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

КУРСОВИЙ ПРОЄКТ

з дисципліни “Системне програмне забезпечення”

на тему: “ Розробка утиліти для очистки системи від непотрібних файлів. Розробка програми, що дозволяє очистити систему від тимчасових та непотрібних файлів на комп'ютері з операційною системою Windows ”

Студента 3-го курсу групи КІ-306
123 «Комп’ютерна інженерія»
Хмільовського С.Р.
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

Розробка утиліти для очистки системи від непотрібних файлів. Розробка програми, що дозволяє очистити систему від тимчасових та непотрібних файлів на комп'ютері з операційною системою Windows.

Завдання проекту

1. Аналіз існуючих рішень
2. Розробка архітектури програми
3. Реалізація функціональних можливостей
    a. Сканування системи для виявлення непотрібних файлів
    b. Видалення тимчасових файлів
    c. Очищення кешу
    d. Видалення тимчасових Internet файлів
    e. Видалення файлів з кошика
    f. Пошук та видалення дублікатів файлів
    g. Запис інформації про видалені файли у текстовий документ
    h. Логування подій роботи програми
4. Провести тестування та відлагодження, усунути знайдені проблеми
5. Створити чітку та детальну документацію про проект

Очікувані результати:

Результатом виконання курсового проекту має стати функціональна програма "SimpleCleaner", яка буде здатна ефективно очищувати систему від непотрібних файлів. Програма повинна бути зручною у використанні, надійною та надавати користувачам усі необхідні можливості для очищення системи від тимчасових та непотрібних файлів.

Анотація

Цей курсовий проект присвячений розробці програмного забезпечення для очищення системи від непотрібних файлів. У роботі досліджено основні типи файлів, які можуть накопичуватися на комп'ютері, включаючи тимчасові файли, кеш браузера, дублікати файлів та інші непотрібні файли. У програмі реалізовані основні методи для сканування системи, виявлення та видалення дублікатів файлів, резервного копіювання важливих файлів перед видаленням, а також моніторингу використання дискового простору.

У роботі розглянуто основні методи та інструменти системного програмування, що використовуються для отримання та обробки системної інформації на платформі Windows. Описано процес розробки утиліти, включаючи вибір мови програмування, структуру програми, реалізацію основних функцій та інтерфейс користувача.

Розроблена утиліта має зручний графічний інтерфейс, що дозволяє користувачам легко очищувати систему від сміттєвих файлів. Особлива увага приділена ефективності роботи утиліти та мінімізації її впливу на продуктивність системи.

Результати тестування показують, що утиліта успішно виконує поставлені завдання та може бути корисним інструментом для очищення дискового простору.

Зміст

Вступ

Розробка програмного забезпечення для очищення системи від непотрібних файлів є важливою складовою підтримки ефективності та продуктивності комп'ютерних систем. У сучасному світі, де обсяги збережених даних постійно зростають, необхідність у регулярному очищенні системи стає все більш актуальною.

По-перше, непотрібні файли, такі як тимчасові файли, залишки від встановлених та видалених програм, а також дублікати файлів, можуть значно знижувати продуктивність комп'ютера. Вони займають цінний дисковий простір, що може призводити до уповільнення роботи системи та зменшення доступного місця для зберігання важливих даних.

По-друге, регулярне очищення системи допомагає підтримувати оптимальну роботу операційної системи та програмного забезпечення. Видалення непотрібних файлів дозволяє зменшити фрагментацію диска, що позитивно впливає на швидкість доступу до даних та загальну стабільність системи.

По-третє, програми для очищення системи надають користувачам інструменти для управління файлами та простором на диску, що дозволяє ефективніше використовувати ресурси комп'ютера. Це особливо важливо в умовах обмеженого дискового простору, коли кожен мегабайт має значення.

Також, для звичайного користувача важливо мати інтуїтивно зрозуміле та ефективне програмне забезпечення, яке дозволяє швидко та легко очищувати систему від непотрібних файлів, підтримуючи таким чином її продуктивність та забезпечуючи безперебійну роботу.

1. Аналітичний огляд

1.1 Аналіз методів для очищення системи

Для ефективного функціонування комп'ютерної системи важливо періодично проводити її очищення. Це дозволяє зберігати високу продуктивність та забезпечувати достатньо вільного місця на диску. Нижче наведені основні методи очищення системи:

1.1.1 Видалення кеш-файлів: Кеш-файли використовуються для зберігання тимчасових даних, що дозволяє швидко отримувати доступ до часто використовуваної інформації. Однак, з часом ці файли можуть накопичуватись і займати значний обсяг дискового простору. Очищення кешу дозволяє звільнити місце на диску та покращити продуктивність системи.

1.1.2 Видалення тимчасових файлів: Тимчасові файли створюються під час роботи програм та операційної системи для зберігання проміжних даних. Вони зазвичай видаляються автоматично після завершення роботи програм, але деякі з них можуть залишатись на диску. Видалення тимчасових файлів допомагає звільнити місце на диску та запобігти накопиченню непотрібних даних.

1.1.3 Виявлення та видалення файлів-дублікатів: Файли-дублікати – це копії файлів, що зберігаються у різних місцях на диску. Вони займають додатковий дисковий простір і можуть ускладнювати управління файлами. Виявлення та видалення дублікатів допомагає звільнити місце та покращити організацію файлів.

1.1.4 Очищення залишків після видалення програм: Після видалення програм на диску можуть залишатись різні файли та записи в реєстрі, які більше не потрібні системі. Видалення таких залишків забезпечує чистоту системи та звільняє додатковий дисковий простір.

Відомі розробки для очищення системи:

На ринку існує багато утиліт для очищення системи, серед яких популярні наступні:

1.1.1 CCleaner [3]: одна з найпопулярніших утиліт для очищення системи. Вона дозволяє видаляти тимчасові файли, кеш, файли-дублікати та інші непотрібні дані. CCleaner також включає функції оптимізації реєстру та управління автозавантаженням.

Переваги:

- Широкий функціонал: Видалення тимчасових файлів, очищення кешу, оптимізація реєстру, управління автозавантаженням.
- Інтуїтивний інтерфейс: Зручний і простий у використанні.
- Регулярні оновлення: Команда розробників активно підтримує програму, що забезпечує актуальність функцій та сумісність з новими версіями ОС.
- Функція відновлення системи: Перед великими змінами створює точку відновлення, що дозволяє повернутися до попереднього стану системи.

Недоліки:

- Платна версія: Базові функції безкоштовні, але для доступу до всіх можливостей необхідно придбати платну версію.
- Проблеми з приватністю: У минулому були питання щодо збирання даних користувачів без їх згоди.
- Нав'язливі пропозиції: У безкоштовній версії можуть з'являтися нав'язливі пропозиції оновити до платної версії.

1.1.2 BleachBit [2]: відкрита утиліта для очищення системи, яка підтримує широкий спектр операційних систем. Вона дозволяє видаляти тимчасові файли, очищати кеш та видаляти залишки після видалення програм.

Переваги:

- Відкритий код: Програма є відкритою, що дозволяє користувачам перевіряти код на наявність потенційних проблем і вносити власні зміни.
- Безкоштовність: Повністю безкоштовна без жодних прихованих платежів або обмежень у функціоналі.
- Мультиплатформність: Підтримує широкий спектр операційних систем, включаючи Windows і Linux.
- Сильна конфіденційність: Зосереджена на захисті приватності користувачів і видаленні даних без можливості їх відновлення.

Недоліки:

- Інтерфейс: Може бути менш інтуїтивним і зрозумілим для новачків у порівнянні з комерційними аналогами.
- Обмежений функціонал: Не має таких розширених можливостей, як у CCleaner, наприклад, оптимізації реєстру або управління автозавантаженням.
- Менше регулярних оновлень: Порівняно з комерційними програмами оновлюється рідше.

1.1.3 Glary Utilities [1]: набір утиліт для оптимізації та очищення системи. Включає інструменти для видалення непотрібних файлів, управління автозавантаженням, оптимізації реєстру та багато іншого.

Переваги:

- Комплексність: Включає великий набір інструментів для очищення системи, оптимізації реєстру, управління автозавантаженням, відновлення файлів тощо.
- Простий інтерфейс: Дружній і зрозумілий користувацький інтерфейс.
- Багато функцій безкоштовно: Основний набір інструментів доступний безкоштовно.
- Автоматичне обслуговування: Можливість налаштувати автоматичне обслуговування системи за розкладом.

Недоліки:

- Платна версія: Деякі розширені функції доступні тільки в платній версії.
- Нав'язливі пропозиції: В безкоштовній версії є реклама і пропозиції перейти на платну версію.
- Проблеми з сумісністю: Іноді можуть виникати проблеми з сумісністю з новими версіями ОС.

Комерційні та відкриті утиліти для очищення системи:

Комерційні утиліти, як правило, пропонують додаткові функції та підтримку, однак вони зазвичай платні. Наприклад, CCleaner має безкоштовну та платну версію, причому платна версія пропонує розширені функції та автоматичні оновлення. Відкриті утиліти, такі як BleachBit, є безкоштовними та доступними для всіх користувачів, проте вони можуть не мати такої ж широкої підтримки та функціональності, як комерційні продукти.

1.2 Цільова аудиторія

Програма для очищення системи орієнтована на три основні категорії користувачів: звичайні користувачі, IT-фахівці та ентузіасти. Кожна з цих груп має свої унікальні потреби, і програма пропонує відповідні функції для задоволення цих потреб.

1.2.1 Звичайні користувачі

Опис:

Звичайні користувачі – це люди, які використовують комп'ютери для повсякденних завдань, таких як перегляд інтернету, робота з документами, перегляд медіа та спілкування. Вони зазвичай не мають глибоких технічних знань і шукають прості та ефективні рішення для підтримки своїх комп'ютерів у хорошому стані.

