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.

Студента 3-го курсу групи

КІ-39

123 «Комп’ютерна інженерія»

Шеін І.А.

Керівник

Олексів М.В.

Національна шкала: _____

Кількість балів: _____

Оцінка ECTS: _____

Члени комісії: _____ _____

_____ _____

_____ _____

Львів – 2023

АНОТАЦІЯ

Ця курсова робота присвячена розробці програмного забезпечення для управління службами Windows. Основна мета роботи полягає у створенні програми, яка забезпечує можливість керування різними службами Windows, зокрема їх запуску, зупинки та перезапуску.

У роботі будуть розглянуті основні поняття, пов'язані з управлінням службами Windows. Для розробки програми будуть використані сучасні технології програмування, зокрема мова програмування C# та фреймворк .NET. Буде описана архітектура програми, включаючи інтерфейс користувача.

У заключній частині роботи будуть проведені тестування та аналіз результатів. Крім того, будуть розглянуті можливості подальшого розвитку та вдосконалення програми.

Отже, ця курсова робота має на меті розробити ефективну програму для управління службами Windows, яка буде забезпечувати користувачам швидкий та зручний доступ до необхідних функцій управління.

ЗМІСТ

ВСТУП

Управління службами Windows є важливим аспектом управління комп'ютерними системами, який дозволяє користувачам контролювати та керувати різноманітними функціями та процесами. Однак, стандартні засоби управління службами у Windows не завжди задовольняють потреби користувачів та можуть бути неефективними. Для вирішення цих проблем користувачі можуть використовувати сторонні програми для управління службами, які зазвичай пропонують розширені можливості та інтерфейси. Наприклад, деякі програми дозволяють зупиняти та запускати служби з одного місця, налаштовувати параметри автозапуску та перевіряти статус служб. Крім того, такі програми можуть надавати інформацію про використання системних ресурсів та відображати подробиці про кожну службу. Також користувачі можуть здійснювати дистанційне управління службами на інших комп'ютерах, а також виконувати більш складні завдання, такі як групове керування службами. Деякі програми також надають можливість налаштовувати права доступу до служб та контролювати, які користувачі можуть взаємодіяти з певними службами. В цілому, використання сторонніх програм для управління службами може значно полегшити та прискорити роботу з комп'ютерними системами на базі Windows.

У цій курсовій роботі буде розглянуто розробку програмного забезпечення для управління службами Windows. Основною метою є створення програми, яка забезпечить можливість керування різними службами Windows, зокрема їх запуску, зупинки та перезапуску. Для досягнення цієї мети будуть використані сучасні технології програмування, зокрема мова програмування C# та фреймворк .NET. В роботі будуть розглянуті основні поняття, пов'язані з управлінням службами Windows. Буде описана архітектура програми, включаючи інтерфейс користувача. У заключній частині роботи будуть проведені тестування та аналіз результатів. Крім того, будуть розглянуті можливості подальшого розвитку та вдосконалення програми.

Метою цієї курсової роботи є розробка ефективної програми для управління службами Windows, яка буде забезпечувати користувачам швидкий та зручний доступ до необхідних функцій управління. Розробка такої програми може бути корисною для інформаційних технологій, системного адміністрування та інших галузей, які використовують операційну систему Windows.

1.КОРОТКИЙ ОПИС, ОГЛЯД МЕТОДІВ, ІСНУЮЧИХ РОЗРОБОК У ДАНОМУ ПИТАННІ

Розберемо деякі програми, що дозволяють керувати службами Windows:

- "Служби Windows" (Services) - це стандартні засоби управління службами у Windows, доступні через Панель керування. За допомогою "Служб Windows" користувачі можуть зупиняти, запускати та перезапускати служби, налаштовувати параметри автозапуску, переглядати статус та опис служби, а також налаштовувати залежності між службами.

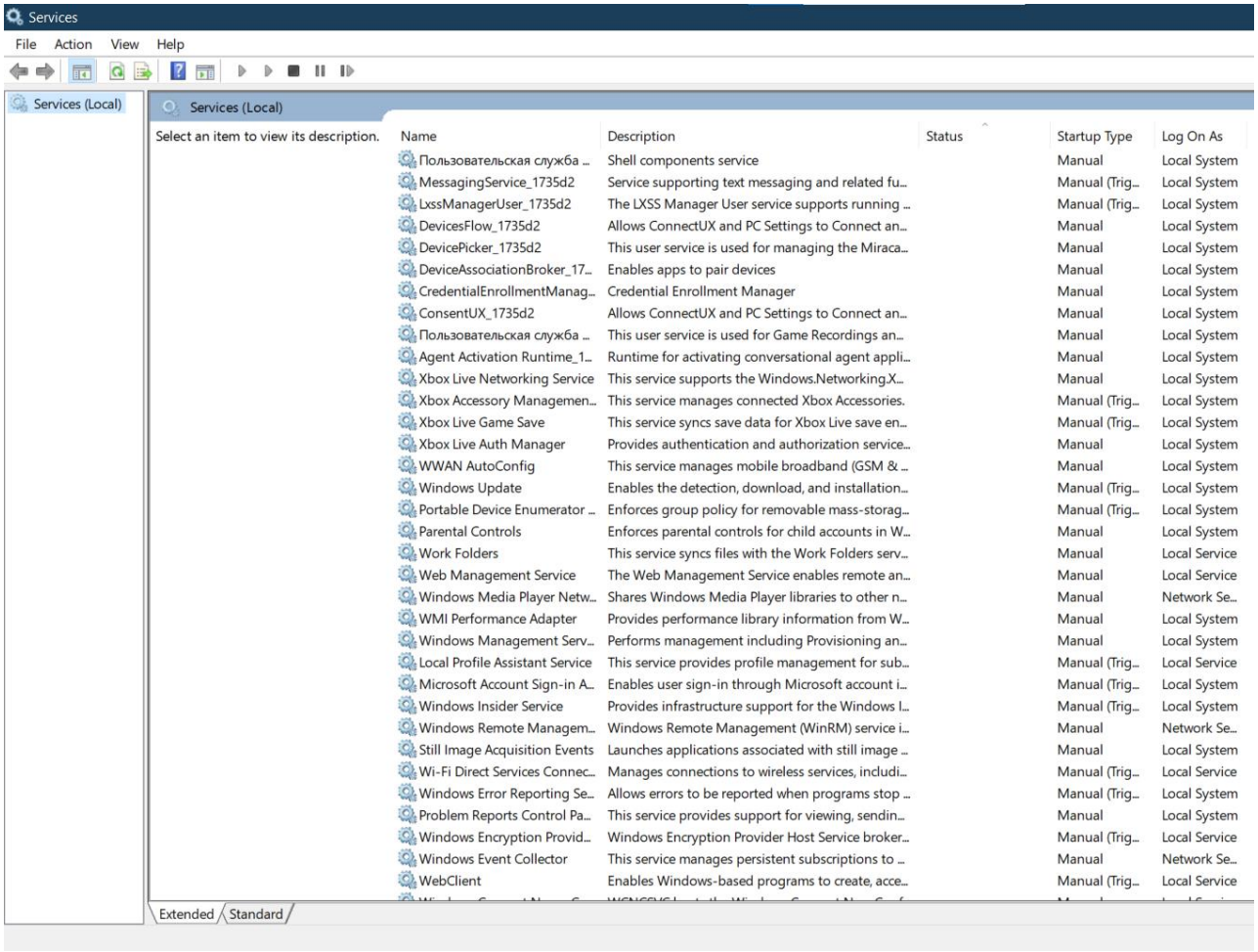

Рис.1.1. Інтерфейс програми "Служби Windows" (Services)

Серед сторонніх програм для управління службами можна виділити такі:

- Service Manager Plus: ця програма надає розширені можливості для управління службами, включаючи моніторинг їх стану, управління автозапуском та виконання більш складних завдань, таких як керування групами служб.

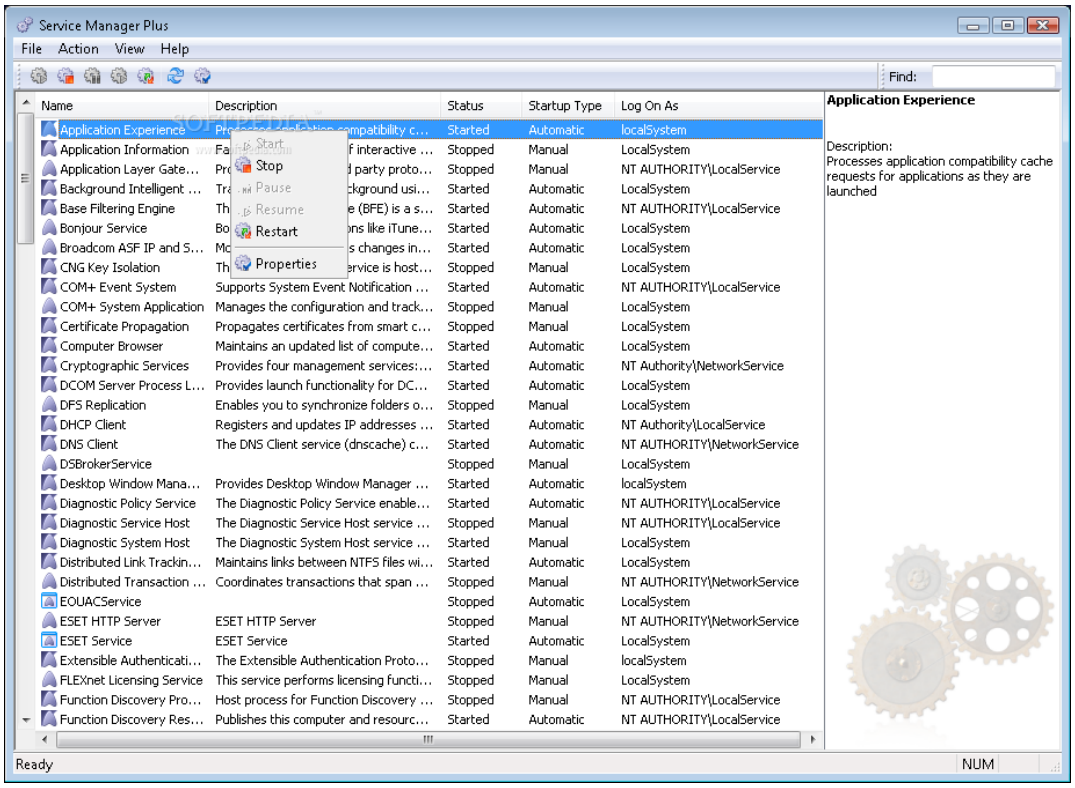

Рис.1.2. Інтерфейс програми "Service Manager Plus "

- NSSM: ця програма дозволяє запускати будь-яку програму як службу у Windows. Вона також надає можливість налаштовувати параметри служби та її автозапуску.

