бачити на рис. 4.11. Також, з цього рисунку видно, що після введення назви служби для пошуку, та натиснення відповідної кнопки, таблиця оновилась, та в ній знаходяться лише ті служби, назва яких містить літери введені в поле для вводу назви.

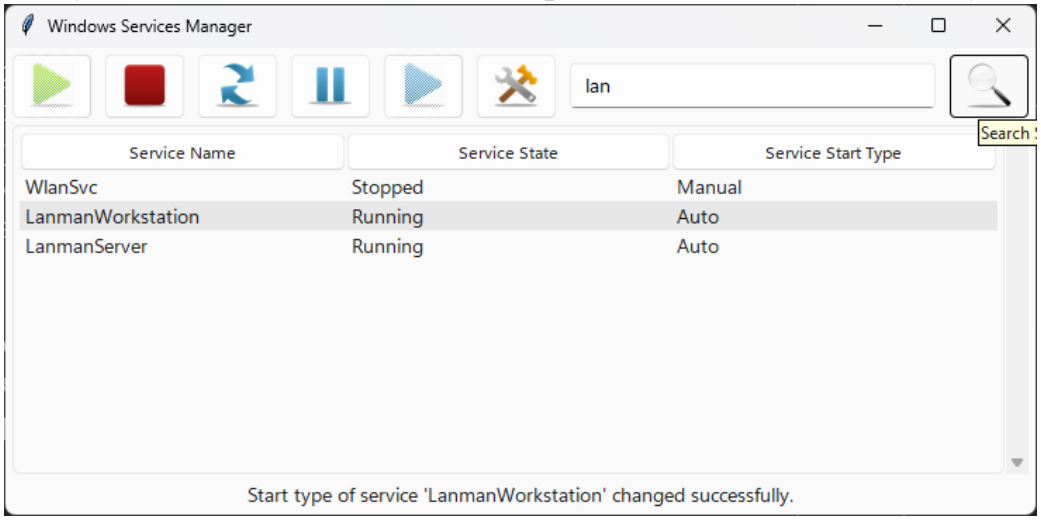


Рис.4.11. Демонстрація функціонування пошуку служб.

5. Сортування служб

Вимоги про реалізацію сортування служб за ім’ям (рис. 4.12, 4.13), станом (рис. 4.14, 4.15), типом запуску (рис. 4.16, 4.17), в прямому та зворотному алфавітних порядках, як можна побачити з відповідних рисунків, програма цілком задовільняє

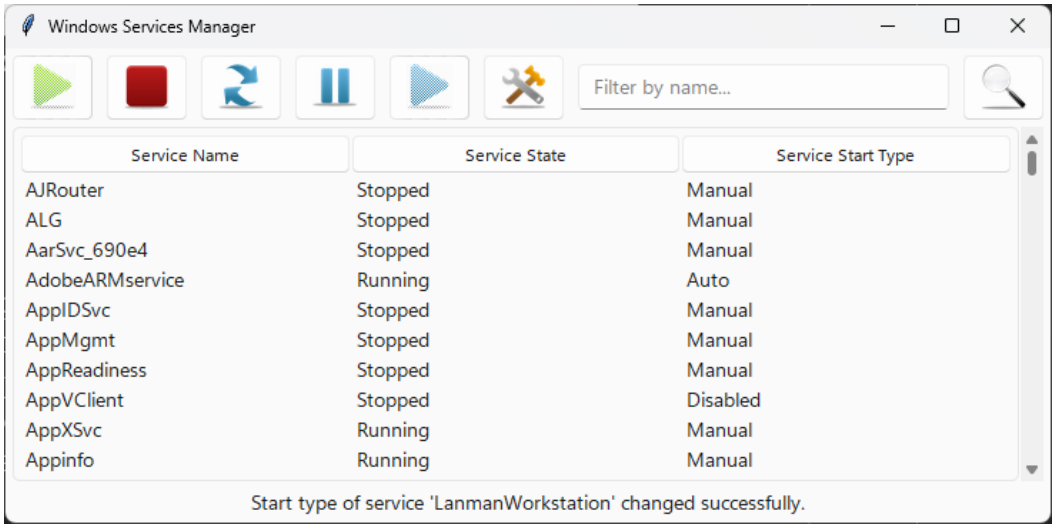


Рис.4.12. Сортування за назвою в алфавітному порядку.

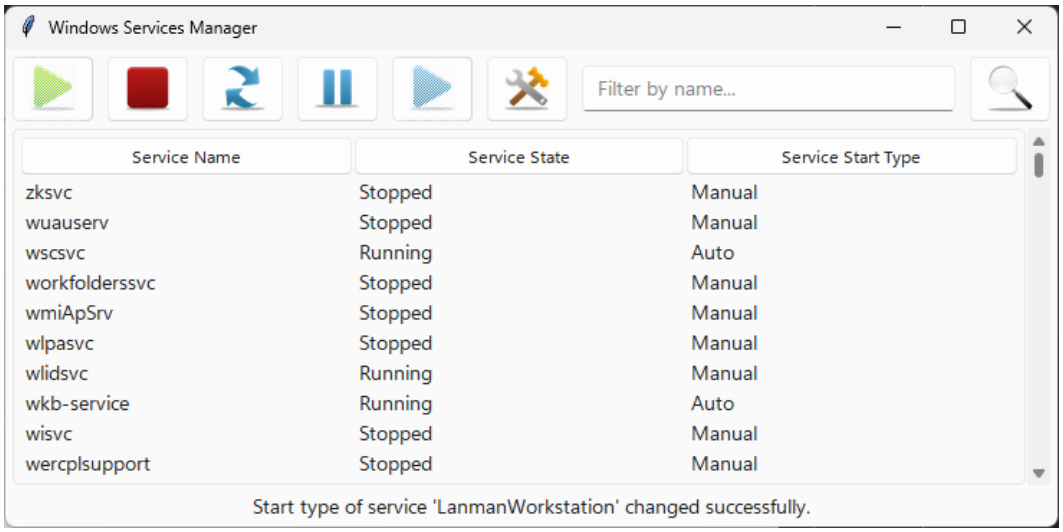


Рис.4.13. Сортування за назвою в зворотному алфавітному порядку.

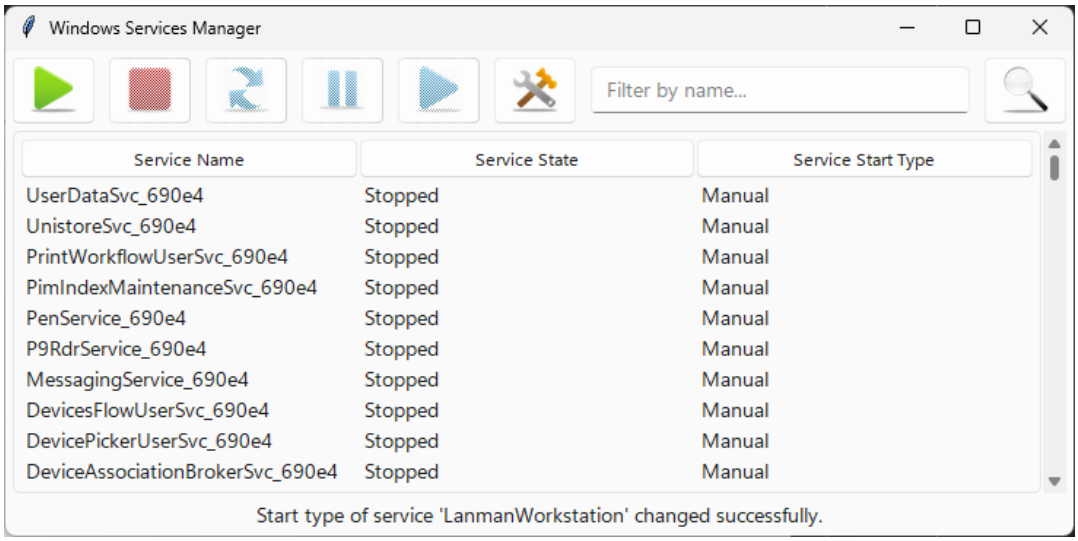


Рис.4.14. Сортування за станом в алфавітному порядку.