Потреби:

- Простота використання програми.
- Швидке та легке видалення непотрібних файлів.
- Збереження продуктивності та швидкості роботи комп'ютера.
- Автоматичне обслуговування системи без необхідності технічних знань.

Функції для задоволення потреб:

- Інтуїтивно зрозумілий інтерфейс користувача, який легко освоїти.
- Автоматичне виявлення та видалення тимчасових файлів, кеш-файлів та залишків від видалених програм.
- Опція автоматичного планування очищення системи для регулярного обслуговування.
- Повідомлення та рекомендації для користувача щодо покращення продуктивності.

1.2.2 IT-фахівці

Опис:

IT-фахівці – це професіонали, які займаються обслуговуванням та підтримкою комп'ютерних систем у корпоративних середовищах. Вони мають глибокі технічні знання і потребують інструментів для ефективного управління великою кількістю комп'ютерів.

Потреби:

- Можливість налаштування параметрів очищення відповідно до специфічних вимог.
- Інструменти для масового обслуговування комп'ютерів у мережі.
- Детальна звітність та моніторинг стану системи.
- Надійні засоби для забезпечення безпеки та стабільності роботи системи.

Функції для задоволення потреб:

- Розширені параметри налаштування очищення для більш точного контролю над процесом.

- Функція сканування та очищення декількох комп'ютерів у мережі.
- Генерація детальних звітів про проведені операції та стан системи.
- Інтеграція з іншими інструментами для управління IT-інфраструктурою.

1.2.3 Ентузіасти

Опис:

Ентузіасти – це користувачі, які цікавляться технікою та комп'ютерними системами. Вони часто експериментують з налаштуваннями своїх комп'ютерів для досягнення максимальної продуктивності та ефективності.

Потреби:

- Гнучкі інструменти для детального налаштування та оптимізації системи.
- Можливість дослідження та аналізу різних аспектів роботи комп'ютера.
- Відстеження продуктивності та виявлення проблем для їх подальшого вирішення.
- Отримання додаткової інформації про стан системи та можливі покращення.

Функції для задоволення потреб:

- Поглиблені інструменти для аналізу та оптимізації системи.
- Опція виведення детальної інформації про файли, що видаляються, та їх вплив на продуктивність.
- Можливість налаштування та запуску скриптів для автоматизації процесів очищення.
- Інтеграція з іншими інструментами для моніторингу та налаштування системи.

Висновок до розділу 1:

Програма для очищення системи розроблена з урахуванням потреб різних категорій користувачів. Для звичайних користувачів пропонується простий та інтуїтивний інтерфейс, що забезпечує легкість використання. IT-фахівці отримують потужні інструменти для налаштування та управління великими мережами комп'ютерів. Ентузіасти мають доступ до гнучких інструментів для детального налаштування та аналізу роботи своїх систем. Завдяки цьому, наша програма є універсальним рішенням для підтримки чистоти та продуктивності комп'ютерних систем.

2. Проектування програми

2.1 Вимоги до програмного забезпечення:

Для ефективної роботи та забезпечення максимальної зручності користувача, програма повинна відповідати певним вимогам як до загальної системи, так і до її функціоналу. Нижче наведено основні вимоги до програмного забезпечення.

- **Вимоги до системи в цілому:**

Дане програмне рішення повинне надавати користувачеві зручний та зрозумілий інтерфейс для очистки “сміттєвих” файлів ОС. Також важливим аспектом буде швидкодія, та стабільність даної програми.

- **Вимоги до функціоналу програми:**

2.1.1 Сканування системи для виявлення непотрібних файлів:

Програма проводить сканування всіх дисків комп'ютера для ідентифікації тимчасових файлів, файлів кешу, застарілих резервних копій та інших непотрібних об'єктів. Слід зазначити, що у наведених нижче директоріях, абсолютно всі файли будуть вважатися сміттєвими. Але користувачеві буде надана можливість вибору типу очистки (очистити певні папки/жодної/усі). Також користувач може задати свій список виключень, у директоріях якого програма не буде проводити сканування. У програмі також буде передбачено користувацький вибір каталогів, у яких проводити сканування, але тоді буде виконуватись лише пошук дублікатів. Також, після завершення сканування на екран буде виведено інформацію про файл, а саме: його назву, розташування, тип та розмір.

2.1.2 Видалення тимчасових файлів:

Після сканування програма видаляє тимчасові файли, які не потрібні для подальшої роботи операційної системи. Тимчасовими будуть вважатися файли, які знаходяться у таких директоріях, як %TEMP%, %TMP%, Temp, Prefetch.

2.1.3 Очищення кешу:

Користувачеві надається можливість очищення кешу, що допомагає покращити швидкодію системи та програм в цілому. Для всіх програм місце розташування кеш-файлів різне, тому на даному етапі необхідно реалізувати очищення кешу як мінімум однієї програми, а саме Windows Store, який знаходиться в директорії: `C:\Users\Ім'я_користувача\AppData\Local\Packages`

2.1.4 Видалення тимчасових Internet файлів:

Забезпечується можливість видалення тимчасових Internet файлів, таких як кеш веб-сторінок, що допомагає звільнити простір на диску. Сміттєвими будуть вважатися, абсолютно всі файли, які знаходяться у даних директоріях: `C:\Users\Ім'я_користувача\AppData\Local\Ім'я_браузера\User Data\Default\Cache.`

2.1.5 Видалення файлів з кошика:

Програма повинна мати можливість остаточного видалення файлів, які знаходяться у кошику. Користувачеві буде надана опціональна можливість очистки кошика, а саме папки “$Recycle. Bin”.

2.1.6 Пошук та видалення дублікатів файлів:

Програма має здійснювати пошук та видалення дублікатів файлів для економії місця на диску. Тут все простіше, немає сенсу перевіряти абсолютно всі файли на ідентичність, тому користувачеві буде надана можливість самостійного вибору папки, у якій він хоче провести сканування на дублікати, та провести їх видалення.

2.1.7 Обрахунок кількості файлів різних типів

Програма має мати функціональність для обрахунку кількості файлів різних типів у зазначеній користувачем директорії. Ця функція дозволяє користувачам отримувати детальну інформацію про структуру файлів у вибраній папці, що може бути корисним для аналізу використання дискового простору та виявлення найбільших споживачів ресурсів.

2.1.8 Запис інформації про видалені файли у текстовий документ:

Передбачається функціонал запису інформації про видалені файли у текстовий документ для подальшого використання або аналізу. Це заміна повноцінному створення резервних копій, адже це досить трудомісткий процес, тому це його спрощена альтернатива, яка буде реалізована у окремій функції.

2.1.9 Логування подій роботи програми:

Програма повинна вести лог подій своєї роботи для відстеження виконаних дій та помилок. Це буде досить корисно розробнику, адже він буде розуміти що саме відбувалось “під капотом” програми під час її тестових запусків.

2.2 Розробка структурної схеми програми

Розробка структурної схеми програми є критично важливим етапом у процесі розробки програмного забезпечення. Вона забезпечує чітке розуміння архітектури програми, сприяє ефективній комунікації та плануванню, а також полегшує процеси обслуговування та оновлення. Завдяки структурній схемі розробники можуть створювати більш якісні, підтримувані та масштабовані програми.