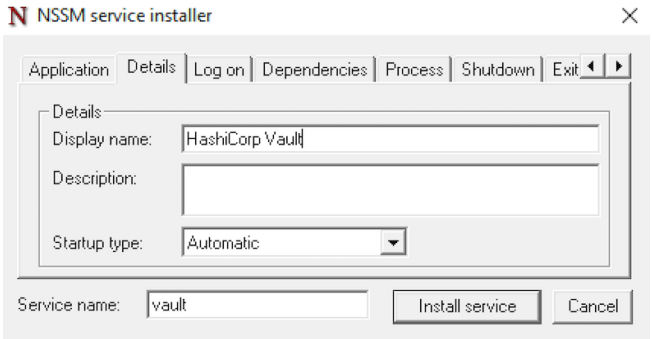

Рис.1.3. Інтерфейс програми "NSSM "

- Advanced SystemCare : ця програма пропонує розширені можливості управління службами, включаючи зупинку та запуск служб з одного місця, налаштування параметрів автозапуску, перегляд статусу та детальної інформації про кожну службу.

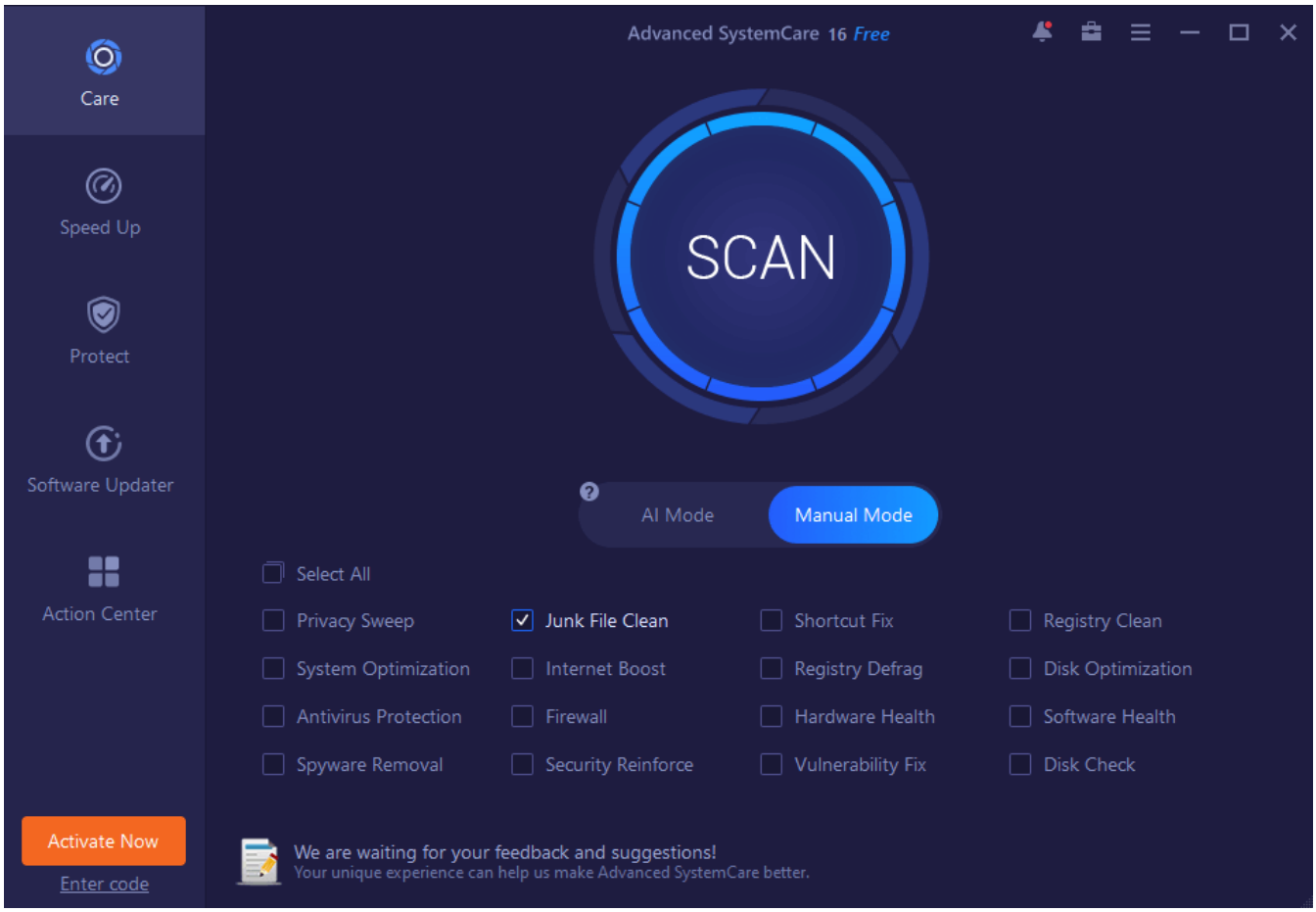

Рис.1.4. Інтерфейс програми "Advanced SystemCare "

Ці програми допомагають забезпечити більш ефективне та зручне управління службами у Windows, що може бути особливо корисним для більших комп'ютерних систем або для користувачів з вимогливими потребами.

Всі вони мають свої плюси і мінуси, так ,наприклад, можна сказати, що Service «Manager Plus» та «Служби Windows» мають забагато елементів інтерфейсу, що може викликати складності у користувачів, що ніколи не користувались цими програмами, з іншого боку, багато кнопок і меню означає і великий функціонал. Програма «Advanced SystemCare» , загалом, не розроблена для управління саме службами Windows. Вона

може допомогти з перевіркою файлів комп’ютера на віруси або з пришвидшенням роботи програм з високими вимогами, наприклад, ігор. Керування службами є лише частоної її функціоналу, а отже, якщо нам крім цього більше нічого не потрібно вона не є кращим варіантом, до того ж, її повна версія є платною.

На основі проведенного аналізу існуючих программ, що дозволяють керувати сервісами Windows, можна прийти до висновку, що вони розділяються на групи, кожна з яких займає свою нішу: Для звичайних користувачів комп'ютерів, які бажають мати зручний інтерфейс і багатофункціональні програми : такі люди рідко працюють зі службами , якщо взагалі це роблять, і для них не потрібно мати можливості поглибленної роботи з ними; Програми для спеціалістів, яким потрібно не стільки програма для керуванням служб, як щось ,навіть, більш спеціализоване, наприклад , як «*NSSM*»; тощо.

Висновок: Найбільш вигідною стратегією буде створення програми, що займе таку нішу, де альтернатива або відсутня, або її надто мало. Її ключовими характеристиками буде:

- Невелика кількість корисних функцій;

- Можливість запуску на слабкому комп’ютері;

- Безкоштовність.

2.РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ, АЛГОРИТМІВ ЇЇ РОБОТИ, ВИБІР ЗАСОБІВ

2.1 Опис консольного меню

У будь-якому екрані меню є можливість перейти на попереднє. На першому екрані є можливість вийти з програми.

При запуску програми користувач бачить консольне меню з можливістю обрати дію:

1) Переглянути активні служби (active services)
2) Переглянути неактивні служби (inactive services)
3) Переглянути всі служби (all services)
4) Дія зі службою (process service)
5) Вийти (exit)

Після натискання на 1 , 2 або 3 користувач бачить меню з питанням виводити детальну інформацію, чи ні. При детальному виводі користувач , окрім назв служб, побачить також:

- Назву служби (display name)
- Опис служби (description)
- Режим запуску (автоматично , вручну) (start type)
- Тип служби (Service type)
- Чи може служба бути зупинена (can stop)
- Назву служби (яку не можна змінити) (service name)

При автоматичному режимі запуска служби вмикається при запуску щпераційної системи.

Після натискання на 3 користувач побачить надпис «Введіть назву служби». У тому випадку, якщо введена служба існує користувач бачить детальну інформацію про неї , а також наступне меню:

1) Запустити / зупинити (залежить від стану служби) (run / stop)
2) Перезапустити (restart)
3) Змінити назву (change display name)
4) Змінити опис (change description)
5) Змінити режим запуску (автоматично, вручну) (change start type)
6) Назад (back)

Якщо службу неможливо зупинити , про це буде вказано в детальному описі служби, і відповідний пункт меню буде відсутній.

Якщо служба неактивна, можливість перезапустити її буде відсутня.

2.2 Архітектура програми

3.ПРОГРАМУВАННЯ, РЕАЛІЗАЦІЯ ПРОЕКТОВАНОГО РІШЕННЯ

SPZCW:

Program.cs

```
using System.ServiceProcess;
using SPZCW.Classes;
using SPZCW.Interfaces;
using SPZCW.Nums;

namespace SPZCW
{
    public class Program
    {
        public static IService[] Services { get; set; } = GetServices();
        static void Main(string[] args)
        {
            MainMenuChartType type = MainMenuChartType.ByStatus;
            while (true)
            {
                Menu.ProcessMainMenu(type);

                if (type == MainMenuChartType.ByMachineName)
                {
                    type = MainMenuChartType.ByStatus;
                }
                else
                {
                    type++;
                }
            }
        }

        public static IService [] GetServices()
        {
            ServiceController[] services = ServiceController.GetServices();
            IService[] Services = new Service[services.Length];

            for (int i = 0; i < services.Length; i++)
            {
                ServiceControllerWrapper SCwrapper = new
ServiceControllerWrapper(services[i]);
                Services[i] = new Service(SCwrapper);
            }

            return Services;
        }
    }
}
```