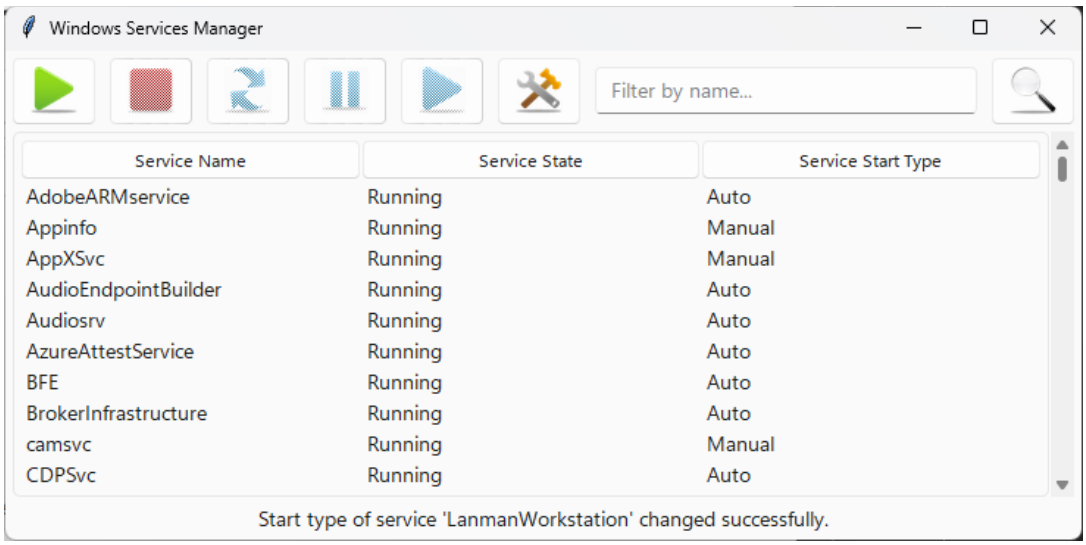


Рис.4.15. Сортування за станом в зворотному алфавітному порядку.

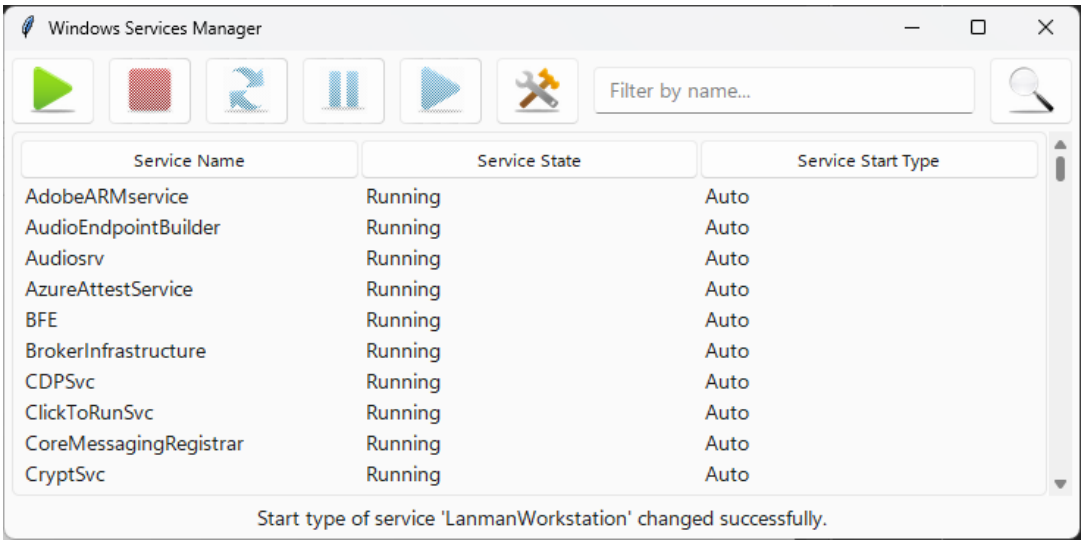


Рис.4.16. Сортування за типом запуску в алфавітному порядку.

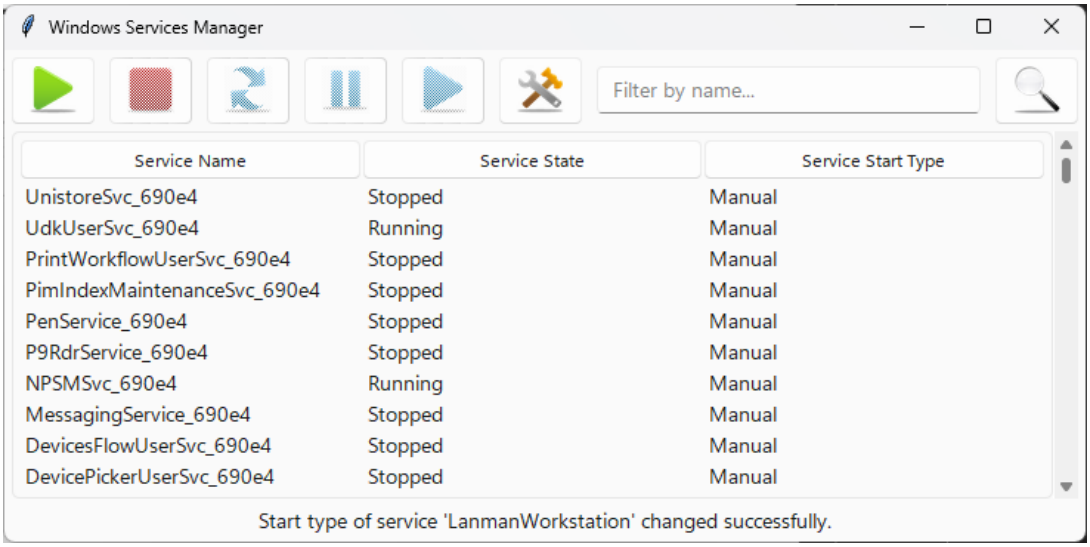


Рис.4.17. Сортування за типом запуску в зворотному алфавітному порядку.

Збереження вибраного сортування при оновленні таблиці демонструє рис. 4.18. Після здійснення пошуку відбулось оновлення таблиці, при цьому обране сортування за типом запуску в зворотному алфавітному порядку збережено.