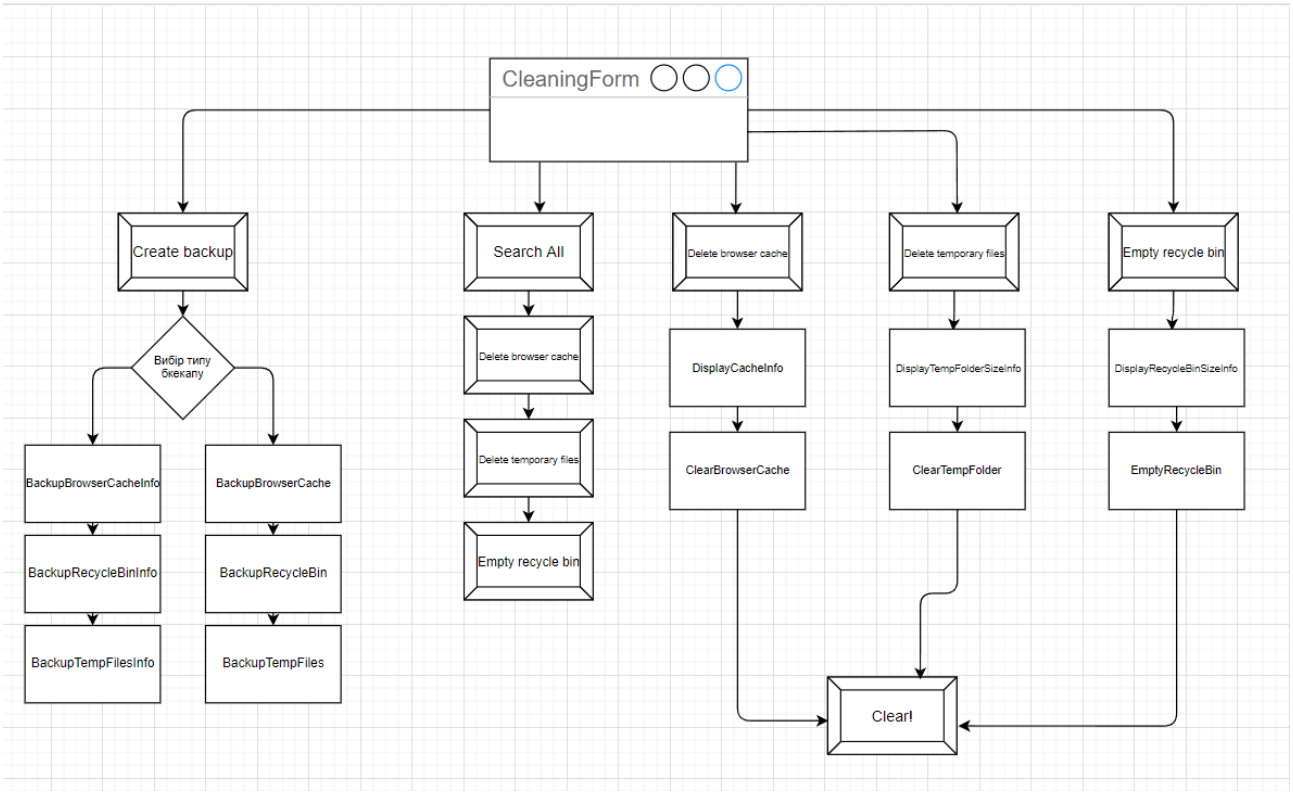

Рис. 2.1 – Діаграма діяльності форми CleaningForm

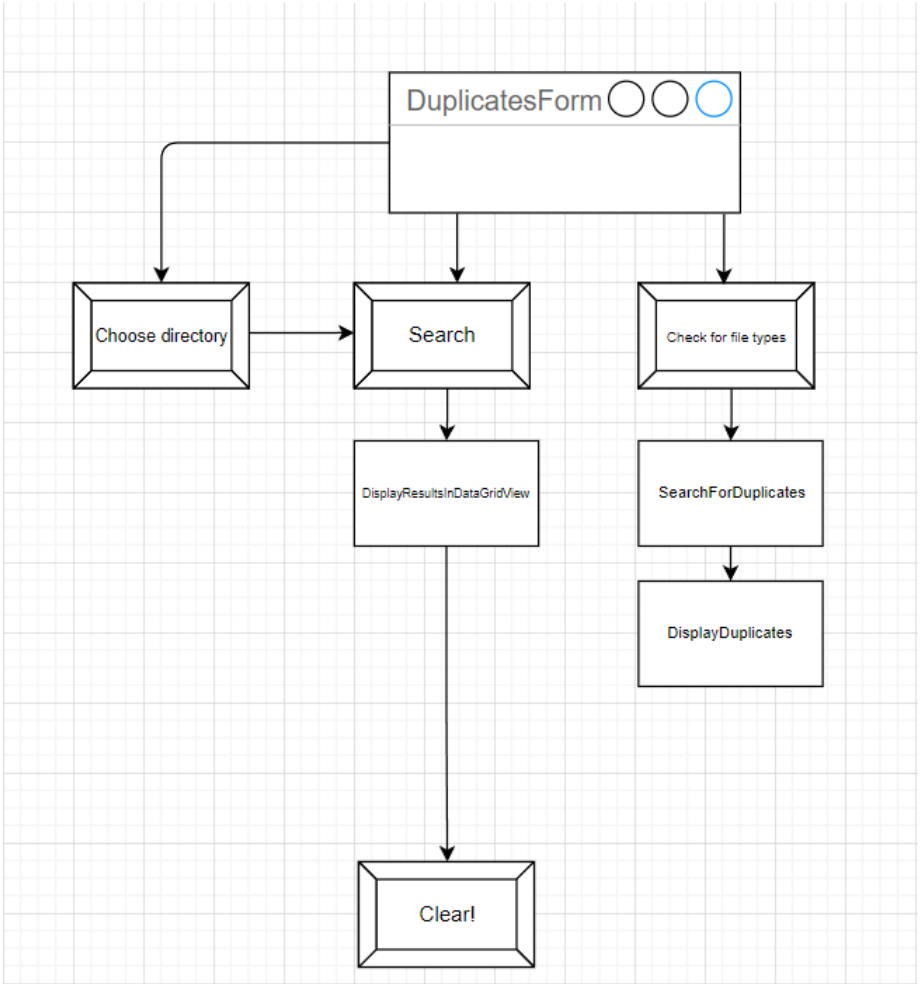

Рис. 2.2 – Діаграма діяльності форми DuplicatesForm

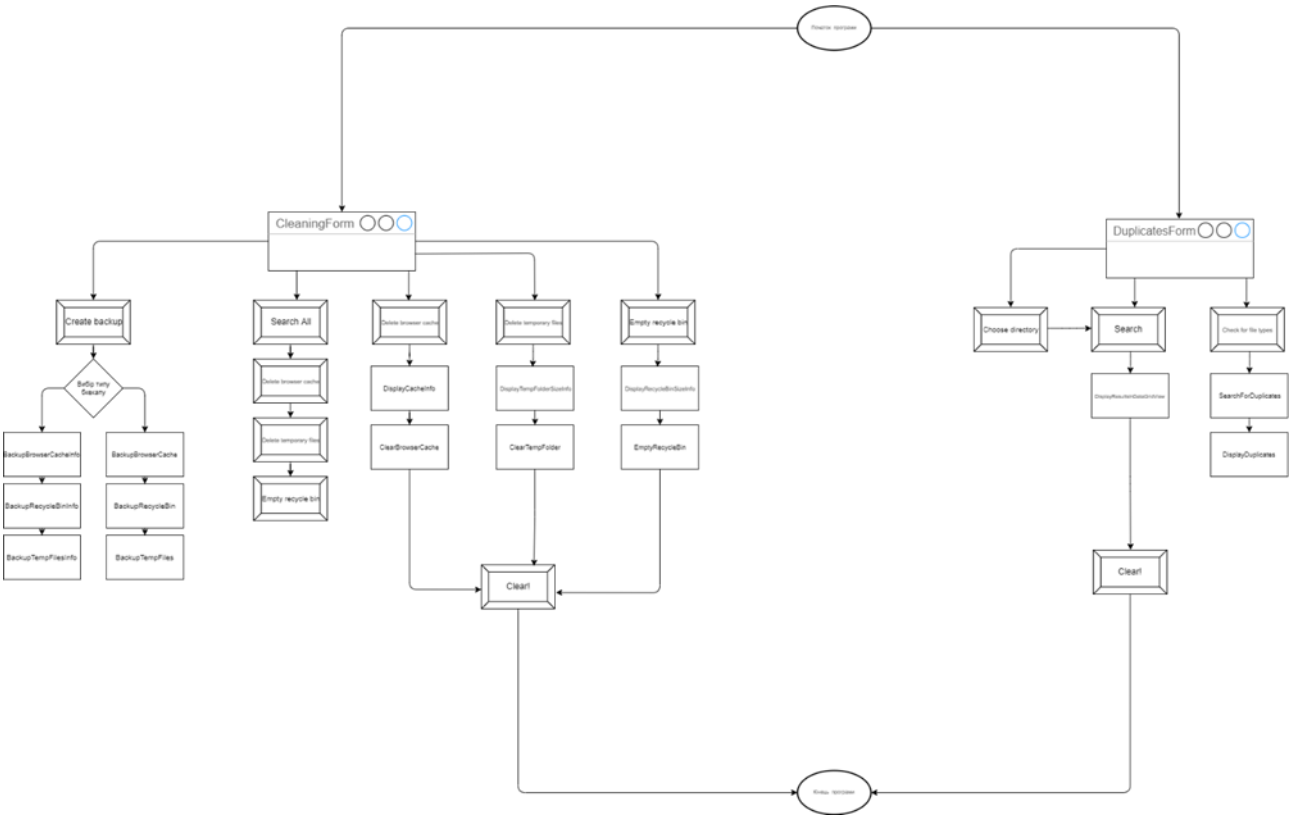

Рис. 2.3 – Діаграма діяльності програми

2.3 Обґрунтування засобів для розробки

C# було обрано для розробки утиліти завдяки своїм численним перевагам, які роблять її найбільш підходящою мовою програмування для цього завдання. C# є об'єктно-орієнтованою мовою з потужною статичною типізацією, що дозволяє ефективно структурувати код, забезпечуючи високу надійність і стабільність програми. Статична типізація C# дозволяє виявляти помилки на етапі компіляції, зменшуючи кількість помилок під час виконання, що є важливою перевагою над динамічно типізованими мовами, такими як Python чи JavaScript, де помилки можуть з'являтися тільки під час виконання програми.

Крім того, C# інтегрується з потужною екосистемою .NET, яка надає широкий спектр бібліотек і інструментів для різних аспектів програмування, включаючи роботу з базами даних, мережевим програмуванням та створенням графічних інтерфейсів. Інші мови, такі як Java, також мають свої екосистеми,

але .NET виділяється своєю тісною інтеграцією з Windows, що є ключовою перевагою для розробки програм, призначених для цієї платформи.

C# також відзначається своєю інтеграцією з Windows Forms та WPF, що дозволяє створювати сучасні та функціональні графічні інтерфейси з мінімальними зусиллями. Інші мови, такі як Python чи Java, також підтримують розробку графічних інтерфейсів, але вони не мають такої тісної інтеграції з Windows, що може ускладнювати процес розробки для цієї платформи.

Ще однією вагомою причиною вибору C# є підтримка Microsoft. Як рідна мова для розробки під платформу Windows, C# отримує регулярні оновлення і нові можливості від Microsoft, що забезпечує постійну актуальність і вдосконалення мови. Інші мови можуть бути більш кросплатформеними, але для розробки під Windows C# є найбільш оптимальним вибором завдяки своїй тісній інтеграції і підтримці від розробників платформи.

Для розробки утиліти використовується середовище розробки Microsoft Visual Studio, яке є потужним і зручним інструментом для роботи з C#. Visual Studio має інтуїтивний інтерфейс, що робить його зручним для розробників будь-якого рівня. Крім того, Visual Studio включає широкий спектр інструментів для розробки, тестування, налагодження та контролю версій, що забезпечує комплексне середовище для створення програмного забезпечення. Можливість розширення Visual Studio за допомогою плагінів дозволяє адаптувати середовище під специфічні потреби розробника, а легка інтеграція з іншими інструментами, такими як Azure DevOps, полегшує керування проектами та спільну роботу.