Service.cs

```
using Microsoft.Win32;
using SPZCW.Interfaces;
using System;
using System.Management;
using System.ServiceProcess;

namespace SPZCW
{
    public class Service : IService
    {
        public string Path { get;}
        public string Description { get; private set; }
        public string DisplayName { get; private set; }
        public string ServiceName { get; }
        public string MachineName { get; }
        public bool CanStop { get; }
        public ServiceStartMode StartType { get; private set; }
        public ServiceType ServiceType { get; }
        public ServiceControllerStatus Status { get; private set; }
        private IServiceController _service { get; set; }

        public Service(IServiceController service)
        {
            _service = service;
            Path = FindPath();
            Description = FindDescription(service);

            DisplayName = service.DisplayName;
            ServiceName = service.ServiceName;
            MachineName = service.MachineName;

            CanStop = service.CanStop;

            StartType = service.StartType;
            ServiceType = service.ServiceType;
            Status = service.Status;
        }

        private string FindPath()
        {
            ManagementObject wmiService = new ManagementObject("Win32_Service.Name='" +
_service.ServiceName + "'");

            try
            {
                if (wmiService["PathName"] == null)
                {
                    return "";
                }
                wmiService.Get();
                return wmiService["PathName"].ToString();
            }
            catch (Exception)
            {
                return "";
            }
        }
```

```
        private static string FindDescription(IServiceController service)
        {
            string description = "";
            ManagementObject serviceObject = new ManagementObject(new
ManagementPath(string.Format("Win32_Service.Name='{0}'", service.ServiceName)));

            try
            {
                if (serviceObject["Description"] == null)
                {
                    description = "-";
                    return description;
                }
                description = serviceObject["Description"].ToString();
                return description;
            }
            catch (Exception)
            {
                description = "-";
                return description;
            }
        }

        public void Start()
        {
            _service.Refresh();

            if (_service.Status != ServiceControllerStatus.Running)
            {
                _service.Start();
            }
            else
            {
                throw new Exception($"Service \"{_service.DisplayName}\" is already running");
            }

            Program.Services = Program.GetServices();
        }

        public void Stop()
        {
            _service.Refresh();

            if (_service.Status != ServiceControllerStatus.Stopped)
            {
                _service.Stop();
            }
            else
            {
                throw new Exception($"Service \"{_service.DisplayName}\" is already stopped");
            }

            Program.Services = Program.GetServices();
        }

        public void Restart()
        {
            _service.Refresh();

            if (_service.Status != ServiceControllerStatus.Stopped)
```

```
            {
                _service.Stop();
                _service.Start();
            }
            else
            {
                throw new Exception($"Service \"{_service.DisplayName}\" is stopped");
            }

            Program.Services = Program.GetServices();
        }

        public void ChangeDisplayName(string newName)
        {
            using (var serviceKey =
Registry.LocalMachine.OpenSubKey($"SYSTEM\\CurrentControlSet\\Services\\{_service.ServiceName}",
true))
            {
                if (serviceKey != null)
                {
                    serviceKey.SetValue("DisplayName", newName);
                }
                else
                {
                    throw new Exception($"Service \"{_service.ServiceName}\" was not found in the
registry.");
                }
            }

            _service.Refresh();
            Program.Services = Program.GetServices();
        }

        public void ChangeDescription(string newDescription)
        {
            using (var serviceKey =
Registry.LocalMachine.OpenSubKey($"SYSTEM\\CurrentControlSet\\Services\\{_service.ServiceName}",
true))
            {
                if (serviceKey != null)
                {
                    serviceKey.SetValue("Description", newDescription);
                }
                else
                {
                    throw new Exception($"Service \"{_service.ServiceName}\" was not found in the
registry.");
                }
            }

            _service.Refresh();
            Program.Services = Program.GetServices();

            Description = FindDescription(_service);
        }

        public void ChangeStartType(ServiceStartMode newMode)
        {
            var serviceKey =
Registry.LocalMachine.OpenSubKey($"SYSTEM\\CurrentControlSet\\Services\\{_service.ServiceName}",
```

```
true);

            if (serviceKey != null)
            {
                serviceKey.SetValue("Start", (int)newMode);
            }
            else
            {
                throw new Exception($"Service \"{_service.ServiceName}\" was not found in the
registry.");
            }

            _service.Refresh();
            Program.Services = Program.GetServices();
        }
    }
}
```

ServiceControllerWrapper.cs

```
using SPZCW.Interfaces;
using System.ServiceProcess;

namespace SPZCW.Classes
{
    public class ServiceControllerWrapper : IServiceController
    {
        private ServiceController _serviceController { get; set; }

        public ServiceControllerWrapper(ServiceController serviceController)
        {
            _serviceController = serviceController;
        }

        public string DisplayName
        {
            get { return _serviceController.DisplayName; }
            set { _serviceController.DisplayName = value; }
        }

        public string MachineName
        {
            get { return _serviceController.MachineName; }
            set { _serviceController.MachineName = value; }
        }

        public string ServiceName
        {
            get { return _serviceController.ServiceName; }
            set { _serviceController.ServiceName = value; }
        }

        public ServiceStartMode StartType
        {
            get { return _serviceController.StartType; }
        }
```

```
        public ServiceType ServiceType
        {
            get { return _serviceController.ServiceType; }
        }

        public bool CanStop
        {
            get { return _serviceController.CanStop; }
        }

        public ServiceControllerStatus Status
        {
            get { return _serviceController.Status; }
        }

        public bool Refresh()
        {
            try
            {
                _serviceController.Refresh();
                return true;
            }
            catch
            {
                return false;
            }
        }

        public void Start()
        {
            _serviceController.Start();
        }
        public void Stop()
        {
            _serviceController.Stop();
        }
    }
}
```

FilterSettings.cs

```
using SPZCW.Enumerations;
using SPZCW.Interfaces;
using System;
using System.Collections.Generic;
using System.ServiceProcess;

namespace SPZCW.Classes
{
    public class FilterSettings : IFilterSettings
    {
        public List<ServiceControllerStatus> Statuses { get; }
        public List<ServiceStartMode> StartModes { get; }
        public List<ServiceLocation> Locations { get; }

        public FilterSettings(List<string> choices)
        {
```

```
            Statuses = new List<ServiceControllerStatus>();
            StartModes = new List<ServiceStartMode>();
            Locations = new List<ServiceLocation>();

            foreach (string choise in choices)
            {
                AddChoiseAsSetting(choise);
            }
        }

        private void AddChoiseAsSetting(string choise)
        {
            if (choise == "Running") // Checking if choise is status
            {
                Statuses.Add(ServiceControllerStatus.Running);
            }
            else if (choise == "Stopped")
            {
                Statuses.Add(ServiceControllerStatus.Stopped);
            }
            else if (choise == "Other")
            {
                Statuses.Add(ServiceControllerStatus.Paused);
                Statuses.Add(ServiceControllerStatus.ContinuePending);
                Statuses.Add(ServiceControllerStatus.PausePending);
                Statuses.Add(ServiceControllerStatus.StartPending);
                Statuses.Add(ServiceControllerStatus.StopPending);
            }
            else if (choise == "Manual") // Checking if choise is StartMode
            {
                StartModes.Add(ServiceStartMode.Manual);
            }
            else if (choise == "Automatic")
            {
                StartModes.Add(ServiceStartMode.Automatic);
            }
            else if (choise == "Disabled")
            {
                StartModes.Add(ServiceStartMode.Disabled);
            }
            else if (choise == "Boot")
            {
                StartModes.Add(ServiceStartMode.Boot);
            }
            else if (choise == "System")
            {
                StartModes.Add(ServiceStartMode.System);
            }
            else if (choise == "Localhost") // Checking if choise is Location
            {
                Locations.Add(ServiceLocation.LocalHost);
            }
            else if (choise == "Another device")
            {
                Locations.Add(ServiceLocation.AnotherDevice);
            }
            else
            {
                throw new Exception($"Unknown setting: {choise}");
            }
```

```
        }
    }
}
```

AllServices.cs

```
using System.ServiceProcess;

namespace SPZCW.Classes.StaticClasses.SpectreConsoleObjects.MainMenuChart
{
    abstract public class AllServices
    {
        protected ServiceController[] Services { get; private set; }

        public AllServices()
        {
            Services = ServiceController.GetServices();
        }
    }
}
```

MainMenuChart.cs

```
using Spectre.Console;
using SPZCW.Classes.StaticClasses.SpectreConsoleObjects.MainMenuChart;

namespace SPZCW.Classes.StaticClasses
{
    public static class MainMenuChart
    {
        public static BreakdownChart GetMainMenuChartByStatus()
        {
            ServicesStatusData servicesStatusData = new ServicesStatusData();

            return new BreakdownChart()
                    .FullSize()
                    .ShowPercentage()
                    .AddItem("Stopped", (int)servicesStatusData.StoppedPercentage,
Colors.Color1)
                    .AddItem("Running", (int)servicesStatusData.RunningPercentage,
Colors.Color2)
                    .AddItem("Other", (int)servicesStatusData.OtherPercentage,
Colors.Color3);
        }

        public static BreakdownChart GetMainMenuChartByStartType()
        {
            ServicesStartTypeData servicesStartTypeData = new ServicesStartTypeData();

            return new BreakdownChart()
                    .FullSize()
                    .ShowPercentage()
```

```
                    .AddItem("Automatic", (int)servicesStartTypeData.AutomaticPercentage,
Colors.Color1)
                    .AddItem("Manual", (int)servicesStartTypeData.ManualPercentage,
Colors.Color2)
                    .AddItem("Disabled", (int)servicesStartTypeData.DisabledPercentage,
Colors.Color3)
                    .AddItem("System", (int)servicesStartTypeData.SystemPercentage,
Colors.Color4)
                    .AddItem("Boot", (int)servicesStartTypeData.BootPercentage,
Colors.Color5);
        }

        public static BreakdownChart GetMainMenuChartByServiceType()
        {
            ServicesTypesData servicesTypesData = new ServicesTypesData();

            return new BreakdownChart()
                    .FullSize()
                    .ShowPercentage()
                    .AddItem("Win32OwnProcess",
(int)servicesTypesData.Win32OwnProcessPercentage, Colors.Color1)
                    .AddItem("Win32ShareProcess",
(int)servicesTypesData.Win32ShareProcessPercentage, Colors.Color2)
                    .AddItem("KernelDriver", (int)servicesTypesData.KernelDriverPercentage,
Colors.Color3)
                    .AddItem("FileSystemDriver",
(int)servicesTypesData.FileSystemDriverPercentage, Colors.Color4)
                    .AddItem("Adapter", (int)servicesTypesData.AdapterPercentage,
Colors.Color5)
                    .AddItem("InteractiveProcess",
(int)servicesTypesData.InteractiveProcessPercentage, Colors.Color4)
                    .AddItem("RecognizerDriver",
(int)servicesTypesData.RecognizerDriverPercentage, Colors.Color3)
                    .AddItem("Other", (int)servicesTypesData.OtherPercentage, Colors.Color2);
        }

        public static BreakdownChart GetMainMenuChartByMachineName()
        {
            ServicesMachineNameData servicesMachineNameData = new ServicesMachineNameData();

            return new BreakdownChart()
                    .FullSize()
                    .ShowPercentage()
                    .AddItem("Localhost", (int)servicesMachineNameData.LocalPercentage,
Colors.Color1)
                    .AddItem("Other", (int)servicesMachineNameData.OtherPercentage,
Colors.Color2);
        }
    }
}
```

ServicesMachineNameData.cs

```
namespace SPZCW.Classes.StaticClasses.SpectreConsoleObjects.MainMenuChart
{
    public class ServicesMachineNameData : AllServices
    {
        public float LocalPercentage { get; private set; }
        public float OtherPercentage { get; private set; }

        private float _localAmount { get; set; }
        private float _otherAmount { get; set; }

        public ServicesMachineNameData()
        {
            CountMachineName();
            CalcMachineNamePercantage();
        }

        private void CountMachineName()
        {
            for (int i = 0; i < Services.Length; i++)
            {
                if (Services[i].MachineName == ".")
                {
                    _localAmount++;
                }
                else
                {
                    _otherAmount++;
                }
            }
        }

        private void CalcMachineNamePercantage()
        {
            LocalPercentage = (100 * _localAmount) / Services.Length;
            OtherPercentage = (100 * _otherAmount) / Services.Length;
        }
    }
}
```