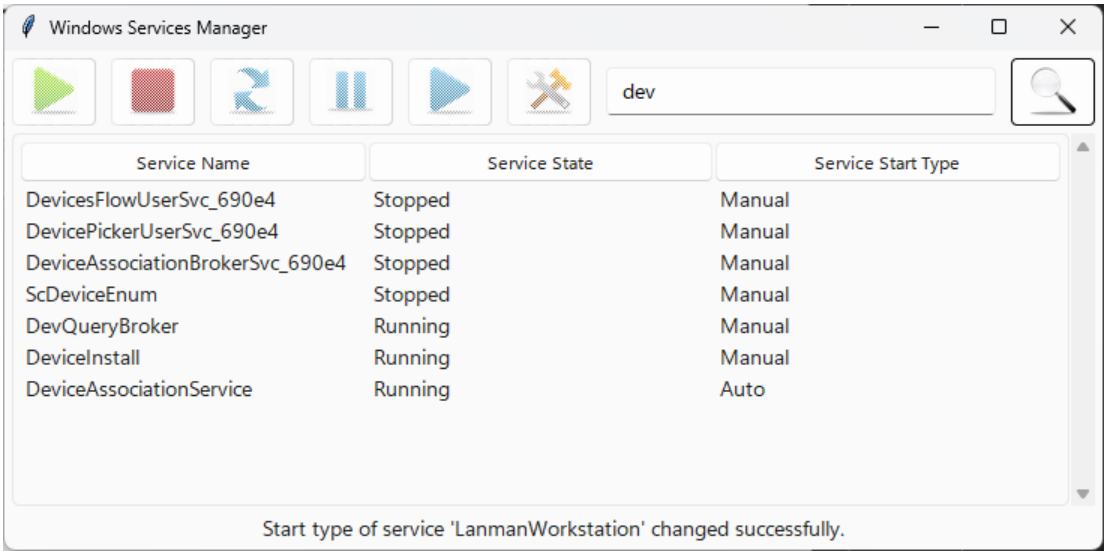


Рис.4.18. Збереження сортування при оновленні таблиці.

6. Інформація про служби

Як можна побачити на рис. 4.19, в програмі відображаються назви служб, їх поточний стан (“Stopped”, “Running”), спосіб запуску (“Manual”, “Disabled”), що задовольняє описані вимоги до даної функції.

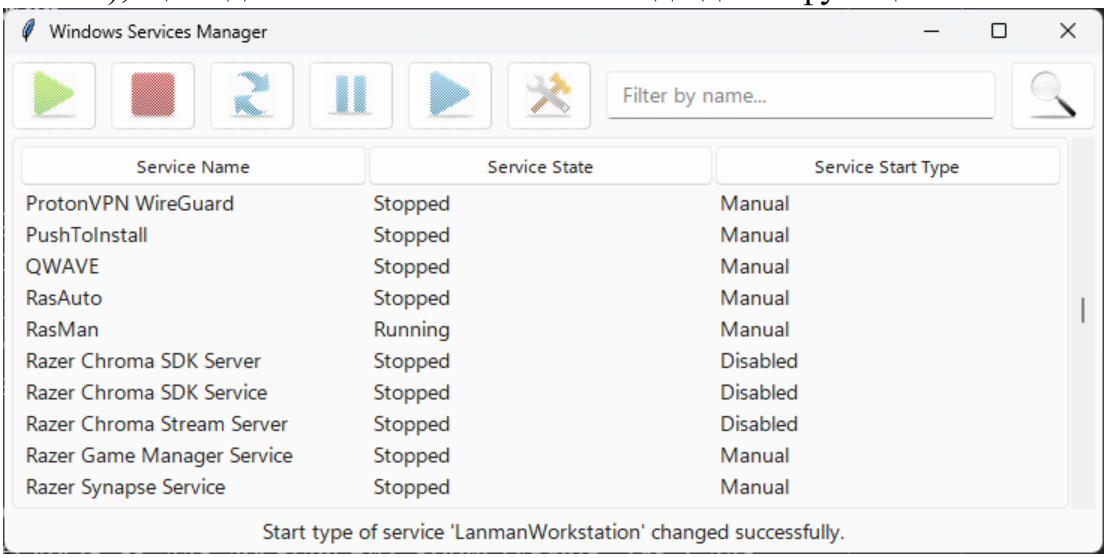


Рис.4.19. Відображення інформації про сервіси.

Висновки

З проведених детальних тестувань задоволення програмою усіх вимог до кожної з функцій, видно, що програма повністю задовольняє усі поставлені вимоги. Кожна функція працює коректно, та чітко. Реалізація програми цілком відповідає проектуванню.

ВИСНОВКИ

В цьому курсовому проекті було розроблено програму для зручного управління службами операційної системи Windows, яка надає інтуїтивно зрозумілий користувацький інтерфейс для зміни поточного стану та способу запуску служб. Програма написана на мові програмування Python та використовує бібліотеки tkinter для реалізації графічного інтерфейсу та ctypes для взаємодії з системними службами через функції бібліотеки написаної на C++ засобами WinAPI.

У результаті розробки програми було досягнуто наступних цілей:

1. Реалізовано функціонал для перегляду списку усіх доступних служб операційної системи Windows.
2. Забезпечено можливість запуску, зупинки, перезапуску, паузи, продовження роботи вибраної служби.
3. Забезпечено можливість вибору способу запуску служб (автоматичний, при запуску системи, вручну, вимкнена).
4. Реалізовано вбудований пошук для швидкого знаходження конкретних служб
5. Реалізовано сортування служб за ім'ям, станом та способом запуску.
6. Надано відображення інформації про кожну службу, включаючи назву, стан та спосіб запуску.

Отже, цей проект є повноцінним інструментом для управління службами Windows, що може бути корисним як для звичайних користувачів, так і для системних адміністраторів. Крім того, є можливість розширити функціонал програми та покращити її, додаючи нові опції і удосконалюючи користувацький інтерфейс.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Panek. (2020). Windows operating system fundamentals (1st edition). Sybex. – 419 p.
2. David A. Solomon (2017). “Windows Internals”.
3. Alan D Moore (2018). Python GUI Programming with Tkinter: Develop responsive and powerful GUI applications with Tkinter. Packt Publishing. – 452 pages.
4. ctypes — A foreign function library for Python [Електронний ресурс]. Режим доступу: docs.python.org/3/library/ctypes
5. Service control manager [Електронний ресурс]. Режим доступу: learn.microsoft.com/en-us/windows/win32/services/service-control-manager

6.

ДОДАТКИ

Додаток А - Оцінка виконання програми за допомогою User Story

User Story 1: Перегляд доступних служб
Оцінка часу виконання: 3 дні
Я, як користувач, хочу мати можливість переглядати список усіх доступних служб операційної системи Windows, щоб отримати загальне уявлення про всі служби, які працюють на моїй системі.
Користь для користувача: Забезпечення зручного візуального доступу до всіх доступних служб.
Критерії готовності:

- Реалізовані функції для збору інформації про доступні служби.
- Створений інтерфейс для відображення списку служб.
- Пройдено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

User Story 2: Запуск служб
Оцінка часу виконання: 2 дні
Я, як користувач, хочу мати можливість запускати вибрану службу, щоб активувати її роботу в разі необхідності.
Користь для користувача: Забезпечення можливості запуску служб для активації необхідних системних функцій та сервісів.
Критерії готовності:

- Реалізовані функції для запуску служб.
- Інтерфейс для вибору та запуску служб створений.
- Проведено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

User Story 3: Зупинка служб
Оцінка часу виконання: 1 день
Я, як користувач, хочу мати можливість зупиняти вибрану службу, щоб припинити її роботу при необхідності.
Користь для користувача: Забезпечення можливості зупинки служб для контролю за використанням системних ресурсів та припинення непотрібних процесів.
Критерії готовності:

- Реалізовані функції для зупинки служб.
- Інтерфейс для вибору та зупинки служб створений.
- Проведено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