Альтернативи, такі як Python, JavaScript, та Java, мають свої переваги. Python відзначається простотою синтаксису та широким спектром бібліотек для різних задач. JavaScript є невід'ємною частиною веб-розробки і має сильну екосистему для створення веб-додатків. Java пропонує кросплатформеність та потужні засоби для серверної розробки. Однак, жодна з цих мов не забезпечує такої тісної інтеграції з Windows, як C#, що робить її найбільш підходящою для розробки утиліти, орієнтованої на цю платформу.

Таким чином, вибір C# і Microsoft Visual Studio для розробки утиліти є обґрунтованим і оптимальним рішенням, яке забезпечує надійну, ефективну та зручну розробку програмного забезпечення під платформу Windows. [4], [5], [13]

Вибір технологій:
Для реалізації програмного рішення необхідно вибрати відповідні технології, які забезпечать ефективність і зручність використання програми. Нижче наведено вибрані технології та їх застосування.

- System.IO: Бібліотека для роботи з файловою системою та виконання операцій введення-виведення. Використовується для пошуку та видалення файлів. [9]
- System.Security.Cryptography: Бібліотека для обчислення хеш-функцій файлів. Використовується для виявлення дублікатів файлів. [15]
- System.Windows.Forms: Бібліотека для створення графічного інтерфейсу користувача. Використовується для створення зручного інтерфейсу утиліти. [6], [5], [13]
- System.Diagnostics: Бібліотека для ведення журналів подій та відстеження помилок. Використовується для запису дій програми та відстеження помилок. [14], [10]
- System.Configuration: Бібліотека для роботи з конфігураційними файлами. Використовується для збереження налаштувань програми. [7]

2.4 Високорівневий огляд програми

Програма "SimpleCleaner" розроблена для ефективного очищення системи від непотрібних файлів на операційній системі Windows. Її основна мета - забезпечити користувачам інструменти для видалення зайвих файлів, підвищення продуктивності комп'ютера та звільнення дискового простору. Програма має широкий спектр функцій, включаючи виявлення та видалення тимчасових файлів, кеш-файлів, залишків від видалених програм, а також

дублікати файлів. Нижче наведено детальний опис архітектури та функціональних можливостей програми.

2.5 Архітектура програми

Програма "SimpleCleaner" складається з декількох основних компонентів:

2.5.1 Інтерфейс користувача (UI):

Зручний та інтуїтивно зрозумілий графічний інтерфейс, розроблений за допомогою WinForms, дозволяє користувачам легко взаємодіяти з програмою та налаштовувати параметри очищення.

2.5.2 Модуль сканування:

Цей модуль відповідає за сканування файлової системи, виявлення тимчасових файлів, кеш-файлів, залишків від програм та дублікатів файлів.

2.5.3 Модуль очищення:

Виконує видалення виявлених непотрібних файлів відповідно до налаштувань користувача. Підтримує резервне копіювання файлів перед видаленням.

2.5.4 Модуль планування:

Дозволяє користувачам налаштовувати розклад автоматичного виконання завдань очищення системи.

2.5.5 Модуль логування:

Відповідає за запис у файл логів про всі виконані дії програми, що дозволяє користувачу відстежувати проведені операції та виявляти можливі проблеми.

2.6 Функціональні можливості

2.6.1 Видалення тимчасових файлів

Опис:

Тимчасові файли створюються операційною системою та програмами під час їх роботи для зберігання проміжних даних. З часом ці файли накопичуються та займають значний обсяг дискового простору.

Функціональність:
Сканування системи: Програма сканує визначені папки (наприклад, Temp, Prefetch) та знаходить тимчасові файли.
Видалення файлів: Користувач може вибрати та видалити непотрібні тимчасові файли для звільнення місця на диску.

Імплементація:
Програма визначає шлях до тимчасової папки, сканує її на наявність файлів і видаляє знайдені файли.

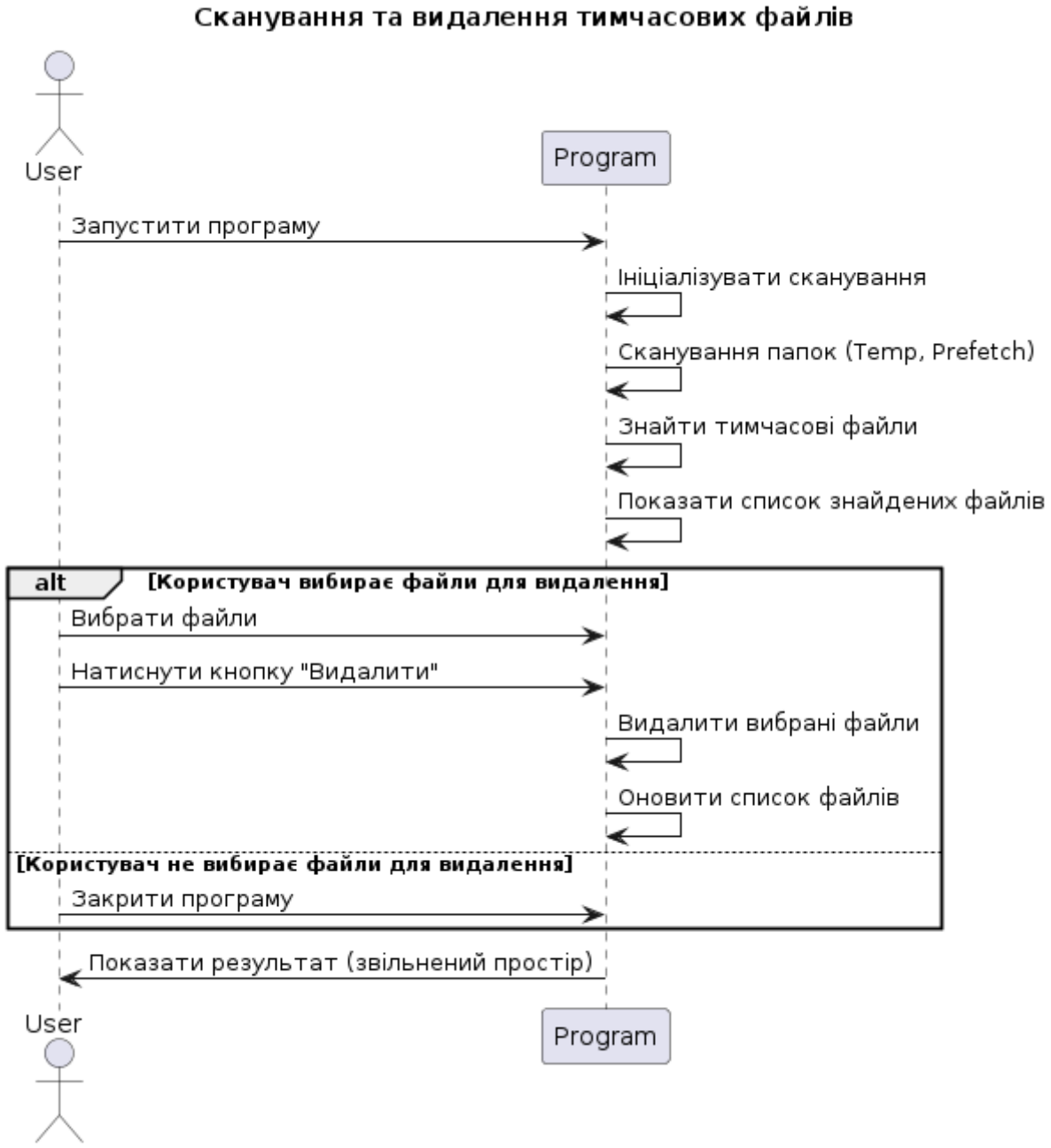

Рис. 2.4 – Діаграма послідовності видалення тимчасових файлів

2.6.2 Очищення кеш-файлів

Опис:
Кеш-файли використовуються програмами та браузерами для прискорення доступу до часто використовуваних даних. Проте з часом кеш може стати надмірним та займати багато дискового простору.

Функціональність:
Виявлення кеш-файлів: Програма ідентифікує кеш-файли різних програм та браузерів.
Видалення кешу: Користувач може очистити кеш для звільнення місця та покращення продуктивності системи.

Імплементація:
Програма містить методи для очищення кешу окремих браузерів, таких як Chrome та Firefox.