ServicesStartTypeData.cs

```
using System.ServiceProcess;

namespace SPZCW.Classes.StaticClasses.SpectreConsoleObjects.MainMenuChart
{
    public class ServicesStartTypeData : AllServices
    {
        public float AutomaticPercentage { get; private set; }
        public float ManualPercentage { get; private set; }
        public float DisabledPercentage { get; private set; }
        public float SystemPercentage { get; private set; }
        public float BootPercentage { get; private set; }
```

```
        private float _automaticAmount { get; set; }
        private float _manualAmount { get; set; }
        private float _disabledAmount { get; set; }
        private float _systemAmount { get; set; }
        private float _bootAmount { get; set; }

        public ServicesStartTypeData()
        {
            Count();
            CalcPercentage();
        }

        private void Count()
        {
            for (int i = 0; i < Services.Length; i++)
            {
                if (Services[i].StartType == ServiceStartMode.Automatic)
                {
                    _automaticAmount++;
                }
                else if (Services[i].StartType == ServiceStartMode.Manual)
                {
                    _manualAmount++;
                }
                else if (Services[i].StartType == ServiceStartMode.Disabled)
                {
                    _disabledAmount++;
                }
                else if (Services[i].StartType == ServiceStartMode.System)
                {
                    _systemAmount++;
                }
                else // services[i].StartType == ServiceStartMode.Boot
                {
                    _bootAmount++;
                }
            }
        }

        private void CalcPercentage()
        {
            AutomaticPercentage = (100 * _automaticAmount) / Services.Length;
            ManualPercentage = (100 * _manualAmount) / Services.Length;
            DisabledPercentage = (100 * _disabledAmount) / Services.Length;
            SystemPercentage = (100 * _systemAmount) / Services.Length;
            BootPercentage = (100 * _bootAmount) / Services.Length;
        }
    }
}
```

ServicesSatatusData.cs

```
using System.ServiceProcess;

namespace SPZCW.Classes.StaticClasses.SpectreConsoleObjects.MainMenuChart
```

```
{
    public class ServicesStatusData : AllServices
    {
        public float StoppedPercentage { get; private set; }
        public float RunningPercentage { get; private set; }
        public float OtherPercentage { get; private set; }

        private float _stoppedAmount { get;  set; }
        private float _runningAmount { get;  set; }
        private float _otherAmount { get;  set; }

        public ServicesStatusData()
        {
            Count();
            CalcPercentage();
        }

        private void Count()
        {
            for (int i = 0; i < Services.Length; i++)
            {
                if (Services[i].Status == ServiceControllerStatus.Stopped)
                {
                    _stoppedAmount++;
                }
                else if (Services[i].Status == ServiceControllerStatus.Running)
                {
                    _runningAmount++;
                }
                else
                {
                    _otherAmount++;
                }
            }
        }

        private void CalcPercentage()
        {
            StoppedPercentage = (100 * _stoppedAmount) / Services.Length;
            RunningPercentage = (100 * _runningAmount) / Services.Length;
            OtherPercentage = (100 * _otherAmount) / Services.Length;
        }
    }
}
```

ServicesTypesData.cs

```
using System.ServiceProcess;

namespace SPZCW.Classes.StaticClasses.SpectreConsoleObjects.MainMenuChart
{
    class ServicesTypesData : AllServices
    {
        public float KernelDriverPercentage { get; private set; }
        public float FileSystemDriverPercentage { get; private set; }
```

```
        public float AdapterPercentage { get; private set; }
        public float InteractiveProcessPercentage { get; private set; }
        public float RecognizerDriverPercentage { get; private set; }
        public float Win32OwnProcessPercentage { get; private set; }
        public float Win32ShareProcessPercentage { get; private set; }
        public float OtherPercentage { get; private set; }

        private float _kernelDriverAmount { get; set; }
        private float _fileSystemDriverAmount { get; set; }
        private float _adapterAmount { get; set; }
        private float _interactiveProcessAmount { get; set; }
        private float _recognizerDriverAmount { get; set; }
        private float _win32OwnProcessAmount { get; set; }
        private float _win32ShareProcessAmount { get; set; }
        private float _otherAmount { get; set; }

        public ServicesTypesData()
        {
            CountServiceType();
            CalcServiceTypePercentage();
        }
        private void CountServiceType()
        {
            for (int i = 0; i < Services.Length; i++)
            {
                if (Services[i].ServiceType == ServiceType.KernelDriver)
                {
                    _kernelDriverAmount++;
                }
                else if (Services[i].ServiceType == ServiceType.FileSystemDriver)
                {
                    _fileSystemDriverAmount++;
                }
                else if (Services[i].ServiceType == ServiceType.Adapter)
                {
                    _adapterAmount++;
                }
                else if (Services[i].ServiceType == ServiceType.InteractiveProcess)
                {
                    _interactiveProcessAmount++;
                }
                else if (Services[i].ServiceType == ServiceType.RecognizerDriver)
                {
                    _recognizerDriverAmount++;
                }
                else if (Services[i].ServiceType == ServiceType.Win32OwnProcess)
                {
                    _win32OwnProcessAmount++;
                }
                else if (Services[i].ServiceType == ServiceType.Win32ShareProcess)
                {
                    _win32ShareProcessAmount++;
                }
                else
                {
                    _otherAmount++;
                }
            }
        }
```

```
        private void CalcServiceTypePercentage()
        {
            KernelDriverPercentage = (100 * _kernelDriverAmount) / Services.Length;
            FileSystemDriverPercentage = (100 * _fileSystemDriverAmount) / Services.Length;
            AdapterPercentage = (100 * _adapterAmount) / Services.Length;
            InteractiveProcessPercentage = (100 * _interactiveProcessAmount) / Services.Length;
            RecognizerDriverPercentage = (100 * _recognizerDriverAmount) / Services.Length;
            Win32OwnProcessPercentage = (100 * _win32OwnProcessAmount) / Services.Length;
            Win32ShareProcessPercentage = (100 * _win32ShareProcessAmount) / Services.Length;
            OtherPercentage = (100 * _otherAmount) / Services.Length;
        }
    }
}
```

Menues.cs

```
using Spectre.Console;
using SPZCW.Interfaces;
using System.ServiceProcess;

namespace SPZCW.Classes.StaticClasses
{
    public static class Menues
    {
        public static SelectionPrompt<string> GetMainMenu()
        {
            return new SelectionPrompt<string>().AddChoices(new[]
            {
            "Active services",
            "Stopped services",
            "All services",
            "Filter services",
            "Process service",
            "Help",
            "[red1]Exit[/]"
            });
        }

        public static MultiSelectionPrompt<string> GetFilterMenu()
        {
            return new MultiSelectionPrompt<string>()
                        .PageSize(15)
                        .AddChoiceGroup<string>("Status", new string[] {
                        "Running" , "Stopped", "Other"})
                        .AddChoiceGroup("Start type", new string[] {
                            "Manual", "Automatic", "Disabled", "Boot", "System"})
                        .AddChoiceGroup("MachineName", new string[] {
                            "Localhost" , "Another device"})
                        .AddChoiceGroup("Back", new string[] { });
        }

        public static SelectionPrompt<string> GetActionsMenu(IService service)
        {
            return new SelectionPrompt<string>().AddChoices(new[]
            {
```

```
            GetStartOrStopChoise(service),
            "Restart",
            "Change start type",
            "Change display name",
            "Change description",
            "[red1]Back[/]",
            });
        }

        private static string GetStartOrStopChoise(IService service)
        {
            if (service.Status == ServiceControllerStatus.Stopped || !service.CanStop)
            {
                return "Start";
            }
            else
            {
                return "Stop";
            }
        }

        public static SelectionPrompt<string> GetChangeStartTypeMenu()
        {
            return new SelectionPrompt<string>().AddChoices(new[]
            {
            "Manual",
            "Automatic",
            "Disabled",
            "Boot",
            "System",
            "[red1]Back[/]"
            });
        }

        public static SelectionPrompt<string> GetHelpMenu()
        {
            return new SelectionPrompt<string>().AddChoices(new[]
           {
            "Program description",
            "Service Status",
            "Service names",
            "Service start types",
            "Service types",
            "[red1]Back[/]"
            });
        }
    }
}
```

PathTree.cs

```
using Spectre.Console;

namespace SPZCW.Classes.StaticClasses.SpectreConsoleObjects
{
    public static class PathTree
```

```
    {
        public static Tree GetServicePathTree(string path)
        {
            string[] splittedPath = path.Split('\\');

            Tree pathRoot = null;
            TreeNode[] branches = new TreeNode[splittedPath.Length - 1];

            if (splittedPath.Length >= 1)
            {
                pathRoot = new Tree(splittedPath[0]);

                if (splittedPath.Length >= 2)
                {
                    branches[0] = pathRoot.AddNode(splittedPath[1]);
                }
            }

            for (int i = 2; i < splittedPath.Length; i++)
            {
                branches[i - 1] = branches[i - 2].AddNode(splittedPath[i]);
            }

            return pathRoot;
        }
    }
}
```

Tables.cs

```
using Spectre.Console;
using SPZCW.Enumerations;
using SPZCW.Interfaces;
using System.ServiceProcess;

namespace SPZCW.Classes.StaticClasses
{
    public static class Tables
    {
        public static Table GetAllServicesTable()
        {
            var table = GetServicesTable(true);

            foreach (var service in Program.Services)
            {
                AddServiceToTableWithStatus(table, service);
            }

            return table;
        }

        public static Table GetServicesTableByStatus(ServiceControllerStatus status)
        {
            var table = GetServicesTable(false);
```

```
            foreach (var service in Program.Services)
            {
                if (service.Status == status)
                {
                    AddServiceToTable(table, service);
                }
            }

            return table;
        }

        public static Table GetFilteredServicesTable(IFilterSettings filterSettings)
        {
            Table table = GetServicesTable(true);

            foreach (var service in Program.Services)
            {
                if (filterSettings.Statuses.Contains(service.Status) &&
filterSettings.StartModes.Contains(service.StartType))
                {
                    if ((service.MachineName == "." &&
filterSettings.Locations.Contains(ServiceLocation.LocalHost))
                    || (service.MachineName != "." &&
filterSettings.Locations.Contains(ServiceLocation.AnotherDevice)))
                    {
                        AddServiceToTableWithStatus(table,service);
                    }
                }
            }

            return table;
        }

        private static void AddServiceToTable(Table table, IService service)
        {
            table.AddRow(service.DisplayName, service.ServiceName, service.MachineName,
service.StartType.ToString(), service.ServiceType.ToString(), service.Path,
service.Description);
        }

        private static void AddServiceToTableWithStatus(Table table , IService service)
        {
            string statusStr = GetStatusStringWithColorNote(service);
            table.AddRow(service.DisplayName, service.ServiceName, service.MachineName,
service.StartType.ToString(), service.ServiceType.ToString(), service.Path,
service.Description, statusStr);
        }

        private static Table GetServicesTable(bool addStatusColumn)
        {
            Table table = new Table();

            table.AddColumn("DisplayName");
            table.AddColumn(new TableColumn("ServiceName"));
            table.AddColumn(new TableColumn("MachineName"));
            table.AddColumn(new TableColumn("Start type"));
            table.AddColumn(new TableColumn("ServiceType"));
            table.AddColumn(new TableColumn("Path"));
            table.AddColumn(new TableColumn("Description"));
            if (addStatusColumn)
```

```
            {
                table.AddColumn(new TableColumn("Status").Centered());
            }

            return table;
        }

        private static string GetStatusStringWithColorNote(IService service)
        {
            string statusStr;
            string color;

            if (service.Status == ServiceControllerStatus.Stopped)
            {
                color = "red1";
            }
            else if (service.Status == ServiceControllerStatus.Running)
            {
                color = "mediumturquoise";
            }
            else
            {
                color = "yellow";
            }

            statusStr = $"[bold black on {color}]{service.Status}[/]";
            return statusStr;
        }
    }
}
```