User Story 4: Перезапуск служб

Оцінка часу виконання: 2 дні
Я, як користувач, хочу мати можливість перезапускати вибрану службу, щоб відновити її роботу без повного вимкнення системи.
Користь для користувача: Забезпечення можливості перезапуску служб для оновлення їх стану без повного вимкнення або перезавантаження системи.
Критерії готовності:

- Реалізовані функції для перезапуску служб.
- Інтерфейс для вибору та перезапуску служб створений.
- Проведено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

User Story 5: Призупинення служб
Оцінка часу виконання: 2 дні
Я, як користувач, хочу мати можливість призупиняти роботу вибраної служби, щоб тимчасово припинити її активність без повного зупинення.
Користь для користувача: Забезпечення можливості призупинення служб для тимчасового припинення їх роботи без повного вимкнення, що дозволяє швидко відновити їх активність при необхідності.
Критерії готовності:

- Реалізовані функції для призупинення служб.
- Інтерфейс для вибору та призупинення служб створений.
- Проведено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

User Story 6: Продовження роботи служб
Оцінка часу виконання: 1 день
Я, як користувач, хочу мати можливість продовжувати роботу вибраної призупиненої служби, щоб відновити її активність.
Користь для користувача: Забезпечення можливості продовження роботи призупинених служб для відновлення їх функціонування після тимчасової зупинки.
Критерії готовності:

- Реалізовані функції для продовження роботи служб.
- Інтерфейс для вибору та продовження роботи служб створений.
- Проведено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

User Story 7: Вибір способу запуску служби
Оцінка часу виконання: 3 дні
Я, як користувач, хочу мати можливість вибирати спосіб запуску служби (автоматичний, вручну, вимкнена), щоб керувати її поведінкою в моїй операційній системі.
Користь для користувача: Забезпечення можливості налаштування способу запуску служб для оптимізації процесу завантаження системи та керування ресурсами.

Критерії готовності:
- Реалізовані функції для вибору способу запуску служб.
- Інтерфейс для налаштування способу запуску служб створений.
- Проведено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

User Story 8: Пошук служб
Оцінка часу виконання: 2 дні
Я, як користувач, хочу мати можливість швидко знаходити конкретні служби за допомогою вбудованого пошуку, щоб ефективно керувати службами системи.
Користь для користувача: Забезпечення можливості швидкого знаходження конкретних служб для ефективного управління їхньою роботою.
Критерії готовності:
- Реалізовані функції для пошуку служб.
- Інтерфейс для пошуку служб створений.
- Проведено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

User Story 9: Сортування служб
Оцінка часу виконання: 2 дні
Я, як користувач, хочу мати можливість сортувати список служб за ім'ям, станом та способом запуску, щоб зручно керувати службами та знаходити потрібну інформацію.
Користь для користувача: Забезпечення можливості сортування служб для зручного перегляду та ефективного управління службами системи.
Критерії готовності:
- Реалізовані функції для сортування служб.
- Інтерфейс для сортування служб створений.
- Проведено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

User Story 10: Інформація про служби
Оцінка часу виконання: 1 день
Я, як користувач, хочу мати можливість переглядати інформацію про кожну службу, включаючи назву, стан та спосіб запуску, щоб краще розуміти їх функції та важливість.
Користь для користувача: Надання детальної інформації про кожну службу для кращого розуміння її ролі та стану.
Критерії готовності:
- Реалізовані функції для отримання інформації про служби.
- Інтерфейс для відображення інформації про кожну службу створений.
- Проведено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