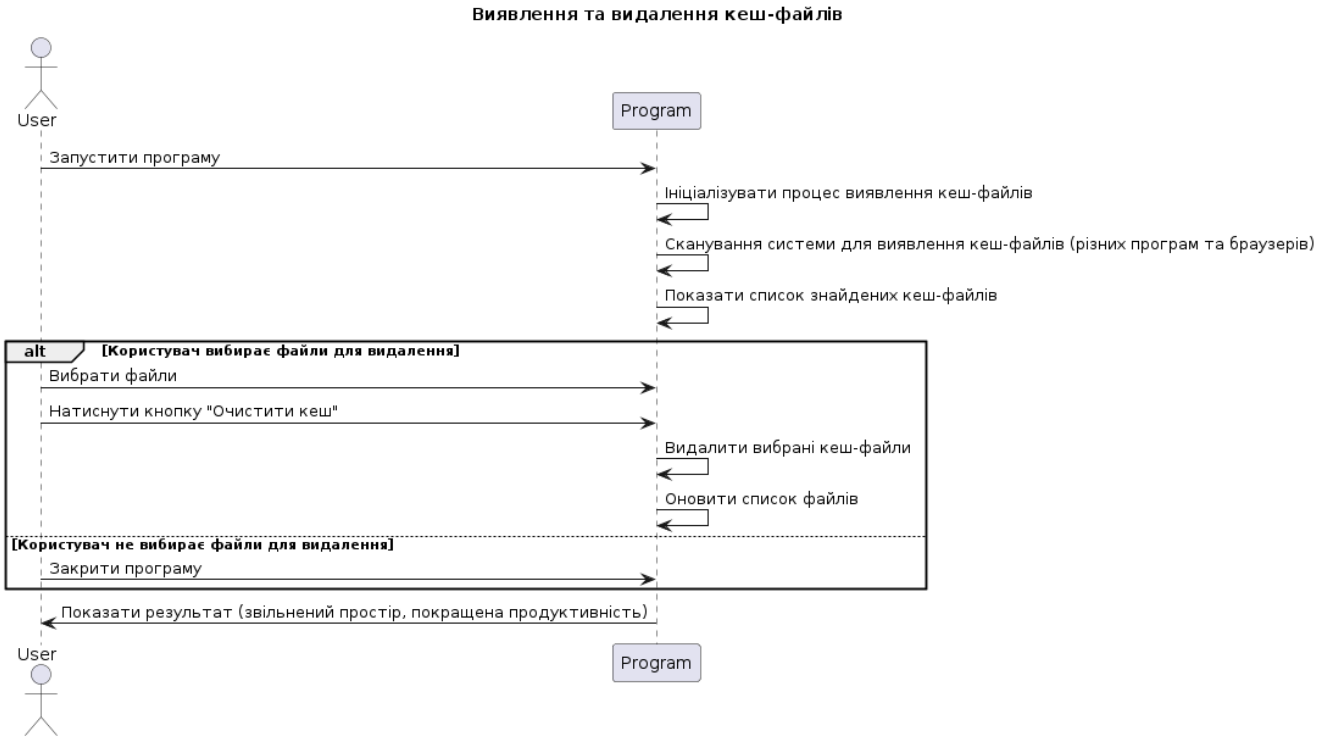

Рис. 2.5 – Діаграма послідовності очищення кеш-файлів

2.6.3 Виявлення та видалення дублікатів файлів

Опис:
Дублікати файлів займають цінний дисковий простір та можуть утруднювати управління файлами.

Функціональність:
Сканування на дублікати: Програма використовує алгоритми хешування для пошуку дублікатів файлів у визначених папках.
Видалення дублікатів: Користувач може переглянути список дублікатів та видалити непотрібні копії.

Імплементація:
Програма використовує хешування для виявлення дублікатів файлів шляхом порівняння хешів файлів.

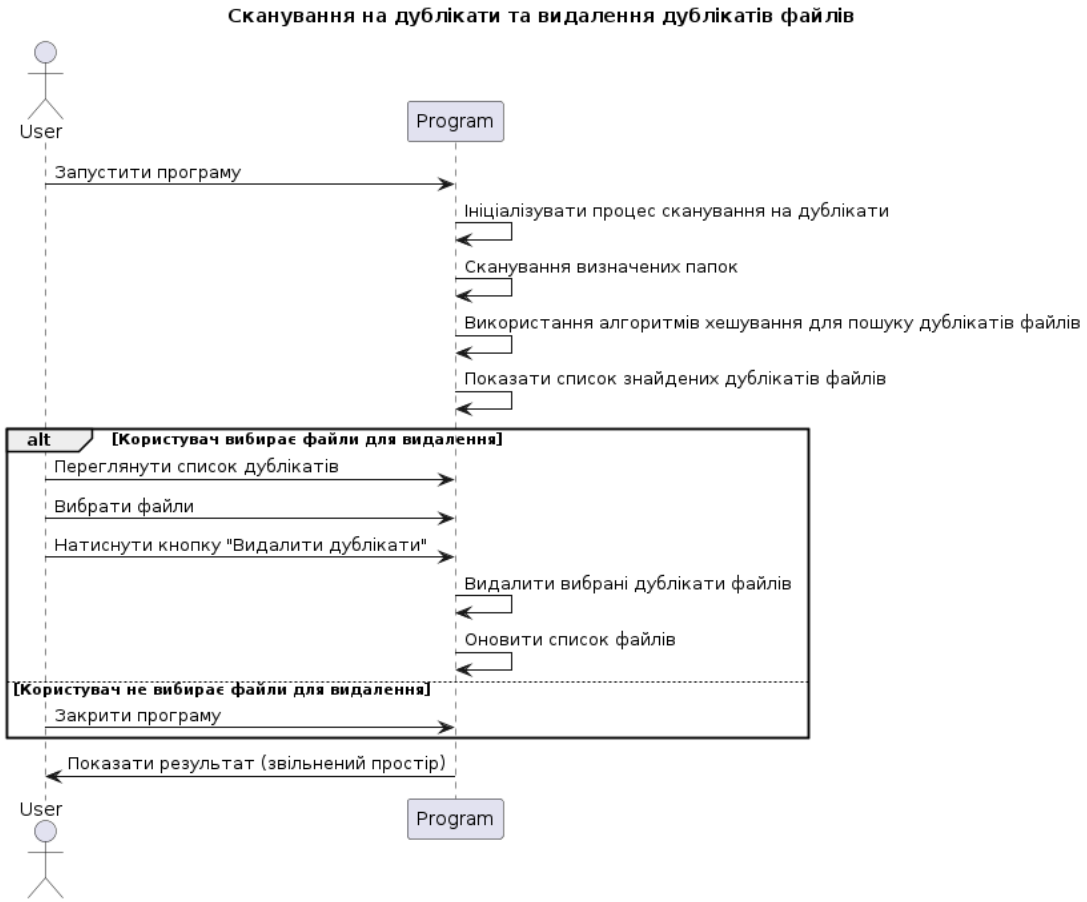

Рис. 2.6 – Діаграма послідовності виявлення та видалення дублікатів файлів

2.6.4 Резервне копіювання файлів перед видаленням

Опис:
Для забезпечення безпеки даних програма пропонує можливість створення резервних копій файлів перед їх видаленням.

Функціональність:
Створення резервних копій: Перед видаленням файлів програма створює їх копії у визначеній папці.
Відновлення файлів: Користувач може відновити видалені файли з резервних копій у разі необхідності.

Імплементація:
Програма копіює вибрані файли до резервної папки перед їх видаленням.

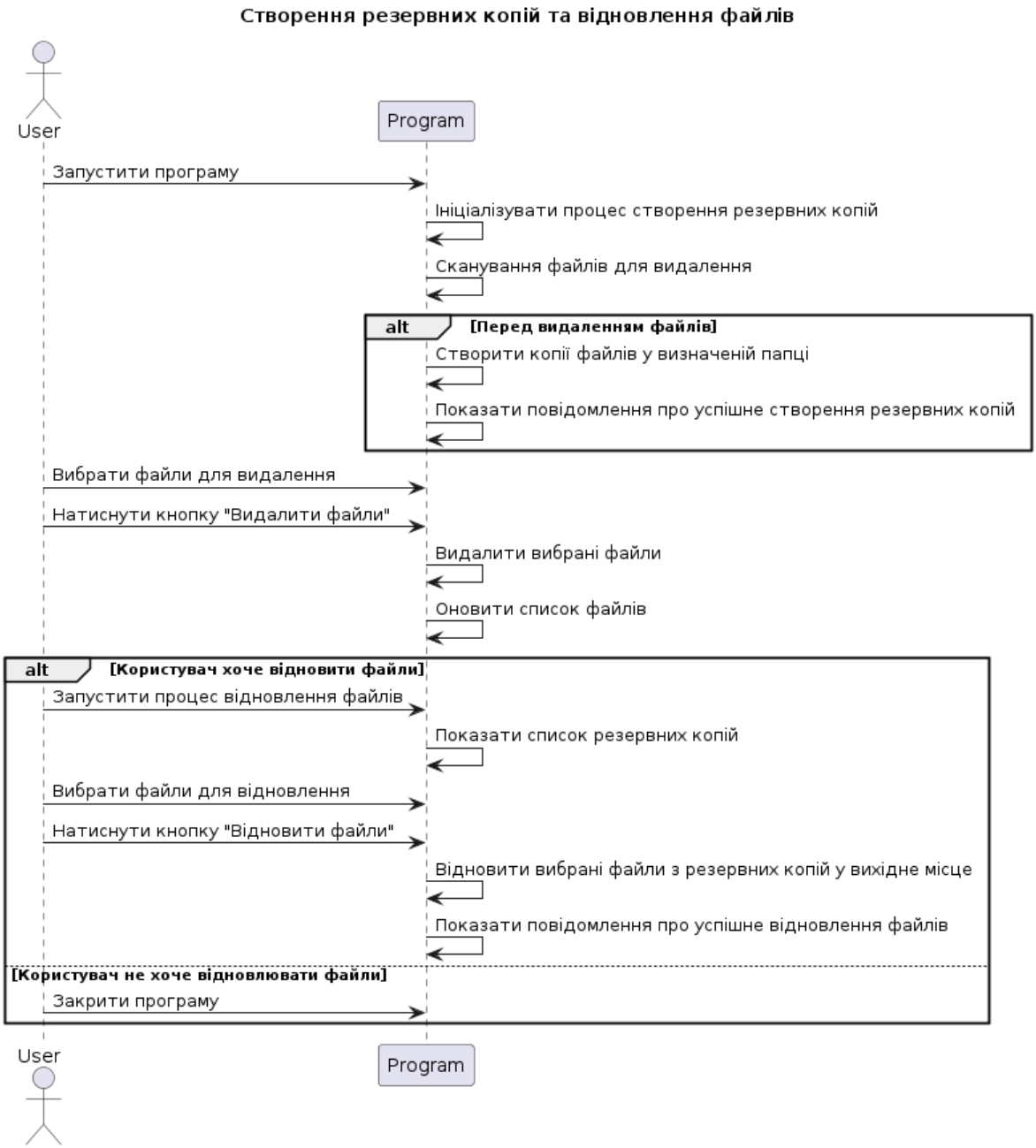

Рис. 2.7 – Діаграма послідовності резервного копіювання файлів

2.6.5 Логування дій програми

Опис:

Для забезпечення прозорості та можливості відстеження дій програма веде журнал всіх виконаних операцій.

Функціональність:

Запис логів: Програма записує інформацію про виконані дії, включаючи час виконання та деталі операцій.