MainTitle.cs

```
using Spectre.Console;

namespace SPZCW.Classes.StaticClasses.SpectreConsoleObjects
{
    public static class MainTitle
    {
        public static FigletText Title { get;} = new
FigletText("SPZCW").LeftJustified().Centered();
    }
}
```

Colors.cs

```
using Spectre.Console;

namespace SPZCW.Classes.StaticClasses
{
```

```
    public static class Colors
    {
        public static Color Color1 { get; } = new Color(0, 111, 116);
        public static Color Color2 { get; } = new Color(129, 205, 194);
        public static Color Color3 { get; } = new Color(254, 242, 215);
        public static Color Color4 { get; } = new Color(243, 140, 118);
        public static Color Color5 { get; } = new Color(242, 78, 74);
    }
}
```

HelpMenuText.cs

```
using Spectre.Console;

namespace SPZCW.Classes.StaticClasses
{
    public static class HelpMenuText
    {
        public static Markup ServiceDescription { get; } = new Markup("[bold]This program is
designed to provide an intuitive and" +
            " user-friendly interface for + managing Windows services.[/]\n\n\r" +
            "You can view information about running, stopped and services with any status using
buttons" +
            " \"Active services\", \"Stopped services\" and \"All services\" respectively. If
you need to search for other" +
            " criteria, or more of them, you can use the \"Filter services\" button. To change
a service, you need to find" +
            " it by display name via the \"Process service\" button. After that, you will see
detailed information about the" +
            " found service and you will be able to perform certain actions with it (start/stop
(depending on the current state)," +
            " restart, change display name, change description, change service type). Please note
that some services cannot be" +
            " stopped, this is indicated in the description to them after the search. Enjoy using
the program.\n\n\r"
        );

        public static Markup ServiceTypes { get; } = new Markup("[bold]In Windows operating
systems, services are classified into several" +
                "types based on how they interact with the system and what resources they
manage.[/]\n\n\r" +
            "[mediumturquoise]Kernel Driver[/]: these are services that run in the context of
the operating system " +
                "kernel and provide access to hardware or other resources that require low-level
control. " +
            "Examples include device drivers for devices such as sound cards, video cards, or
network adapters.\n\n\r" +
            "[mediumturquoise]File System Driver[/]: these are services that provide access to
file systems and" +
                " manage read and write operations on disks. Such services can be used to work
with various types" +
                " of file systems, such as NTFS or FAT32.\n\n\r" +
            "[mediumturquoise]Adapter[/]: these are services that provide access to network
adapters and manage " +
                "data transmission through them. Examples include services related to TCP/IP
```

```
protocols, as well as " +
                "services for processing network packet traffic.\n\n\r" +
            "[mediumturquoise]Interactive Process[/]: these are services designed to interact
with the user through" +
                " a graphical interface. Examples include services for managing the display or
services for managing" +
                " user notifications.\n\n\r" +
            "[mediumturquoise]Recognizer Driver[/]: these are services that provide" +
                " recognition of various devices and resources. Examples include services that
detect and recognize" +
                " connected USB devices or services that manage speech recognition.\n\n\r" +
            "[mediumturquoise]Win32 Own Process[/]: these are services that run in their own
process and do not share" +
                " resources with other processes. Examples include services for running server
applications" +
                " or services that manage databases.\n\n\r" +
            "[mediumturquoise]Win32 Share Process[/]: these are services that run in a shared
process with" +
                " other services, allowing them to share system resources. Examples include
services for managing" +
                " printing or services for providing network security.\n\n\r"
        );

        public static Markup ServiceNames { get; } = new Markup("[bold]Each service is identified
by a unique name, which consists of three components: " +
                "the service name, the display name, and the machine name.[/]\n\n\r" +
            "[mediumturquoise]DisplayName[/]: The display name is the name that is shown to users
in the Windows" +
                " Services applet in the Control Panel. The display name can be different from
the service name," +
            "[mediumturquoise]ServiceName[/]: The service name is the name used to identify the
service internally in" +
                " Windows. This name is used by the operating system to start and stop the service,"
+
                " as well as to monitor its status.\n\n\r" +
            "[mediumturquoise]MachineName[/]: This name is used to specify the name of the machine
on which" +
                " the service is running. This property is optional, and if it is not specified,
the local machine" +
                " is assumed.\n\n\r"
        );

        public static Markup ServiceStatus { get; } = new Markup("[bold]In Windows operating
system, services are background processes that" +
                " can be started automatically when the system boots up or manually by a user
or another application." +
                " Services have a set of predefined statuses that indicate their current state
of operation." +
                "The status of a service is one of the most important pieces of information when
it comes to" +
                " managing and troubleshooting services in Windows.There are seven possible
service statuses" +
                " in Windows. These statuses are:[/]\n\n\r" +
            "[mediumturquoise]Stopped[/]: This value indicates that the service is not running.
" +
                "This means that the service is not currently executing any tasks or performing
any operations.\n\n\r" +
            "[mediumturquoise]StartPending[/]: This value indicates that the service has been
requested to start, but" +
                " it is not yet running. This means that the service is currently in the process
```

```
of starting up.\n\n\r" +
            "[mediumturquoise]StopPending[/]: This value indicates that the service has been
requested to stop," +
                " but it is not yet stopped. This means that the service is currently in the
process" +
                " of shutting down.\n\n\r" +
            "[mediumturquoise]Running[/]: This value indicates that the service is currently
running. " +
                "This means that the service is currently executing tasks or performing
operations.\n\n\r" +
            "[mediumturquoise]ContinuePending[/]: This value indicates that the service has been
requested" +
                " to continue from a paused state, but it is not yet running. This means that
the service" +
                " is currently in the process of resuming its operations after being
paused.\n\n\r" +
            "[mediumturquoise]PausePending[/]: This value indicates that the service has been
requested" +
                " to pause, but it is not yet paused. This means that the service is currently
in the process" +
                " of being paused.\n\n\r" +
            "[mediumturquoise]Paused[/]: This value indicates that the service is currently
paused. This means that" +
                " the service is currently not executing any tasks or performing any operations,
" +
                "but it can be resumed later.\n\n\r"
        );

        public static Markup ServiceStartTypes { get; } = new Markup("[bold]In Windows operating
systems, a service is a program that runs in the background and" +
                " performs various tasks, such as managing network connections, printing
documents, or providing " +
                "remote access to resources. Windows services can have different start types,
which determine when " +
                "the service is loaded and started by the operating system.There are five
different service start " +
                "types in Windows:[/]\n\n\r" +
            "[mediumturquoise]Automatic[/]: This is the default start type for most Windows
services. " +
                "Services set to this start type are started automatically when the operating
system boots up. " +
                "This type of service is important for background processes that are essential
to the proper " +
                "functioning of the system, such as device drivers or antivirus software.\n\n\r"
+
            "[mediumturquoise]Manual[/]: Services set to this start type do not start
automatically when the operating " +
                "system boots up. Instead, they must be started manually by a user or by another
application. " +
                "This type of service is typically used for services that are not essential to
the functioning " +
                "of the system, but may be needed by specific applications.\n\n\r" +
            "[mediumturquoise]Disabled[/]: Services set to this start type are not started at
all, even if they " +
                "are required by other applications. This type of service is typically used for
services that " +
                "are no longer needed or that have  been replaced by another service.\n\n\r" +
            "[mediumturquoise]Boot[/]: Services set to this start type are started during the
boot process, before any" +
                " user logs in. This type of service is only used by system services and is not
```

```
available for " +
                "user-defined services.\n\n\r" +
            "[mediumturquoise]System[/]: This start type is used only by kernel-mode device
drivers and services that" +
                " are critical to the operating system. Services set to this start type are loaded
by the operating " +
                "system loader before any other start type.\n\n\r"
        );
    }
}
```