Додаток Б - Вихідний код програми

wsm.py

```
import tkinter as tk
from tkinter import ttk
import ctypes
from ctypes import wintypes
import os
from PIL import Image, ImageTk
from idlelib.tooltip import Hovertip

import sv_ttk

class WindowsServiceManager:
    def __init__(self):
        self.setup_dll()
        self.setup_gui()

    def setup_dll(self):
        self.current_dir = os.path.dirname(os.path.abspath(__file__))
        self.dll_path = os.path.join(self.current_dir + "\\wsm_dll\\x64\\Debug\\", "wsm_dll.dll")
        self.wsm_dll = ctypes.CDLL(self.dll_path)

        self.wsm_dll.GetServicesCount.restype = wintypes.DWORD
        self.wsm_dll.EnumerateServicesWithInfo.restype = ctypes.POINTER(ctypes.c_char_p)
        self.wsm_dll.GetServiceInfo.argtypes = [ctypes.c_char_p]
        self.wsm_dll.GetServiceInfo.restype = ctypes.c_char_p
        self.wsm_dll.CanServiceBePaused.argtypes = [ctypes.c_char_p]
        self.wsm_dll.CanServiceBePaused.restype = ctypes.c_bool
        self.wsm_dll.CanServiceBeStopped.argtypes = [ctypes.c_char_p]
        self.wsm_dll.CanServiceBeStopped.restype = ctypes.c_bool

    def setup_gui(self):
        self.root = tk.Tk()
        self.root.title("Windows Services Manager")

        sv_ttk.set_theme("light")

        self.operations_frame = ttk.Frame(self.root)
        self.operations_frame.grid(row=0, column=0)

        self.icons = self.load_icons()
        self.setup_buttons()
        self.setup_search_entry()
        self.setup_services_tree()
        self.setup_status_label()

    def setup_buttons(self):
        self.start_button = ttk.Button(self.operations_frame, image=self.icons["start"],
command=self.start_service)
        self.start_button.grid(row=0, column=0, padx=5, pady=5)
        Hovertip(self.start_button, 'Start Service', hover_delay=500)

        self.stop_button = ttk.Button(self.operations_frame, image=self.icons["stop"],
command=self.stop_service)
        self.stop_button.grid(row=0, column=1, padx=5, pady=5)
        Hovertip(self.stop_button, 'Stop Service', hover_delay=500)

        self.restart_button = ttk.Button(self.operations_frame, image=self.icons["restart"],
command=self.restart_service)
        self.restart_button.grid(row=0, column=2, padx=5, pady=5)
        Hovertip(self.restart_button, 'Restart Service', hover_delay=500)

        self.pause_button = ttk.Button(self.operations_frame, image=self.icons["pause"],
command=self.pause_service)
        self.pause_button.grid(row=0, column=3, padx=5, pady=5)
        Hovertip(self.pause_button, 'Pause Service', hover_delay=500)
```

```
        self.continue_button = ttk.Button(self.operations_frame, image=self.icons["continue"],
command=self.continue_service)
        self.continue_button.grid(row=0, column=4, padx=5, pady=5)
        Hovertip(self.continue_button, 'Continue Service', hover_delay=500)

        self.change_start_type_button = ttk.Button(self.operations_frame,
image=self.icons["change_start_type"], command=self.on_change_start_type_button_click)
        self.change_start_type_button.grid(row=0, column=5, padx=5, pady=5)
        Hovertip(self.change_start_type_button, 'Change Start Type', hover_delay=500)

    def load_icons(self):
        icon_paths = {
            "start": "icons/start_icon.png",
            "stop": "icons/stop_icon.png",
            "restart": "icons/restart_icon.png",
            "pause": "icons/pause_icon.png",
            "continue": "icons/continue_icon.png",
            "change_start_type": "icons/change_start_type_icon.png",
            "search": "icons/search_icon.png"
        }
        icons = {}
        for name, path in icon_paths.items():
            image = Image.open(path).resize((32, 32), Image.LANCZOS)
            icons[name] = ImageTk.PhotoImage(image)
        return icons

    def setup_search_entry(self):
        self.search_entry = ttk.Entry(self.operations_frame, width=30)
        self.search_entry.grid(row=0, column=6, padx=5, pady=5)
        self.search_entry.insert(0, 'Filter by name...')
        self.search_entry.config(foreground='grey')
        self.search_entry.bind('<FocusIn>', self.on_search_entry_click)
        self.search_entry.bind('<FocusOut>', self.on_search_focusout)

        self.search_button = ttk.Button(self.operations_frame, image=self.icons["search"],
command=self.update_services_tree)
        self.search_button.grid(row=0, column=7, padx=5, pady=5)
        Hovertip(self.search_button, 'Search Service', hover_delay=500)

    def setup_services_tree(self):
        self.current_sort_column = None
        self.current_sort_reverse = False

        self.services_frame = ttk.Frame(self.root)
        self.services_frame.grid(row=1, column=0, sticky="nsew", padx=5)
        self.services_tree_scrollbar = tk.Scrollbar(self.services_frame)
        self.services_tree_scrollbar.pack(side="right", fill="y")

        self.services_tree = ttk.Treeview(self.services_frame, columns=('Service Name', 'Service
State', 'Service Start Type'), show='headings', yscrollcommand=self.services_tree_scrollbar.set)
        self.services_tree_scrollbar.config(command=self.services_tree.yview)

        self.services_tree.heading('Service Name', text='Service Name', command=lambda:
self.sort_treeview_column('Service Name', False))
        self.services_tree.heading('Service State', text='Service State', command=lambda:
self.sort_treeview_column('Service State', False))
        self.services_tree.heading('Service Start Type', text='Service Start Type',
command=lambda: self.sort_treeview_column('Service Start Type', False))
        self.services_tree.pack(fill='both', expand=True)

        self.services_tree.bind('<<TreeviewSelect>>', self.update_buttons_state)
        self.update_services_tree()

    def sort_treeview_column(self, col, reverse):
        self.current_sort_column = col
        self.current_sort_reverse = reverse
```

```
        data = [(self.services_tree.set(child, col), child) for child in
self.services_tree.get_children('')]
        data.sort(reverse=reverse)
        for index, (val, child) in enumerate(data):
            self.services_tree.move(child, '', index)
        self.services_tree.heading(col, command=lambda: self.sort_treeview_column(col, not
reverse))

    def setup_status_label(self):
        self.status_label = ttk.Label(self.root, text="", anchor='center')
        self.status_label.grid(row=2, column=0, sticky="ew", padx=5, pady=5)

    def on_search_entry_click(self, event):
        if self.search_entry.get() == "Filter by name...":
            self.search_entry.delete(0, "end")
            self.search_entry.insert(0, '')
            self.search_entry.config(foreground='black')

    def on_search_focusout(self, event):
        if self.search_entry.get() == '':
            self.search_entry.insert(0, 'Filter by name...')
            self.search_entry.config(foreground='grey')

    def update_buttons_state(self, event=None):
        selected_service = self.services_tree.selection()
        if selected_service:
            service_name = self.services_tree.item(selected_service, 'text')
            current_state = self.wsm_dll.GetServiceInfo(service_name.encode('utf-
8')).decode('utf-8').split(",")[0]

            if current_state == "Stopped":
                self.start_button.config(state=tk.NORMAL)
                self.stop_button.config(state=tk.DISABLED)
                self.restart_button.config(state=tk.DISABLED)
                self.pause_button.config(state=tk.DISABLED)
                self.continue_button.config(state=tk.DISABLED)
                self.change_start_type_button.config(state=tk.NORMAL)
            elif current_state == "Running":
                self.start_button.config(state=tk.DISABLED)
                state = tk.DISABLED
                if self.wsm_dll.CanServiceBeStopped(service_name.encode("utf-8")):
                    state = tk.NORMAL
                self.stop_button.config(state=state)
                self.restart_button.config(state=state)
                self.pause_button.config(state=tk.NORMAL if
self.wsm_dll.CanServiceBePaused(service_name.encode("utf-8")) else tk.DISABLED)
                self.continue_button.config(state=tk.DISABLED)
                self.change_start_type_button.config(state=tk.NORMAL)
            elif current_state == "Paused":
                self.start_button.config(state=tk.DISABLED)
                self.stop_button.config(state=tk.DISABLED)
                self.restart_button.config(state=tk.DISABLED)
                self.pause_button.config(state=tk.DISABLED)
                self.continue_button.config(state=tk.NORMAL)
                self.change_start_type_button.config(state=tk.NORMAL)
            elif current_state in ["Start Pending", "Stop Pending", "Continue Pending", "Pause
Pending"]:
                self.start_button.config(state=tk.DISABLED)
                self.stop_button.config(state=tk.DISABLED)
                self.restart_button.config(state=tk.DISABLED)
                self.pause_button.config(state=tk.DISABLED)
                self.continue_button.config(state=tk.DISABLED)
                self.change_start_type_button.config(state=tk.NORMAL)
            else:
                self.start_button.config(state=tk.DISABLED)