Перегляд логів: Користувач може переглядати логи для аналізу виконаних дій та виявлення можливих проблем.

Імплементація:
Програма записує кожну дію до текстового файлу з зазначенням часу виконання та опису дії.

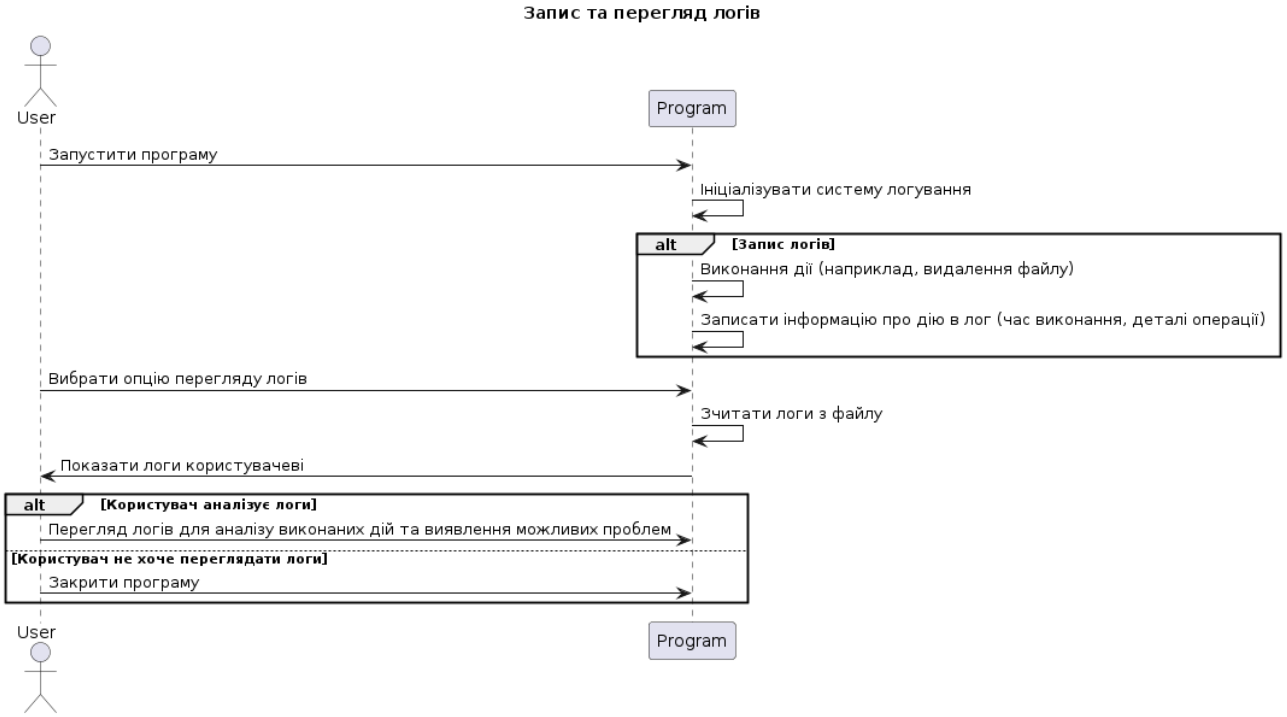

Рис. 2.8 – Діаграма послідовності логування дій програми

2.6.6 Обрахунок кількості файлів з їх типами

Опис:
Ця функція дозволяє користувачам отримати детальну інформацію про кількість файлів різних типів у вибраній директорії. Вона корисна для аналізу структури файлів на диску та виявлення найбільш поширених типів файлів.

Функціональність:
Сканування директорій: Програма сканує вибрану директорію та всі її піддиректорії для збору інформації про файли.
Класифікація файлів: Файли класифікуються за типами (розширеннями) і підраховується кількість файлів кожного типу.
Вивід результатів: Результати сканування відображаються у вигляді таблиці або списку, де для кожного типу файлів вказується їх кількість та загальний розмір.

Імплементація:
Програма сканує вибрану директорію, підраховує кількість файлів кожного типу та відображає результати у вигляді таблиці або списку.

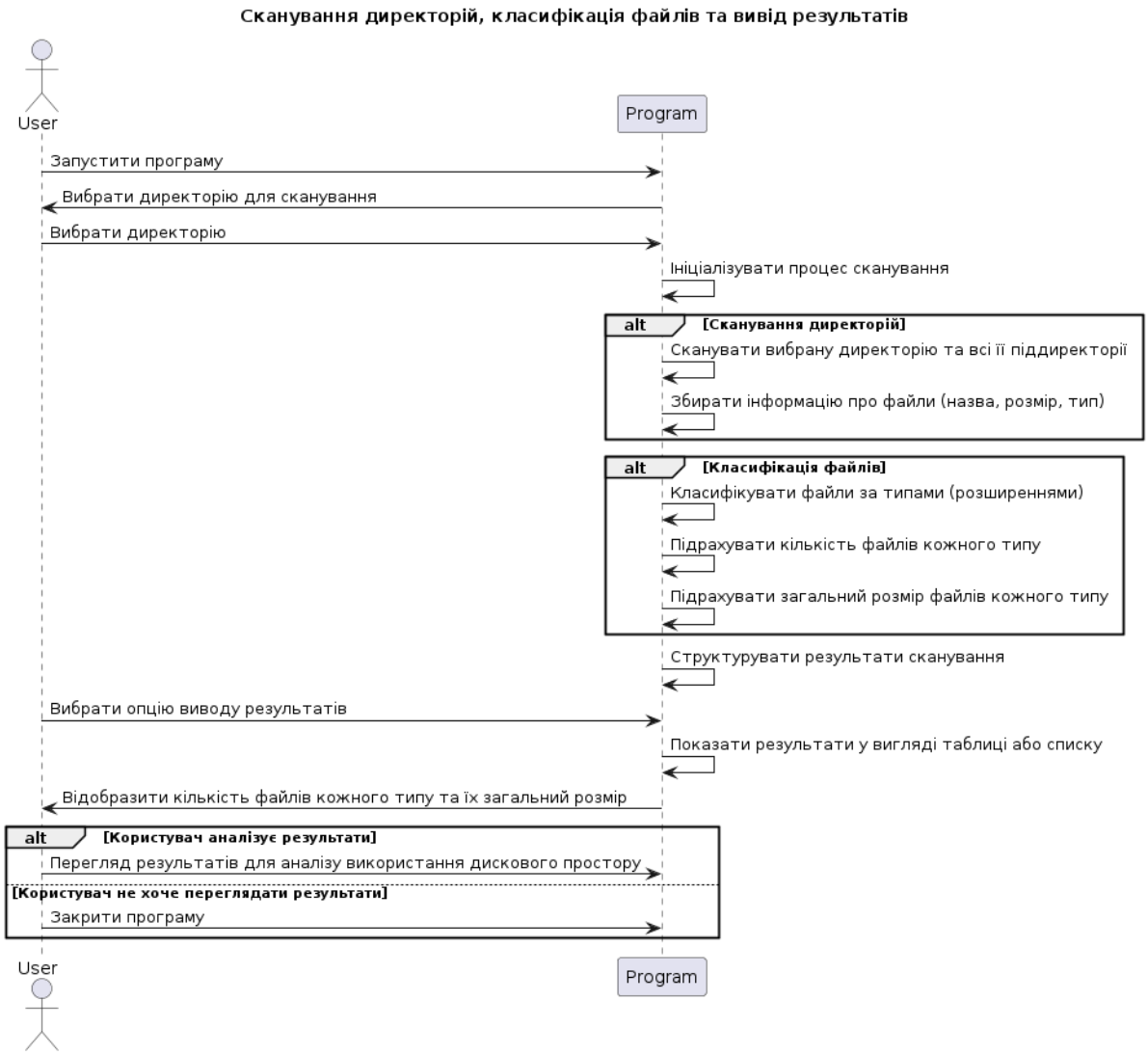

Рис. 2.9 – Діаграма послідовності обрахунку кількості файлів з їх типами

2.6.7 Зручний графічний інтерфейс користувача (UI)

Опис:

Програма має інтуїтивно зрозумілий та зручний графічний інтерфейс, що дозволяє користувачам легко взаємодіяти з програмою та налаштовувати параметри очищення.

Функціональність:

Основне вікно: Інтерфейс поділений на логічні секції для різних функцій очищення.

Налаштування користувача: Можливість налаштування параметрів очищення та планування завдань.

Повідомлення та сповіщення: Програма інформує користувача про завершення завдань та наявність проблем через спливаючі повідомлення

Імплементація:

Програма створює основне вікно з логічними секціями для різних функцій очищення, налаштування користувача та повідомлень.

Висновок до розділу 2:

Ці функціональні можливості роблять SimpleCleaner потужним інструментом для підтримки чистоти та оптимізації роботи операційної системи Windows. Програма підходить для широкого кола користувачів, від звичайних домашніх користувачів до IT-фахівців, які прагнуть підтримувати свої системи у найкращому стані.

3. Реалізація

3.1 Високорівневий опис реалізації

Для реалізації проектованого рішення програми "SimpleCleaner" я використав мову програмування C# та платформу .NET Framework, що забезпечує високу продуктивність і легку інтеграцію з операційною системою Windows.