Menu.cs

```
using Spectre.Console;
using SPZCW.Classes;
using SPZCW.Classes.StaticClasses;
using SPZCW.Classes.StaticClasses.SpectreConsoleObjects;
using SPZCW.Interfaces;
using SPZCW.Nums;
using System;
using System.ServiceProcess;

namespace SPZCW
{
    static class Menu
    {
        public static void ProcessMainMenu(MainMenuChartType type)
        {
            ProcessTitleAndBar(type);
            var mainMenuChoise = AnsiConsole.Prompt(Menues.GetMainMenu());
            MainMenuChoiseProcessing(mainMenuChoise);
        }

        private static void ProcessTitleAndBar(MainMenuChartType type)
        {
            BreakdownChart chart;
            switch (type)
            {
                case MainMenuChartType.ByStatus:
                    chart = MainMenuChart.GetMainMenuChartByStatus();
                    break;
                case MainMenuChartType.ByStartType:
                    chart = MainMenuChart.GetMainMenuChartByStartType();
                    break;
                case MainMenuChartType.ByServiceType:
                    chart = MainMenuChart.GetMainMenuChartByServiceType();
                    break;
                case MainMenuChartType.ByMachineName:
                    chart = MainMenuChart.GetMainMenuChartByMachineName();
                    break;
                default:
                    throw new ArgumentException($"non-existent type: {type}");
            }
            AnsiConsole.Write(MainTitle.Title);
```

```
            AnsiConsole.Write(chart);
            Console.WriteLine("\n");
        }

        private static void MainMenuChoiseProcessing(string mainMenuChoise)
        {
            switch (mainMenuChoise)
            {
                case "Active services":
                    ActiveServicesChoiseProcessing();
                    break;
                case "Stopped services":
                    StoppedServicesChoiseProcessing();
                    break;
                case "All services":
                    AllServicesChoiseProcessing();
                    break;
                case "Filter services":
                    FilterChoisePocessing();
                    break;
                case "Process service":
                    ProcessServiceChoiseProcessing();
                    break;
                case "Help":
                    HelpChoiseProcessing();
                    break;
                case "[red1]Exit[/]":
                    Environment.Exit(0);
                    break;
            }
        }

        private static void ActiveServicesChoiseProcessing()
        {
            Console.WriteLine(Messages.ServicesWithStatus(ServiceControllerStatus.Running));
            var activeServices =
Tables.GetServicesTableByStatus(ServiceControllerStatus.Running);
            AnsiConsole.Write(activeServices);
        }

        private static void StoppedServicesChoiseProcessing()
        {
            Console.WriteLine(Messages.ServicesWithStatus(ServiceControllerStatus.Stopped));
            var stoppedServices =
Tables.GetServicesTableByStatus(ServiceControllerStatus.Stopped);
            AnsiConsole.Write(stoppedServices);
        }

        private static void AllServicesChoiseProcessing()
        {
            Console.WriteLine("All services:");
            var allServices = Tables.GetAllServicesTable();
            AnsiConsole.Write(allServices);
        }

        private static void FilterChoisePocessing()
        {
            var filterMenuChoises = AnsiConsole.Prompt(Menues.GetFilterMenu());
            if(filterMenuChoises.Contains("Back"))
            {
```

```
                return;
            }
            IFilterSettings filterSettings = new FilterSettings(filterMenuChoises);

            var filteredTable = Tables.GetFilteredServicesTable(filterSettings);
            AnsiConsole.Write(filteredTable);
        }

        private static void ProcessServiceChoiseProcessing()
        {
            IService service;
            try
            {
                service = GetServiceViaInput();
            }
            catch(Exception ex)
            {
                AnsiConsole.Write(Messages.Error(ex.Message));
                return;
            }

            ServiceActionsMenuProcessing(service);
        }

        private static IService GetServiceViaInput()
        {
            Console.Write("\nService DisplayName: ");
            string serviceDisplayName = Console.ReadLine();

            IService service = FindService(serviceDisplayName);
            if (service == null)
            {
                throw new Exception($"Service \"{serviceDisplayName}\" not found");
            }

            return service;
        }

        private static IService FindService(string serviceDisplayName)
        {
            for (int i = 0; i < Program.Services.Length; i++)
            {
                if (serviceDisplayName == Program.Services[i].DisplayName)
                {
                    return Program.Services[i];
                }
            }

            return null;
        }

        private static void ServiceActionsMenuProcessing(IService service)
        {
            while (true)
            {
                Console.WriteLine();
                AnsiConsole.Write(PathTree.GetServicePathTree(service.Path));
                Console.WriteLine(Messages.ServiceInfo(service));

                var actionsMenuChoise = AnsiConsole.Prompt(Menues.GetActionsMenu(service));
```

```
                if (actionsMenuChoise == "[red1]Back[/]")
                {
                    break;
                }
                ServiceActionsMenuChoiseProcessing(ref service, actionsMenuChoise);

                if(actionsMenuChoise != "Change display name")
                {
                    service = FindService(service.DisplayName);
                }
            }
        }

        private static void ServiceActionsMenuChoiseProcessing(ref IService service, string
actionsMenuChoise)
        {
            try
            {
                switch (actionsMenuChoise)
                {
                    case "Stop":
                        StopChoiseProcessing(service);
                        break;
                    case "Start":
                        StartChoiseProcessing(service);
                        break;
                    case "Restart":
                        RestartChoiseProcessing(service);
                        break;
                    case "Change display name":
                        ChangeDisplayNameChoiseProcessing(ref service);
                        break;
                    case "Change start type":
                        ChangeStartTypeChoiseProcession(service);
                        break;
                    case "Change description":
                        ChangeDescriptionChoiseProcession(service);
                        break;
                }
            }
            catch (Exception ex)
            {
                AnsiConsole.Write(Messages.Error(ex.Message));
            }
        } private static void StopChoiseProcessing(IService service)
        {
            try
            {
                service.Stop();
            }
            catch (Exception ex)
            {
                throw new Exception(ex.Message);
            }

            Console.WriteLine($"Service \"{service.DisplayName}\" stopped");
        }

        private static void StartChoiseProcessing(IService service)
        {
```

```
            if (service.StartType == ServiceStartMode.Disabled)
            {
                throw new Exception($"Service \"{service.DisplayName}\" can not be started
because it has \"Disabled\" start mode." +
                    " Change start mode to start service");
            }

            service.Start();
            Console.WriteLine($"Service \"{service.DisplayName}\" started\n");
        }

        private static void RestartChoiseProcessing(IService service)
        {
            if (service.Status == ServiceControllerStatus.Stopped)
            {
                throw new Exception($"Service \"{service.DisplayName}\" is not running");
            }

            service.Stop();
            service.Start();
            Console.WriteLine($"Service \"{service.DisplayName}\" restarted\n");
        }

        public static void ChangeDisplayNameChoiseProcessing(ref IService service)
        {
            Console.Write("New name: ");
            string oldName = service.DisplayName;
            string newName = Console.ReadLine();

            if (oldName == newName)
            {
                throw new Exception("It's current service DisplayName");
            }

            try
            {
                service.ChangeDisplayName(newName);
            }
            catch(Exception ex)
            {
                throw new Exception(ex.Message);
            }

            service = FindService(newName);
            Console.WriteLine($"Service DisplayName changed : {oldName} -> {newName} ");
        }

        private static void ChangeStartTypeChoiseProcession(IService service)
        {
            string ChangeStartTypeMenuChoise =
AnsiConsole.Prompt(Menues.GetChangeStartTypeMenu());
            string oldStartType = service.StartType.ToString();

            if (ChangeStartTypeMenuChoise == oldStartType)
            {
                throw new Exception("It's current service start type");
            }

            try
            {
```

```
                ChangeStartTypeMenuChoiseProcession(service, ChangeStartTypeMenuChoise);
            }
            catch(Exception ex)
            {
                throw new Exception(ex.Message);
            }

            Console.WriteLine($"Start type changed: {oldStartType} ->
{ChangeStartTypeMenuChoise}");
        }

        private static void ChangeStartTypeMenuChoiseProcession(IService service , string
ChangeStartTypeMenuChoise)
        {
            try
            {
                switch (ChangeStartTypeMenuChoise)
                {
                    case "Manual":
                        service.ChangeStartType(ServiceStartMode.Manual);
                        break;
                    case "Automatic":
                        service.ChangeStartType(ServiceStartMode.Automatic);
                        break;
                    case "Disabled":
                        service.ChangeStartType(ServiceStartMode.Disabled);
                        break;
                    case "Boot":
                        service.ChangeStartType(ServiceStartMode.Boot);
                        break;
                    case "System":
                        service.ChangeStartType(ServiceStartMode.System);
                        break;
                    case "[red1]Back[/]":
                        return;
                }
            }
            catch (Exception ex)
            {
                throw new Exception(ex.Message);
            }
        }

        private static void ChangeDescriptionChoiseProcession(IService service)
        {
            Console.Write("New description: ");
            string newDesc = Console.ReadLine();
            string oldDesc = service.Description;

            if (oldDesc == newDesc)
            {
                throw new Exception("It's current service description");
            }

            try
            {
                service.ChangeDescription(newDesc);
            }
            catch(Exception ex)
            {
```

```
                throw new Exception(ex.Message);
            }

            Console.WriteLine($"Service DisplayName changed : {oldDesc} -> {newDesc} ");
        }
        private static void HelpChoiseProcessing()
        {
            while (true)
            {
                var helpMenuChoise = AnsiConsole.Prompt(Menues.GetHelpMenu());

                if (helpMenuChoise == "[red1]Back[/]")
                {
                    break;
                }
                helpMenuChoiseProcessing(helpMenuChoise);
            }
        }

        private static void helpMenuChoiseProcessing(string helpMenuChoise)
        {
            switch (helpMenuChoise)
            {
                case "Program description":
                    AnsiConsole.Write(HelpMenuText.ServiceDescription);
                    break;
                case "Service names":
                    AnsiConsole.Write(HelpMenuText.ServiceNames);
                    break;
                case "Service start types":
                    AnsiConsole.Write(HelpMenuText.ServiceStartTypes);
                    break;
                case "Service Status":
                    AnsiConsole.Write(HelpMenuText.ServiceStatus);
                    break;
                case "Service types":
                    AnsiConsole.Write(HelpMenuText.ServiceTypes);
                    break;
            }
        }
    }
}
```

Messages.cs

```
using Spectre.Console;
using SPZCW.Interfaces;
using System.ServiceProcess;

namespace SPZCW.Classes.StaticClasses
{
    public static class Messages
    {
        public static string ServicesWithStatus(ServiceControllerStatus status)
        {
            return $"Services with status \"{status}\":";
        }
```

```
        public static Markup Error(string errorMessage)
        {
            return new Markup($"\n[red1]Error: {errorMessage}[/]\n");
        }

        public static string ServiceInfo(IService service)
        {
            string description = $"\nDescription  :\t{service.Description}\n\r";
            string displayName = $"DisplayName  :\t{service.DisplayName}\n\r";
            string serviceName = $"ServiceName  :\t{service.ServiceName}\n\r";
            string machineName = $"MachineName  :\t{service.MachineName}\n\r";
            string serviceType = $"Service Type :\t{service.ServiceType}\n\r";
            string startType =   $"Start Type   :\t{service.StartType}\n\r";
            string servicePath = $"Path         :\t{service.Path}\n\r";
            string status =      $"Status       :\t{service.Status}\n\r";
            string canStop =     $"Can stop     :\t{service.CanStop}\n\r";

            return description + displayName + serviceName + machineName + serviceType +
servicePath + startType + status + canStop;
        }
    }
}
```

MainMenuChartType.cs

```
namespace SPZCW.Nums
{
    public enum MainMenuChartType
    {
        ByStatus,
        ByStartType,
        ByServiceType,
        ByMachineName,
    }
}
```

ServiceLocation.cs

```
namespace SPZCW.Enumerations
{
    public enum ServiceLocation
    {
        LocalHost,
        AnotherDevice,
    }
}
```

IfilterSettings.cs

```
using System.ServiceProcess;
using SPZCW.Enumerations;
using System.Collections.Generic;

namespace SPZCW.Interfaces
{
    public interface IFilterSettings
    {
        List<ServiceControllerStatus> Statuses {get;}
        List<ServiceStartMode> StartModes { get;}
        List<ServiceLocation> Locations { get; }
    }
}
```

IService.cs

```
using System.ServiceProcess;

namespace SPZCW.Interfaces
{
    public  interface IService
    {
        string Path { get; }
        string Description { get; }
        string DisplayName { get; }
        string ServiceName { get; }
        string MachineName { get; }
        bool CanStop { get; }
        ServiceStartMode StartType { get; }
        ServiceType ServiceType { get; }
        ServiceControllerStatus Status { get; }

        void Start();
        void Stop();
        void Restart();
        void ChangeDisplayName(string newName);
        void ChangeStartType(ServiceStartMode newMode);
        void ChangeDescription(string newDescription);
    }
}
```