                self.stop_button.config(state=tk.DISABLED)
                self.restart_button.config(state=tk.DISABLED)
                self.pause_button.config(state=tk.DISABLED)
                self.continue_button.config(state=tk.DISABLED)
```

```
                self.change_start_type_button.config(state=tk.DISABLED)
        else:
            self.start_button.config(state=tk.DISABLED)
            self.stop_button.config(state=tk.DISABLED)
            self.restart_button.config(state=tk.DISABLED)
            self.pause_button.config(state=tk.DISABLED)
            self.continue_button.config(state=tk.DISABLED)
            self.change_start_type_button.config(state=tk.DISABLED)

    def start_service(self):
        selected_service = self.services_tree.selection()
        if selected_service:
            service_name = self.services_tree.item(selected_service, 'text')
            result = self.wsm_dll.StartServiceC(service_name.encode('utf-8'))
            if result == 0:
                self.status_label.config(text=f"Service '{service_name}' started successfully.")
                self.update_services_tree()
                self.wait_until_service_state(service_name, "Running")
            else:
                self.status_label.config(text=f"Failed to start service '{service_name}'.")

    def stop_service(self):
        selected_service = self.services_tree.selection()
        if selected_service:
            service_name = self.services_tree.item(selected_service, 'text')
            result = self.wsm_dll.StopService(service_name.encode('utf-8'))
            if result == 0:
                self.status_label.config(text=f"Service '{service_name}' stopped successfully.")
                self.update_services_tree()
                self.wait_until_service_state(service_name, "Stopped")
            else:
                self.status_label.config(text=f"Failed to stop service '{service_name}'.")

    def restart_service(self):
        selected_service = self.services_tree.selection()
        if selected_service:
            service_name = self.services_tree.item(selected_service, 'text')
            self.stop_service()
            self.wait_until_service_state(service_name, "Stopped(Restart)")

    def pause_service(self):
        selected_service = self.services_tree.selection()
        if selected_service:
            service_name = self.services_tree.item(selected_service, 'text')
            result = self.wsm_dll.PauseService(service_name.encode('utf-8'))
            if result == 0:
                self.status_label.config(text=f"Service '{service_name}' paused successfully.")
                self.update_services_tree()
            else:
                self.status_label.config(text=f"Failed to pause service '{service_name}'.")

    def continue_service(self):
        selected_service = self.services_tree.selection()
        if selected_service:
            service_name = self.services_tree.item(selected_service, 'text')
            result = self.wsm_dll.ContinueService(service_name.encode('utf-8'))
            if result == 0:
                self.status_label.config(text=f"Service '{service_name}' resumed successfully.")
                self.update_services_tree()
            else:
                self.status_label.config(text=f"Failed to resume service '{service_name}'.")

    def change_start_type(self, service_name, start_type):
        result = self.wsm_dll.ChangeStartType(service_name.encode('utf-8'), start_type)
        if result == 0:
            self.status_label.config(text=f"Start type of service '{service_name}' changed
successfully.")
            self.update_services_tree()
        else:
```

```
            self.status_label.config(text=f"Failed to change start type of service
'{service_name}'.")

    def update_services_tree(self):
        selected_service_name = None
        selected_service = self.services_tree.selection()
        if selected_service:
            selected_service_name = self.services_tree.item(selected_service, 'text')

        if self.services_tree.get_children():
            for item in self.services_tree.get_children():
                self.services_tree.delete(item)

        services = self.get_services()

        search_string = self.search_entry.get()
        if search_string != "Filter by name...":
            services = [service for service in services if search_string.lower() in
service.split(',')[0].lower()]

        for service in services:
            service_name, service_state, service_start_type = service.split(',')
            self.services_tree.insert('', 'end', text=service_name, values=(service_name,
service_state, service_start_type))

        if selected_service_name:
            for item in self.services_tree.get_children():
                if self.services_tree.item(item, 'text') == selected_service_name:
                    self.services_tree.selection_set(item)
                    self.services_tree.focus(item)
                    break
        if self.current_sort_column is not None and self.services_tree.get_children():
            self.sort_treeview_column(self.current_sort_column, self.current_sort_reverse)

        self.update_buttons_state()

    def on_change_start_type_button_click(self):
        selected_service = self.services_tree.selection()
        if selected_service:
            service_name = self.services_tree.item(selected_service, 'text')
            current_start_type = self.wsm_dll.GetServiceInfo(service_name.encode('utf-
8')).decode('utf-8').split(",")[1]
            self.show_start_type_selection_dialog(service_name, current_start_type)

    def show_start_type_selection_dialog(self, service_name, current_start_type):
        self.SERVICE_AUTO_START = 2
        self.SERVICE_BOOT_START = 0
        self.SERVICE_DEMAND_START = 3
        self.SERVICE_DISABLED = 4
        self.SERVICE_SYSTEM_START = 1

        self.start_type_mapping = {
            "Boot": self.SERVICE_BOOT_START,
            "System": self.SERVICE_SYSTEM_START,
            "Auto": self.SERVICE_AUTO_START,
            "Manual": self.SERVICE_DEMAND_START,
            "Disabled": self.SERVICE_DISABLED,
            "Unknown": 0
        }

        def confirm_and_change_start_type():
            new_start_type = selected_start_type.get()
            dialog.destroy()
            if new_start_type != self.start_type_mapping.get(current_start_type, 0):
                self.change_start_type(service_name, new_start_type)

        dialog = tk.Toplevel(self.root)
        dialog.title("Select Start Type")
```

```
        tk.Label(dialog, text="Choose new start type: ").pack()

        selected_start_type = tk.IntVar(value=self.start_type_mapping.get(current_start_type, 0))

        ttk.Radiobutton(dialog, text="Auto Start", variable=selected_start_type,
value=self.SERVICE_AUTO_START).pack(anchor=tk.W)
        ttk.Radiobutton(dialog, text="Boot Start", variable=selected_start_type,
value=self.SERVICE_BOOT_START).pack(anchor=tk.W)
        ttk.Radiobutton(dialog, text="Manual Start", variable=selected_start_type,
value=self.SERVICE_DEMAND_START).pack(anchor=tk.W)
        ttk.Radiobutton(dialog, text="Disabled", variable=selected_start_type,
value=self.SERVICE_DISABLED).pack(anchor=tk.W)
        ttk.Radiobutton(dialog, text="System Start", variable=selected_start_type,
value=self.SERVICE_SYSTEM_START).pack(anchor=tk.W)

        confirm_button = ttk.Button(dialog, text="Confirm",
command=confirm_and_change_start_type)
        confirm_button.pack()

        dialog.wait_window()

        return selected_start_type.get()

    def get_services(self):
        services_ptr = self.wsm_dll.EnumerateServicesWithInfo()
        services = []

        if self.wsm_dll.EnumerateServicesWithInfo():
            for i in range(self.wsm_dll.GetServicesCount()):
                services.append(services_ptr[i].decode("utf-8"))

        return services

    def wait_until_service_state(self, service_name, state):
        current_state = self.wsm_dll.GetServiceInfo(service_name.encode('utf-8')).decode('utf-
8').split(",")[0]

        if current_state == state:
            self.update_services_tree()
            return
        else:
            if state == "Running":
                if current_state == "Stopped":
                    self.update_services_tree()
                    self.status_label.config(text=f"Service '{service_name}' has stopped
unexpectedly.")
                    return
            elif state == "Stopped":
                if current_state == "Running":
                    self.update_services_tree()
                    self.status_label.config(text=f"Service '{service_name}' has started
unexpectedly.")
                    return
            elif state == "Stopped(Restart)":
                if current_state == "Stopped":
                    self.update_services_tree()
                    self.start_service()
                    self.update_services_tree()
                    self.status_label.config(text=f"Service '{service_name}' has restarted
successfully.")
                    return
            self.root.after(1000, self.wait_until_service_state, service_name, state)

    def run(self):
        self.root.mainloop()

if __name__ == "__main__":
    app = WindowsServiceManager()
    app.run()
```