IserviceController.cs

```
using System.ServiceProcess;

namespace SPZCW.Interfaces
{
    public interface IServiceController
    {
        string DisplayName { get; set; }
        string MachineName { get; set; }
```

```
        string ServiceName { get; set; }
        ServiceStartMode StartType { get; }
        ServiceType ServiceType { get; }
        bool CanStop { get; }
        ServiceControllerStatus Status { get; }

        bool Refresh();
        void Start();
        void Stop();
    }
}
```

SPZCWTests:

MainMenuChartTests.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using SPZCW.Classes.StaticClasses;

namespace SPZCWTests.TestsClasses
{
    [TestClass]
    public class MainMenuChartTests
    {
        [TestMethod]
        public void TestGetMainMenuChartByStatusReturnsNotNull()
        {
            Assert.IsNotNull(MainMenuChart.GetMainMenuChartByStatus());
        }

        [TestMethod]
        public void TestGetMainMenuChartByStartTypeReturnsNotNull()
        {
            Assert.IsNotNull(MainMenuChart.GetMainMenuChartByStartType());
        }

        [TestMethod]
        public void TestGetMainMenuChartByServiceTypeReturnsNotNull()
        {
            Assert.IsNotNull(MainMenuChart.GetMainMenuChartByServiceType());
        }

        [TestMethod]
        public void TestGetMainMenuChartByMachineNameReturnsNotNull()
        {
            Assert.IsNotNull(MainMenuChart.GetMainMenuChartByMachineName());
        }
    }
}
```

MenuesTests.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Moq;
using SPZCW.Classes.StaticClasses;
using SPZCW.Interfaces;

namespace SPZCWTests.TestsClasses
{
    [TestClass]
    public class MenuesTests
    {
        [TestMethod]
        public void TestGetMainMenuReturnsNotNull()
        {
            Assert.IsNotNull(Menues.GetMainMenu());
        }

        [TestMethod]
        public void TestGetChangeStartTypeMenuReturnsNotNull()
        {
            Assert.IsNotNull(Menues.GetChangeStartTypeMenu());
        }

        [TestMethod]
        public void TestGetFilterMenuReturnsNotNull()
        {
            Assert.IsNotNull(Menues.GetFilterMenu());
        }

        [TestMethod]
        public void TestGetActionsMenuReturnsNotNull()
        {
            //Arange
            var serviceControllerWrapperMock = new Mock<IService>();

            //Act , Assert
            Assert.IsNotNull(Menues.GetActionsMenu(serviceControllerWrapperMock.Object));
        }

        [TestMethod]
        public void TestGetHelpMenuReturnsNotNull()
        {
            Assert.IsNotNull(Menues.GetHelpMenu());
        }
    }
}
```

PathTreeTests.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using SPZCW.Classes.StaticClasses.SpectreConsoleObjects;

namespace SPZCWTests.TestsClasses.StaticClassesTests.SpectreConsoleObjectsTests
{
```

```
    [TestClass]
    public class PathTreeTests
    {
        [TestMethod]
        public void Test()
        {
            Assert.IsNotNull(PathTree.GetServicePathTree("path"));
        }
    }
}
```

TablesTests.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using SPZCW.Classes;
using SPZCW.Classes.StaticClasses;
using System.Collections.Generic;
using System.ServiceProcess;

namespace SPZCWTests.TestsClasses.StaticClassesTests.SpectreConsoleObjectsTests
{
    [TestClass]
    public class TablesTests
    {
        [TestMethod]
        public void TestGetServicesTableByStatusReturnsNotNull()
        {

Assert.IsNotNull(Tables.GetServicesTableByStatus(ServiceControllerStatus.Running));

Assert.IsNotNull(Tables.GetServicesTableByStatus(ServiceControllerStatus.Stopped));

Assert.IsNotNull(Tables.GetServicesTableByStatus(ServiceControllerStatus.Paused));

Assert.IsNotNull(Tables.GetServicesTableByStatus(ServiceControllerStatus.StartPending));

Assert.IsNotNull(Tables.GetServicesTableByStatus(ServiceControllerStatus.StopPending));

Assert.IsNotNull(Tables.GetServicesTableByStatus(ServiceControllerStatus.ContinuePending));

Assert.IsNotNull(Tables.GetServicesTableByStatus(ServiceControllerStatus.PausePending));
        }

        [TestMethod]
        public void TestGetAllServicesTableReturnsNotNull()
        {
            Assert.IsNotNull(Tables.GetAllServicesTable());
        }

        [TestMethod]
        public void TestGetFilteredServicesTableReturnsNotNull()
        {
            //Arrange
            var filterSettings = new FilterSettings(new List<string>());
```

```
            //Act, Assert
            Assert.IsNotNull(Tables.GetFilteredServicesTable(filterSettings));
        }
    }
}
```

MessagesTests.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Moq;
using SPZCW.Classes.StaticClasses;
using SPZCW.Interfaces;
using System.ServiceProcess;

namespace SPZCWTests
{
    [TestClass]
    public class MessagesTests
    {
        [TestMethod]
        public void TestServicesWithStatusReturnsNotNull()
        {
            //Act
            string result1 = Messages.ServicesWithStatus(ServiceControllerStatus.Running);
            string result2 = Messages.ServicesWithStatus(ServiceControllerStatus.Stopped);
            string result3 = Messages.ServicesWithStatus(ServiceControllerStatus.Paused);
            string result4 =
Messages.ServicesWithStatus(ServiceControllerStatus.PausePending);
            string result5 =
Messages.ServicesWithStatus(ServiceControllerStatus.StartPending);
            string result6 = Messages.ServicesWithStatus(ServiceControllerStatus.StopPending);
            string result7 =
Messages.ServicesWithStatus(ServiceControllerStatus.ContinuePending);

            //Assert
            Assert.IsNotNull(result1);
            Assert.IsNotNull(result2);
            Assert.IsNotNull(result3);
            Assert.IsNotNull(result4);
            Assert.IsNotNull(result5);
            Assert.IsNotNull(result6);
            Assert.IsNotNull(result7);
        }

        [TestMethod]
        public void TestServiceInfoHelpReturnsNotNull()
        {
            //Arange
            var serviceControllerWrapperMock = new Mock<IService>();

            //Act , Assert
            Assert.IsNotNull(Messages.ServiceInfo(serviceControllerWrapperMock.Object));
        }

        [TestMethod]
```

```
        public void TestServiceInfoReturnsCorrrectValue()
        {
            //Arrange
            var serviceControllerWrapperMock = new Mock<IService>();
            serviceControllerWrapperMock.Setup(x => x.Description).Returns("Description");
            serviceControllerWrapperMock.Setup(x => x.DisplayName).Returns("DisplayName");
            serviceControllerWrapperMock.Setup(x => x.DisplayName).Returns("ServiceName");
            serviceControllerWrapperMock.Setup(x => x.DisplayName).Returns("MachineName");
            serviceControllerWrapperMock.Setup(x =>
x.ServiceType).Returns(ServiceType.KernelDriver);
            serviceControllerWrapperMock.Setup(x =>
x.StartType).Returns(ServiceStartMode.Automatic);
            serviceControllerWrapperMock.Setup(x => x.Path).Returns("Path");
            serviceControllerWrapperMock.Setup(x =>
x.Status).Returns(ServiceControllerStatus.Running);
            serviceControllerWrapperMock.Setup(x => x.CanStop).Returns(true);

            string description = $"\nDescription
:\t{serviceControllerWrapperMock.Object.Description}\n\r";
            string displayName = $"DisplayName
:\t{serviceControllerWrapperMock.Object.DisplayName}\n\r";
            string serviceName = $"ServiceName
:\t{serviceControllerWrapperMock.Object.ServiceName}\n\r";
            string machineName = $"MachineName
:\t{serviceControllerWrapperMock.Object.MachineName}\n\r";
            string serviceType = $"Service Type
:\t{serviceControllerWrapperMock.Object.ServiceType}\n\r";
            string startType =   $"Start Type
:\t{serviceControllerWrapperMock.Object.StartType}\n\r";
            string servicePath = $"Path
:\t{serviceControllerWrapperMock.Object.Path}\n\r";
            string status =      $"Status
:\t{serviceControllerWrapperMock.Object.Status}\n\r";
            string canStop =     $"Can stop
:\t{serviceControllerWrapperMock.Object.CanStop}\n\r";

            string expected = description + displayName + serviceName + machineName + serviceType
+ servicePath + startType + status + canStop;
            //Act
            string actual = Messages.ServiceInfo(serviceControllerWrapperMock.Object);

            //Assert
            Assert.AreEqual(expected, actual);
        }

        [TestMethod]
        public void TestServicesWithStatusReturnsCorrectValueWithArguments()
        {
            //Arrange
            string expected1 = "Services with status \"Running\":";
            string expected2 = "Services with status \"Stopped\":";
            string expected3 = "Services with status \"Paused\":";
            string expected4 = "Services with status \"PausePending\":";
            string expected5 = "Services with status \"StartPending\":";
            string expected6 = "Services with status \"StopPending\":";
            string expected7 = "Services with status \"ContinuePending\":";

            //Act
            string actual1 = Messages.ServicesWithStatus(ServiceControllerStatus.Running);
            string actual2 = Messages.ServicesWithStatus(ServiceControllerStatus.Stopped);
```

```
            string actual3 = Messages.ServicesWithStatus(ServiceControllerStatus.Paused);
            string actual4 =
Messages.ServicesWithStatus(ServiceControllerStatus.PausePending);
            string actual5 =
Messages.ServicesWithStatus(ServiceControllerStatus.StartPending);
            string actual6 = Messages.ServicesWithStatus(ServiceControllerStatus.StopPending);
            string actual7 =
Messages.ServicesWithStatus(ServiceControllerStatus.ContinuePending);

            //Assert
            Assert.AreEqual(expected1, actual1);
            Assert.AreEqual(expected2, actual2);
            Assert.AreEqual(expected3, actual3);
            Assert.AreEqual(expected4, actual4);
            Assert.AreEqual(expected5, actual5);
            Assert.AreEqual(expected6, actual6);
            Assert.AreEqual(expected7, actual7);
        }