servicecontrol.h

```
#pragma once
#include <windows.h>

extern "C" __declspec(dllexport) int StartServiceC(const char* serviceName);
extern "C" __declspec(dllexport) int StopService(const char* serviceName);
extern "C" __declspec(dllexport) int PauseService(const char* serviceName);
extern "C" __declspec(dllexport) int ContinueService(const char* serviceName);
extern "C" __declspec(dllexport) int ChangeStartType(const char* serviceName, DWORD startType);
extern "C" __declspec(dllexport) const char* GetServiceInfo(const char* serviceName);
extern "C" __declspec(dllexport) DWORD GetServicesCount();
extern "C" __declspec(dllexport) const char** EnumerateServicesWithInfo();
extern "C" __declspec(dllexport) BOOL CanServiceBePaused(const char* serviceName);
extern "C" __declspec(dllexport) BOOL CanServiceBeStopped(const char* serviceName);
```

servicecontrol.cpp

```
#include "pch.h"
#include "servicecontrol.h"
#include <windows.h>
#include <iostream>

wchar_t* ConvertToWideString(const char* str) {
    int len = MultiByteToWideChar(CP_ACP, 0, str, -1, NULL, 0);
    if (len == 0) {
        return NULL;
    }
    wchar_t* wideStr = new wchar_t[len];
    if (MultiByteToWideChar(CP_ACP, 0, str, -1, wideStr, len) == 0) {
        delete[] wideStr;
        return NULL;
    }
    return wideStr;
}

std::string GetServiceStateString(DWORD currentState) {
    std::string stateString;
    switch (currentState) {
    case SERVICE_STOPPED:
        stateString = "Stopped";
        break;
    case SERVICE_START_PENDING:
        stateString = "Start Pending";
        break;
    case SERVICE_STOP_PENDING:
        stateString = "Stop Pending";
        break;
    case SERVICE_RUNNING:
        stateString = "Running";
        break;
    case SERVICE_CONTINUE_PENDING:
        stateString = "Continue Pending";
        break;
    case SERVICE_PAUSE_PENDING:
        stateString = "Pause Pending";
        break;
    case SERVICE_PAUSED:
        stateString = "Paused";
        break;
    default:
        stateString = "Unknown";
        break;
    }
```

```
    return stateString;
}

std::string GetStartTypeString(DWORD startType) {
    std::string startTypeString;
    switch (startType) {
    case SERVICE_BOOT_START:
        startTypeString = "Boot";
        break;
    case SERVICE_SYSTEM_START:
        startTypeString = "System";
        break;
    case SERVICE_AUTO_START:
        startTypeString = "Auto";
        break;
    case SERVICE_DEMAND_START:
        startTypeString = "Manual";
        break;
    case SERVICE_DISABLED:
        startTypeString = "Disabled";
        break;
    default:
        startTypeString = "Unknown";
        break;
    }
    return startTypeString;
}

int PerformServiceControl(const char* serviceName, DWORD managerDesiredAccess, DWORD
serviceDesiredAccess, DWORD controlType) {
    SC_HANDLE scm = OpenSCManager(NULL, NULL, managerDesiredAccess);
    if (scm == NULL) {
        return -1; // Failed to open service control manager
    }

    wchar_t* wideServiceName = ConvertToWideString(serviceName);

    SC_HANDLE service = OpenService(scm, wideServiceName, serviceDesiredAccess);

    delete[] wideServiceName;

    if (service == NULL) {
        CloseServiceHandle(scm);
        return -2; // Failed to open service
    }

    SERVICE_STATUS serviceStatus;
    if (!ControlService(service, controlType, &serviceStatus)) {
        CloseServiceHandle(service);
        CloseServiceHandle(scm);
        return -3; // Failed to control service
    }

    CloseServiceHandle(service);
    CloseServiceHandle(scm);
    return 0; // Service controlled successfully
}

extern "C" __declspec(dllexport) int StartServiceC(const char* serviceName) {
    SC_HANDLE scm = OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);
    if (scm == NULL) {
        return -1; // Failed to open service control manager
    }

    wchar_t* wideServiceName = ConvertToWideString(serviceName);

    SC_HANDLE service = OpenService(scm, wideServiceName, SERVICE_START);

    delete[] wideServiceName;
```

```
    if (service == NULL) {
        CloseServiceHandle(scm);
        return -2; // Failed to open service
    }


    if (!StartService(service, 0, NULL)) {
        CloseServiceHandle(service);
        CloseServiceHandle(scm);
        return -3; // Failed to start service
    }

    CloseServiceHandle(service);
    CloseServiceHandle(scm);
    return 0; // Service started successfully
}

extern "C" __declspec(dllexport) int StopService(const char* serviceName) {
    return PerformServiceControl(serviceName, SC_MANAGER_ALL_ACCESS, SERVICE_STOP,
SERVICE_CONTROL_STOP);
}

extern "C" __declspec(dllexport) int PauseService(const char* serviceName) {
    return PerformServiceControl(serviceName, SC_MANAGER_ALL_ACCESS, SERVICE_PAUSE_CONTINUE,
SERVICE_CONTROL_PAUSE);
}

extern "C" __declspec(dllexport) int ContinueService(const char* serviceName) {
    return PerformServiceControl(serviceName, SC_MANAGER_ALL_ACCESS, SERVICE_PAUSE_CONTINUE,
SERVICE_CONTROL_CONTINUE);
}

extern "C" __declspec(dllexport) int ChangeStartType(const char* serviceName, DWORD startType) {
    SC_HANDLE scm = OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);
    if (scm == NULL) {
        return -1; // Failed to open service control manager
    }

    wchar_t* wideServiceName = ConvertToWideString(serviceName);

    SC_HANDLE service = OpenService(scm, wideServiceName, SERVICE_CHANGE_CONFIG);

    delete[] wideServiceName;

    if (service == NULL) {
        CloseServiceHandle(scm);
        return -2; // Failed to open service
    }

    if (!ChangeServiceConfig(service, SERVICE_NO_CHANGE, startType, SERVICE_NO_CHANGE, NULL,
NULL, NULL, NULL, NULL, NULL, NULL)) {
        CloseServiceHandle(service);
        CloseServiceHandle(scm);
        return -3; // Failed to change service start type
    }

    CloseServiceHandle(service);
    CloseServiceHandle(scm);
    return 0; // Service start type changed successfully
}

extern "C" __declspec(dllexport) const char* GetServiceInfo(const char* serviceName) {
    SC_HANDLE scm = OpenSCManager(NULL, NULL, SC_MANAGER_CONNECT);
    if (scm == NULL) {
        return "Failed to open service control manager.";
    }

    wchar_t* wideServiceName = ConvertToWideString(serviceName);
```

```
    SC_HANDLE service = OpenService(scm, wideServiceName, SERVICE_QUERY_STATUS |
SERVICE_QUERY_CONFIG);

    delete[] wideServiceName;

    if (service == NULL) {
        CloseServiceHandle(scm);
        return "Failed to open service.";
    }

    SERVICE_STATUS_PROCESS serviceStatus;
    DWORD bytesNeeded;
    if (!QueryServiceStatusEx(service, SC_STATUS_PROCESS_INFO, (LPBYTE)&serviceStatus,
sizeof(SERVICE_STATUS_PROCESS), &bytesNeeded)) {
        CloseServiceHandle(service);
        CloseServiceHandle(scm);
        return "Failed to query service status.";
    }

    LPQUERY_SERVICE_CONFIG serviceConfig = NULL;
    DWORD bufferSize = 0;
    if (!QueryServiceConfig(service, NULL, 0, &bufferSize) && GetLastError() ==
ERROR_INSUFFICIENT_BUFFER) {
        serviceConfig = (LPQUERY_SERVICE_CONFIG)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY,
bufferSize);
        if (serviceConfig == NULL) {
            CloseServiceHandle(service);
            CloseServiceHandle(scm);
            return "Failed to allocate memory.";
        }

        if (!QueryServiceConfig(service, serviceConfig, bufferSize, &bytesNeeded)) {
            HeapFree(GetProcessHeap(), 0, serviceConfig);
            CloseServiceHandle(service);
            CloseServiceHandle(scm);
            return "Failed to query service configuration.";
        }
    }

    std::string serviceInfo;

    serviceInfo += GetServiceStateString(serviceStatus.dwCurrentState);
    serviceInfo += ",";

    if (serviceConfig != NULL) {
        serviceInfo += GetStartTypeString(serviceConfig->dwStartType);

        HeapFree(GetProcessHeap(), 0, serviceConfig);
    }

    CloseServiceHandle(service);
    CloseServiceHandle(scm);

    // Convert std::string to const char* for return
    char* result = new char[serviceInfo.length() + 1];
    strcpy_s(result, serviceInfo.length() + 1, serviceInfo.c_str());

    return result;
}

extern "C" __declspec(dllexport) DWORD GetServicesCount() {
    SC_HANDLE scm = OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);
    if (scm == NULL) {
        std::cerr << "Failed to open service control manager." << std::endl;
        return 0;
    }

    DWORD bytesNeeded, servicesCount, resumeHandle = 0;
```

```
    EnumServicesStatus(scm, SERVICE_WIN32, SERVICE_STATE_ALL, NULL, 0, &bytesNeeded,
&servicesCount, &resumeHandle);

    if (GetLastError() != ERROR_MORE_DATA) {
        std::cerr << "Failed to enumerate services." << std::endl;
        CloseServiceHandle(scm);
        return 0;
    }

    LPENUM_SERVICE_STATUS services = (LPENUM_SERVICE_STATUS)HeapAlloc(GetProcessHeap(),
HEAP_ZERO_MEMORY, bytesNeeded);
    if (!EnumServicesStatus(scm, SERVICE_WIN32, SERVICE_STATE_ALL, services, bytesNeeded,
&bytesNeeded, &servicesCount, &resumeHandle)) {
        std::cerr << "Failed to enumerate services." << std::endl;
        CloseServiceHandle(scm);
        HeapFree(GetProcessHeap(), 0, services);
        return 0;
    }
    CloseServiceHandle(scm);
    return servicesCount;
}

extern "C" __declspec(dllexport) const char** EnumerateServicesWithInfo() {
    SC_HANDLE scm = OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);
    if (scm == NULL) {
        std::cerr << "Failed to open service control manager." << std::endl;
        return nullptr;
    }

    DWORD bytesNeeded, servicesCount, resumeHandle = 0;
    EnumServicesStatus(scm, SERVICE_WIN32, SERVICE_STATE_ALL, NULL, 0, &bytesNeeded,
&servicesCount, &resumeHandle);

    if (GetLastError() != ERROR_MORE_DATA) {
        std::cerr << "Failed to enumerate services." << std::endl;
        CloseServiceHandle(scm);
        return nullptr;
    }

    LPENUM_SERVICE_STATUS services = (LPENUM_SERVICE_STATUS)HeapAlloc(GetProcessHeap(),
HEAP_ZERO_MEMORY, bytesNeeded);
    if (!EnumServicesStatus(scm, SERVICE_WIN32, SERVICE_STATE_ALL, services, bytesNeeded,
&bytesNeeded, &servicesCount, &resumeHandle)) {
        std::cerr << "Failed to enumerate services." << std::endl;
        CloseServiceHandle(scm);
        HeapFree(GetProcessHeap(), 0, services);
        return nullptr;
    }

    // Allocate memory for array of char pointers
    const char** servicesInfo = new const char* [servicesCount];
    for (DWORD i = 0; i < servicesCount; i++) {
        SC_HANDLE service = OpenService(scm, services[i].lpServiceName, SERVICE_QUERY_CONFIG);
        if (service == NULL) {
            std::cerr << "Failed to open service." << std::endl;
            continue;
        }

        LPQUERY_SERVICE_CONFIG serviceConfig = NULL;
        DWORD bufferSize = 0;
        if (!QueryServiceConfig(service, NULL, 0, &bufferSize) && GetLastError() ==
ERROR_INSUFFICIENT_BUFFER) {
            serviceConfig = (LPQUERY_SERVICE_CONFIG)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY,
bufferSize);
            if (serviceConfig == NULL) {
                std::cerr << "Failed to allocate memory." << std::endl;
                CloseServiceHandle(service);
                continue;
            }
```

```
            if (!QueryServiceConfig(service, serviceConfig, bufferSize, &bytesNeeded)) {
                std::cerr << "Failed to query service configuration." << std::endl;
                HeapFree(GetProcessHeap(), 0, serviceConfig);
                CloseServiceHandle(service);
                continue;
            }
        }

        int name_len = WideCharToMultiByte(CP_ACP, 0, services[i].lpServiceName, -1, NULL, 0,
NULL, NULL);
        char* serviceNameBuf = new char[name_len];
        WideCharToMultiByte(CP_ACP, 0, services[i].lpServiceName, -1, serviceNameBuf, name_len,
NULL, NULL);

        std::string serviceInfo = std::string(serviceNameBuf) + ",";

        delete[] serviceNameBuf;

        serviceInfo += GetServiceStateString(services[i].ServiceStatus.dwCurrentState);
        serviceInfo += ",";

        if (serviceConfig != NULL) {
            serviceInfo += GetStartTypeString(serviceConfig->dwStartType);

            HeapFree(GetProcessHeap(), 0, serviceConfig);
        }

        // Allocate memory for each service info and copy it
        size_t len = serviceInfo.length();
        servicesInfo[i] = new char[len + 1];
        strcpy_s(const_cast<char*>(servicesInfo[i]), len + 1, serviceInfo.c_str());

        CloseServiceHandle(service);
    }

    CloseServiceHandle(scm);
    HeapFree(GetProcessHeap(), 0, services);

    // Return pointer to array of service info
    return servicesInfo;
}

extern "C" __declspec(dllexport) BOOL CanServiceBePaused(const char* serviceName) {
    SC_HANDLE scm = OpenSCManager(NULL, NULL, SC_MANAGER_CONNECT);
    if (scm == NULL) {
        std::cerr << "Failed to open service control manager." << std::endl;
        return FALSE;
    }

    wchar_t* wideServiceName = ConvertToWideString(serviceName);

    SC_HANDLE service = OpenService(scm, wideServiceName, SERVICE_QUERY_STATUS);

    delete[] wideServiceName;

    if (service == NULL) {
        std::cerr << "Failed to open service." << std::endl;
        CloseServiceHandle(scm);
        return FALSE;
    }

    SERVICE_STATUS_PROCESS statusProcess;
    DWORD bytesNeeded;
    BOOL success = QueryServiceStatusEx(service, SC_STATUS_PROCESS_INFO, (LPBYTE)&statusProcess,
sizeof(SERVICE_STATUS_PROCESS), &bytesNeeded);

    if (!success) {
        std::cerr << "Failed to query service status." << std::endl;
```

```
    }
    else {
        DWORD controlsAccepted = statusProcess.dwControlsAccepted;
        BOOL canBePaused = (controlsAccepted & SERVICE_ACCEPT_PAUSE_CONTINUE) != 0;
        CloseServiceHandle(service);
        CloseServiceHandle(scm);
        return canBePaused;
    }

    CloseServiceHandle(service);
    CloseServiceHandle(scm);
    return FALSE;
}

extern "C" __declspec(dllexport) BOOL CanServiceBeStopped(const char* serviceName) {
    SC_HANDLE scm = OpenSCManager(NULL, NULL, SC_MANAGER_CONNECT);
    if (scm == NULL) {
        std::cerr << "Failed to open service control manager." << std::endl;
        return FALSE;
    }

    wchar_t* wideServiceName = ConvertToWideString(serviceName);

    SC_HANDLE service = OpenService(scm, wideServiceName, SERVICE_QUERY_STATUS);

    delete[] wideServiceName;

    if (service == NULL) {
        std::cerr << "Failed to open service." << std::endl;
        CloseServiceHandle(scm);
        return FALSE;
    }

    SERVICE_STATUS_PROCESS statusProcess;
    DWORD bytesNeeded;
    BOOL success = QueryServiceStatusEx(service, SC_STATUS_PROCESS_INFO, (LPBYTE)&statusProcess,
sizeof(SERVICE_STATUS_PROCESS), &bytesNeeded);

    if (!success) {
        std::cerr << "Failed to query service status." << std::endl;
    }
    else {
        DWORD controlsAccepted = statusProcess.dwControlsAccepted;
        BOOL canBeStopped = (controlsAccepted & SERVICE_ACCEPT_STOP) != 0;
        CloseServiceHandle(service);
        CloseServiceHandle(scm);
        return canBeStopped;
    }

    CloseServiceHandle(service);
    CloseServiceHandle(scm);
    return FALSE;
}
```