        [TestMethod]
        public void TestErrorDontReturnsNull()
        {
            Assert.IsNotNull(Messages.Error("test"));
        }
    }
}
```

FilterSettingsTests.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using SPZCW.Classes;
using SPZCW.Enumerations;
using System.Collections.Generic;
using System.ServiceProcess;

namespace SPZCWTests
{
    [TestClass]
    public class FilterSettingsTests
    {
        [TestMethod]
        public void TestConstructorCreatesCorrectSizesOfLists()
        {
            //Arrange
            List<string> choices = new List<string>();
            choices.Add("Running");
            choices.Add("Other");

            choices.Add("Automatic");

            choices.Add("Another device");
            choices.Add("Localhost");

            //Act
            FilterSettings filterSettings = new FilterSettings(choices);
```

```
            //Assert
            Assert.AreEqual(6, filterSettings.Statuses.Count);
            Assert.AreEqual(1, filterSettings.StartModes.Count);
            Assert.AreEqual(2, filterSettings.Locations.Count);
        }

        [TestMethod]
        public void TestConstructorCreatesCorrectLists()
        {
            //Arrange
            List<string> choices = CreateListWithChoices();
            //Act
            FilterSettings filterSettings = new FilterSettings(choices);
            bool correctFilling = CheckCorrectFilterSettingsFilling(filterSettings);

            //Assert
            Assert.IsTrue(correctFilling);
        }

        private List<string>  CreateListWithChoices()
        {
            List<string> choices = new List<string>();
            choices.Add("Running");
            choices.Add("Stopped");
            choices.Add("Other");

            choices.Add("Automatic");
            choices.Add("Manual");
            choices.Add("Disabled");
            choices.Add("System");
            choices.Add("Boot");

            choices.Add("Another device");
            choices.Add("Localhost");

            return choices;
        }

        private bool CheckCorrectFilterSettingsFilling(FilterSettings filterSettings)
        {
            if (filterSettings.Locations.Contains(ServiceLocation.AnotherDevice)
                && filterSettings.Locations.Contains(ServiceLocation.LocalHost)
                && filterSettings.StartModes.Contains(ServiceStartMode.Automatic)
                && filterSettings.StartModes.Contains(ServiceStartMode.Disabled)
                && filterSettings.StartModes.Contains(ServiceStartMode.Boot)
                && filterSettings.StartModes.Contains(ServiceStartMode.Manual)
                && filterSettings.StartModes.Contains(ServiceStartMode.System)
                && filterSettings.Statuses.Contains(ServiceControllerStatus.Running)
                && filterSettings.Statuses.Contains(ServiceControllerStatus.Stopped)
                && filterSettings.Statuses.Contains(ServiceControllerStatus.ContinuePending)
                && filterSettings.Statuses.Contains(ServiceControllerStatus.Paused)
                && filterSettings.Statuses.Contains(ServiceControllerStatus.StartPending)
                && filterSettings.Statuses.Contains(ServiceControllerStatus.StartPending)
                && filterSettings.Statuses.Contains(ServiceControllerStatus.StopPending))
            {
                return true;
            }

            return false;
```

```
        }
    }
}
```

ServiceTests.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Moq;
using SPZCW;
using SPZCW.Interfaces;
using System;
using System.ServiceProcess;

namespace SPZCWTests
{
    [TestClass]
    public class ServiceTests
    {
        [TestMethod]
        public void TestDisplayNameReturnsCorrectValue()
        {
            // Arrange
            string expected = "DisplayName";
            var serviceControllerWrapperMock = new Mock<IServiceController>();
            serviceControllerWrapperMock.Setup(x => x.DisplayName).Returns(expected);

            var service = new Service(serviceControllerWrapperMock.Object);

            // Act
            var actual = service.DisplayName;

            //Assert
            Assert.AreEqual(expected, actual);
        }

        [TestMethod]
        public void TestServiceNameReturnsCorrectValue()
        {
            // Arrange
            string expected = "ServiceName";
            var serviceControllerWrapperMock = new Mock<IServiceController>();
            serviceControllerWrapperMock.Setup(x => x.ServiceName).Returns(expected);

            var service = new Service(serviceControllerWrapperMock.Object);

            // Act
            var actual = service.ServiceName;

            //Assert
            Assert.AreEqual(expected, actual);
        }

        [TestMethod]
        public void TestMachineNameReturnsCorrectValue()
        {
```

```
            // Arrange
            string expected = "MachineName";
            var serviceControllerWrapperMock = new Mock<IServiceController>();
            serviceControllerWrapperMock.Setup(x => x.MachineName).Returns(expected);

            var service = new Service(serviceControllerWrapperMock.Object);

            // Act
            var actual = service.MachineName;

            //Assert
            Assert.AreEqual(expected, actual);
        }

        [TestMethod]
        public void TestStatusReturnsCorrectValue()
        {
            // Arrange
            var expected = ServiceControllerStatus.Running;
            var serviceControllerWrapperMock = new Mock<IServiceController>();
            serviceControllerWrapperMock.Setup(x =>
x.Status).Returns(ServiceControllerStatus.Running);

            var service = new Service(serviceControllerWrapperMock.Object);

            // Act
            var actual = service.Status;

            //Assert
            Assert.AreEqual(expected, actual);
        }

        [TestMethod]
        public void TestStartTypeReturnsCorrectValue()
        {
            // Arrange
            var expected = ServiceStartMode.Automatic;
            var serviceControllerWrapperMock = new Mock<IServiceController>();
            serviceControllerWrapperMock.Setup(x =>
x.StartType).Returns(ServiceStartMode.Automatic);

            var service = new Service(serviceControllerWrapperMock.Object);

            // Act
            var actual = service.StartType;

            //Assert
            Assert.AreEqual(expected, actual);
        }

        [TestMethod]
        public void TestServiceTypeReturnsCorrectValue()
        {
            // Arrange
            var expected = ServiceType.KernelDriver;
            var serviceControllerWrapperMock = new Mock<IServiceController>();
            serviceControllerWrapperMock.Setup(x =>
x.ServiceType).Returns(ServiceType.KernelDriver);

            var service = new Service(serviceControllerWrapperMock.Object);
```

```
            // Act
            var actual = service.ServiceType;

            //Assert
            Assert.AreEqual(expected, actual);
        }

        [TestMethod]
        public void TestStartThrowExceptionWhenServiceIsAlreadyRunning()
        {
            // Arrange
            var serviceControllerWrapperMock = new Mock<IServiceController>();
            serviceControllerWrapperMock.Setup(x =>
x.Status).Returns(ServiceControllerStatus.Running);

            var service = new Service(serviceControllerWrapperMock.Object);

            //Act, Assert
            Assert.ThrowsException<Exception>(() => service.Start());
        }

        [TestMethod]
        public void TestStopThrowExceptionWhenServiceIsAlreadyStopped()
        {
            // Arrange
            var serviceControllerWrapperMock = new Mock<IServiceController>();
            serviceControllerWrapperMock.Setup(x =>
x.Status).Returns(ServiceControllerStatus.Stopped);

            var service = new Service(serviceControllerWrapperMock.Object);

            //Act, Assert
            Assert.ThrowsException<Exception>(() => service.Stop());
        }


    }
}
```

SolutionItems

.gitignore

```
## Ignore Visual Studio temporary files, build results, and
## files generated by popular Visual Studio add-ons.

# User-specific files
*.user
*.sln.docstates

# Build results

[Dd]ebug/
[Rr]elease/
```

```
x64/
[Bb]in/
[Oo]bj/

# MSTest test Results
[Tt]est[Rr]esult*/
[Bb]uild[Ll]og.*

*_i.c
*_p.c
*_i.h
*.ilk
*.meta
*.obj
*.pch
*.pdb
*.pgc
*.pgd
*.rsp
*.sbr
*.tlb
*.tli
*.tlh
*.tmp
*.tmp_proj
*.log
*.vspscc
*.vssscc
.builds
*.pidb
*.log
*.svclog
*.scc

# Visual C++ cache files
ipch/
*.aps
*.ncb
*.opensdf
*.sdf
*.cachefile

# Visual Studio profiler
*.psess
*.vsp
*.vspx

# Guidance Automation Toolkit
*.gpState

# ReSharper is a .NET coding add-in
_ReSharper*/
*.[Rr]e[Ss]harper
*.DotSettings.user

# Click-Once directory
publish/

# Publish Web Output
*.Publish.xml
```

```
*.pubxml
*.azurePubxml

# NuGet Packages Directory
## TODO: If you have NuGet Package Restore enabled, uncomment the next line
packages/
## TODO: If the tool you use requires repositories.config, also uncomment the next line
!packages/repositories.config

# Windows Azure Build Output
csx/
*.build.csdef

# Windows Store app package directory
AppPackages/

# Others
sql/
*.Cache
ClientBin/
[Ss]tyle[Cc]op.*
![Ss]tyle[Cc]op.targets
~$*
*~
*.dbmdl
*.[Pp]ublish.xml

*.publishsettings

# RIA/Silverlight projects
Generated_Code/

# Backup & report files from converting an old project file to a newer
# Visual Studio version. Backup files are not needed, because we have git ;-)
_UpgradeReport_Files/
Backup*/
UpgradeLog*.XML
UpgradeLog*.htm

# SQL Server files
App_Data/*.mdf
App_Data/*.ldf

# =========================
# Windows detritus
# =========================

# Windows image file caches
Thumbs.db
ehthumbs.db

# Folder config file
Desktop.ini

# Recycle Bin used on file shares
$RECYCLE.BIN/

# Mac desktop service store files
.DS_Store
```

```
_NCrunch*

# Misc folders
[Bb]in/
[Oo]bj/
[Tt]emp/
[Pp]ackages/
/.artifacts/
/[Tt]ools/
.idea
.DS_Store

# Cakeup
cakeup-x86_64-latest.exe

# .NET Core CLI
/.dotnet/
/.packages/
dotnet-install.sh*
*.lock.json

# Visual Studio
.vs/
.vscode/
launchSettings.json
*.sln.ide/

# Rider
src/.idea/**/workspace.xml
src/.idea/**/tasks.xml
src/.idea/dictionaries
src/.idea/**/dataSources/
src/.idea/**/dataSources.ids
src/.idea/**/dataSources.xml
src/.idea/**/dataSources.local.xml
src/.idea/**/sqlDataSources.xml
src/.idea/**/dynamic.xml
src/.idea/**/uiDesigner.xml

## Ignore Visual Studio temporary files, build results, and
## files generated by popular Visual Studio add-ons.

# User-specific files
*.suo
*.user
*.sln.docstates
*.userprefs
*.GhostDoc.xml
*StyleCop.Cache

# Build results
[Dd]ebug/
[Rr]elease/
x64/
*_i.c
*_p.c
*.ilk
*.meta
*.obj
*.pch
```

```
*.pdb
*.pgc
*.pgd
*.rsp
*.sbr
*.tlb
*.tli
*.tlh
*.tmp
*.log
*.vspscc
*.vssscc
.builds

# Visual Studio profiler
*.psess
*.vsp
*.vspx

# ReSharper is a .NET coding add-in
_ReSharper*

# NCrunch
.*crunch*.local.xml
_NCrunch_*

# NuGet Packages Directory
packages

# Windows
Thumbs.db

*.received.*

node_modules
```

4.ТЕСТУВАННЯ І ВАЛІДАЦІЯ ПРОГРАМНО ЗАБЕЗПЕЧЕННЯ, РОЗРОБКА ІНСТРУКЦІЇ КОРИСТУВАЧА

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