Використання бібліотек та фреймворків:

System.IO: Ця бібліотека використовується для роботи з файловою системою. Вона дозволяє сканувати директорії, читати та записувати файли, отримувати інформацію про файли (розмір, дату створення, розширення тощо). [9]

System.Windows.Forms: Ця бібліотека забезпечує побудову графічного інтерфейсу користувача (GUI). Використовується для створення форм, кнопок, таблиць (DataGridView) та інших елементів інтерфейсу, що дозволяють користувачам взаємодіяти з програмою. [5]

System.Security.Cryptography: Використовується для хешування файлів з метою виявлення дублікатів. За допомогою алгоритму MD5 можна обчислити хеш для кожного файлу та порівнювати їх для визначення ідентичності файлів. [15]

Основні функціональні можливості:

- Сканування директорій
- Класифікація файлів
- Виявлення дублікатів
- Очищення тимчасових файлів та кешу
- Бекап дублікатів
- Логування подій

3.2 Реалізація інтерфейсу користувача

Графічний інтерфейс користувача (GUI)

Для створення зручного та інтуїтивно зрозумілого інтерфейсу користувача в програмі "SimpleCleaner" було використано можливості бібліотеки System.Windows.Forms. Ця бібліотека надає широкий спектр інструментів для розробки інтерфейсів, забезпечуючи при цьому адаптацію під різні теми оформлення та розміри екранів.

Обґрунтування дизайну кнопок та їх призначення:

Рис. 3.1 – Кнопка пошуку всіх сміттєвих файлів

Запускає процес повного сканування системи для виявлення всіх непотрібних файлів, включаючи тимчасові файли, кеші, залишки від видалених програм та інші невикористовувані дані.

Рис. 3.2 – Кнопка пошуку кеш-файлів браузерів

Ініціює спеціалізоване сканування для виявлення кеш-файлів, створених браузерами. Це дозволяє видаляти тимчасові інтернет-файли, які можуть займати значний дисковий простір та сповільнювати роботу браузера.

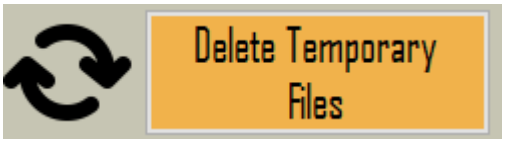

Рис. 3.3 – Кнопка видалення тимчасових файлів системи

Виконує очищення системних тимчасових файлів, які створюються операційною системою та програмами під час їх роботи. Видалення цих файлів допомагає звільнити дисковий простір і покращити продуктивність системи.

Рис. 3.4 – Кнопка для очищення корзини

Очищує корзину Windows, видаляючи всі файли, які були переміщені до неї. Це звільняє дисковий простір та забезпечує безповоротне видалення файлів, які користувач більше не потребує.

Рис. 3.5 – Кнопка створення бекапу файлів перед їх видаенням

Створює резервні копії вибраних файлів перед їх видаленням. Це забезпечує додаткову безпеку, дозволяючи відновити файли у випадку помилкового видалення або потреби в них у майбутньому.

Рис. 3.6 – Кнопка для очищення сміттєвих файлів

Видаляє всі знайдені сміттєві файли після завершення процесу сканування. Ця функція дозволяє швидко звільнити дисковий простір та покращити загальну продуктивність системи.

Рис. 3.7 – Кнопка для переходу в форму роботи з дублікатами

Перемикає користувача до спеціальної форми, призначеної для управління та видалення файлів-дублікатів. Це дозволяє зручно знаходити і видаляти повторювані файли, що займають додатковий дисковий простір.

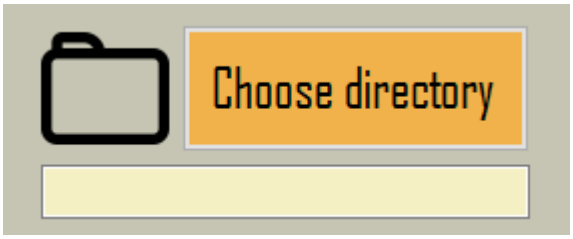

Рис. 3.8 – Кнопка вибору директорії сканування на наявність дублікатів

Відкриває діалогове вікно для вибору директорії, яка буде скануватись на наявність файлів-дублікатів. Це дозволяє користувачу визначити, які саме папки мають бути перевірені на дублікати.

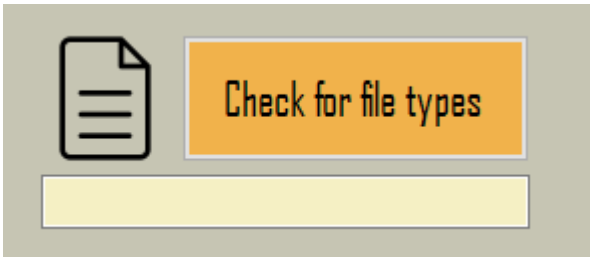

Рис. 3.9 – Кнопка для перевірки кількості файлів різних типів у папці

Запускає аналіз вибраної директорії для підрахунку файлів різних типів. Це допомагає користувачеві зрозуміти розподіл файлів у директорії та визначити, які файли займають найбільше місця.

Рис. 3.10 – Кнопка для отримання відповідей на найчастіші питання

Відкриває розділ з відповідями на часто задавані питання (FAQ). Це забезпечує користувачам швидкий доступ до інформації про використання програми та вирішення можливих проблем.

Рис. 3.11 – Кнопка "Назад" для навігації між формами

Повертає користувача до попередньої форми або головного меню програми, що забезпечує зручну навігацію всередині програми.

Загальний вигляд GUI:

Рис. 3.12 - Вигляд головного меню програми, а саме форми CleaningForm

Як бачимо, користувач має можливість провести сканування всіх “сміттєвих” файлів, або видалити окремі їх типи. Також, у боковому вікні (DataGridView) виводиться інформація про дані файли. Також у користувача присутня можливість створити бекап файлів перед їх видаленням. Ось короткий опис функціоналу кожної клавіші:

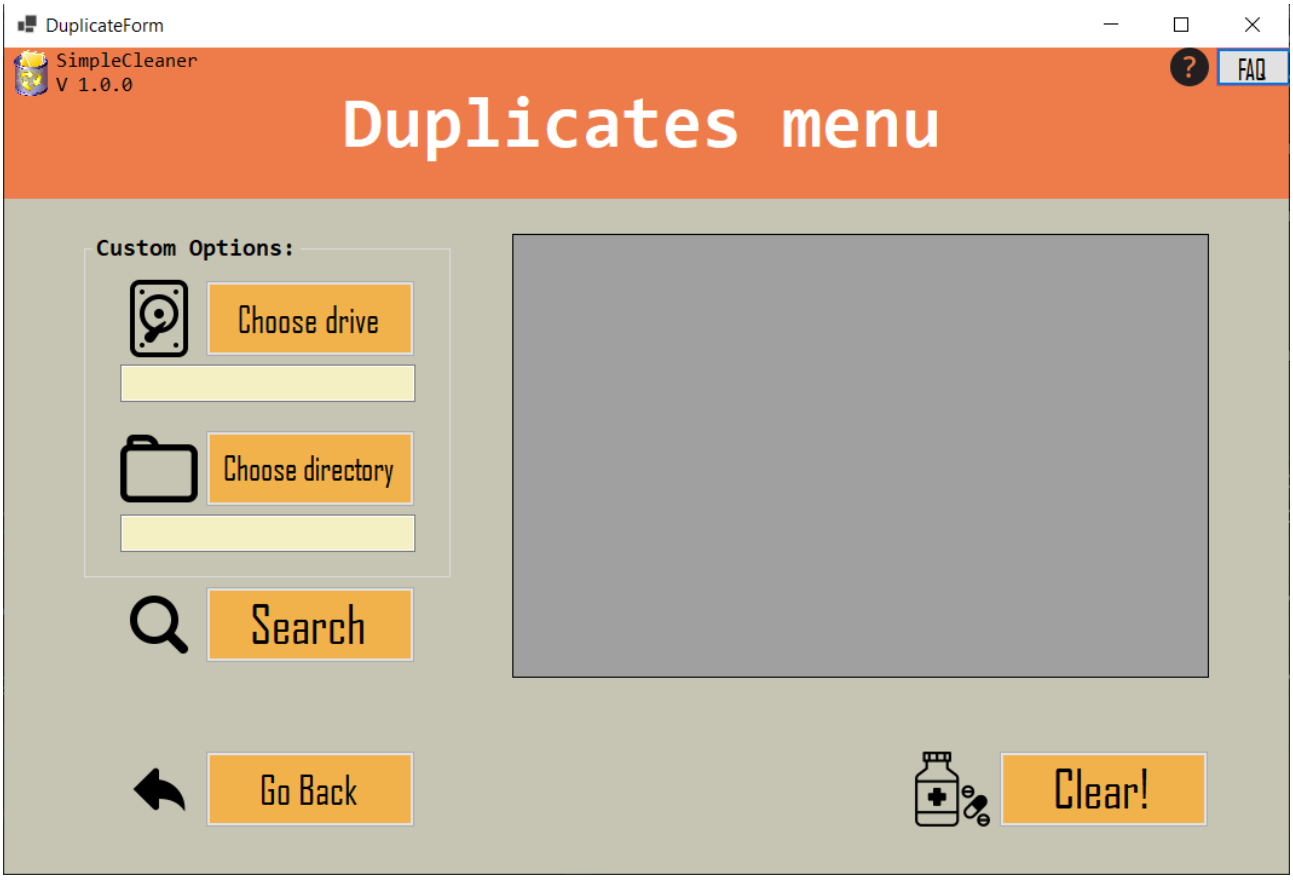

Рис. 3.13 - Вигляд другорядного меню програми, а саме форми DuplicateForm

Як бачимо, у цьому вікні користувач має можливість створити дублікати вибраних ним файлів. Також є клавіша повернення назад, тобто у головне меню програми. Форми програми в цілому схожі одна на одну, але притримуються однієї стилістики, виконуючи різний функціонал. Ось короткий опис функціоналу кожної клавіші:

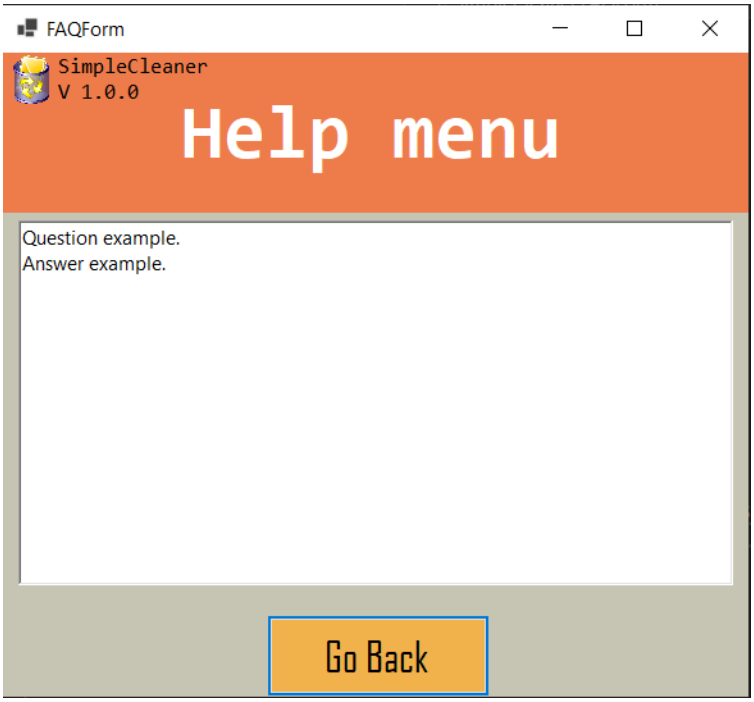

Рис. 3.14 - Вигляд другорядного меню програми, а саме форми FAQForm

У цьому вікні відображається інформація про програму, та відповіді на найчастіші питання. Було прийнято рішення не ускладнювати дане завдання, і реалізувати вивід даної інформації у простому RichTextBox.

Розміщення елементів інтерфейсу
Основний інтерфейс програми складається з двох ключових розділів:

1. Список з кнопками для переходу між завданнями програми
2. Основне вікно яке відображає вміст вкладок

Такий підхід до реалізації дизайну забезпечує користувачам максимально зручний та функціональний інтерфейс для моніторингу та аналізу стану системи, що сприяє ефективному управлінню ресурсами та підтримці стабільної роботи комп'ютера.

Використані Елементи інтерфейсу користувача (UI Elements)

3.2.1 Button (Кнопки)

- GoToDuplicatesButton
- FaqButton
- SearchButton
- BackupButton
- ClearButton
- RecycleBinButton
- TemporaryFilesButton
- BrowserCacheButton

Призначення: Кожна кнопка викликає певну дію, визначену в обробнику подій. Наприклад, SearchButton виконує пошук, BackupButton створює резервну копію, а ClearButton очищає дані.

3.2.2 ProgressBar (Прогрес-бар)

Призначення: Відображає стан виконання тривалих операцій, таких як пошук, створення резервної копії або очищення. Значення прогрес-бару змінюються відповідно до ходу виконання завдання, що надає користувачеві візуальний індикатор процесу.

3.2.3 DataGridView (Таблиця даних)

Призначення: Відображає результати різних операцій у вигляді таблиці. Наприклад, результати пошуку кешу або розмірів тимчасових файлів. Ця таблиця може мати кілька колонок, таких як "Cache type:" та "Cache Size (MB):", і автоматично налаштовує розмір колонок для зручного відображення даних.

3.2.4 FolderBrowserDialog (Діалог вибору папки)

Призначення: Використовується для надання користувачеві можливості вибрати папку для збереження резервної копії. Це стандартне діалогове вікно Windows, яке забезпечує зручний і уніфікований інтерфейс вибору папки.

3.2.5 MessageBox (Повідомлення)

Призначення: Використовується для відображення повідомлень користувачеві. Наприклад, повідомлення про вибір папки, успішне завершення резервного копіювання або попередження про відсутність даних для очищення. MessageBox також використовується для підтвердження дій, таких як створення резервної копії перед очищенням.

3.2.6 RichTextBox

Призначення: відображення найчастіших питань, та відповідей на них

3.3 Високорівневий опис реалізації програмного рішення згідно розроблених вимог

3.3.1 Архітектура програми:

- Мови програмування: Програма реалізована з використанням C# у середовищі .NET Framework для забезпечення інтеграції з операційною системою Windows та ефективної роботи з системними ресурсами.
- Модульна структура: Програма складається з декількох модулів, кожен з яких відповідає за окремий аспект очищення системи, включаючи модулі для пошуку сміттєвих файлів, видалення кешу браузерів, управління тимчасовими файлами, очищення корзини та управління файлами-дублікатами.

3.3.2 Очищення системи:

- Пошук сміттєвих файлів: Програма використовує спеціальні алгоритми для сканування системи та виявлення файлів, які не потрібні для нормального функціонування системи, включаючи залишки від видалених програм, тимчасові файли та кеш файли.
- Видалення кешу браузерів: Використовується API браузерів для доступу до їхніх кеш-файлів та їх безпечного видалення.
- Управління тимчасовими файлами: Програма сканує системні директорії для виявлення та видалення тимчасових файлів, що накопичуються з часом.
- Очищення корзини: Використання системних викликів для безпечного очищення корзини, видаляючи всі файли, що були переміщені до неї.

3.3.3 Інтерфейс користувача:

- Графічний інтерфейс: Інтерфейс програми розроблений з використанням Windows Forms, що забезпечує зручний та інтуїтивно зрозумілий користувацький досвід.
- Налаштування інтерфейсу: Користувач може налаштовувати інтерфейс відповідно до своїх потреб, включаючи вибір директорій для сканування та управління різними типами очищення.

3.3.4 Системні дані та журнали:

- Логування: Програма веде журнали дій, включаючи інформацію про виконані очищення, створення резервних копій та інші важливі події, що допомагає виявляти та вирішувати потенційні проблеми.
- Використання системних API Windows: Програма інтегрується з системними API для отримання доступу до системних ресурсів та даних, забезпечуючи ефективне та безпечне очищення.

3.3.5 Підтримка та поширення:

- Підтримка операційної системи Windows: Програма підтримує всі сучасні версії Windows, починаючи з Windows 7 і до останніх версій, для обох архітектур (32- та 64-бітних).
- Оновлення та підтримка: Програма передбачає можливість регулярних оновлень для додавання нових функцій та покращення існуючих, а також для забезпечення сумісності з новими версіями операційної системи.

3.4 Архітектура класів та їх призначення.

3.4.1 Клас CleaningForm : Form {}

Призначення: Клас, відповідає за головне меню програми, а саме за форму CleaningForm і працює з очищенням тимчасових файлів.

Функції:

- IsBrowserInstalled(string browserName): Перевіряє, чи встановлено вказаний браузер на комп'ютері, шляхом перевірки наявності папки відповідного браузера в системних каталогах. Це використовується для визначення, які браузери встановлені, і де знаходяться їхні кешові файли.
- DisplayCacheInfo(): Відображає інформацію про кеш для всіх встановлених браузерів шляхом виклику методів DisplayCacheInfoForChrome(), DisplayCacheInfoForFirefox() і так далі.
- Методи типу DisplayCacheInfoFor[BROWSER]: Відображають інформацію про кеш для окремого браузера, такого як Chrome, Firefox і

т. д. Кожен з цих методів визначає шлях до кешу конкретного браузера і виводить розмір кешу у мегабайтах.

- DisplayRecycleBinSizeInfo(): Відображає розмір кошика для всіх фіксованих і знімних дисків.
- DisplayTempFolderSizeInfo(string tempFolderPath): Відображає розмір тимчасових файлів у вказаному каталозі.
- ClearBrowserCache(): Очищає кеш для всіх встановлених браузерів, викликаючи відповідні методи очищення для кожного з них.
- Методи очищення кешу окремих браузерів, такі як ClearChromeCache(), ClearFirefoxCache(), і т. д.: Очищають кеш для конкретного браузера, видаляючи всі файли та папки відповідного каталогу.
- EmptyRecycleBin(): Очищає вміст кошика.
- ClearTempFolder(): Очищає тимчасові файли.
- BackupBrowserCache(string backupFolderPath, BrowserType browserType): Робить резервну копію кешу вказаного браузера, копіюючи вміст каталогу кешу в інший каталог.
- Методи створення резервних копій тимчасових файлів і кошика, такі як BackupTempFiles() і BackupRecycleBin(): Створюють резервні копії вмісту тимчасових файлів і кошика.
- Методи створення текстових файлів з інформацією про файли, такі як BrowserCacheInfo(), TempFilesInfo() і RecycleBinInfo(): Створюють текстові файли, які містять інформацію про файли у кеші браузерів, тимчасові файли та кошик.

3.4.2 Клас DuplicateForm : Form {}

Призначення: Клас, відповідає за другорядне меню програми, а саме за форму DuplicatesForm і працює з дублікатами файлів.

Функції:

- SearchForDuplicates(string directory): Пошук дубльованих файлів у вказаному каталозі та його підкаталогах.

- GetFileHash(Stream stream): Обчислення хеша файлу з використанням алгоритму MD5.
- DisplayDuplicates(List<string> duplicateFilePaths): Відображення списку дубльованих файлів у DataGridView.
- GetFileCountByCategory(string diskPath): Підрахунок кількості та розміру файлів за категоріями у вказаному каталозі.
- GetCategoryByExtension(string extension): Визначення категорії файлу на основі його розширення.
- DisplayResultsInDataGridView(Dictionary<string, (int count, double totalSize)> results): Відображення результатів підрахунку файлів за категоріями у DataGridView.
- CalculateTotalFiles(): Обчислення загальної кількості файлів за всіма категоріями.
- ClearDuplicateFiles(List<string> duplicateFilePaths): Видалення дубльованих файлів.
- CheckForAccessibility(DataGridView dataGridView): Перевірка доступності DataGridView для очищення.
- BackupDuplicateFiles(List<string> duplicateFilePaths): Створення резервних копій дубльованих файлів.
- FormClosingFunction(object sender, FormClosingEventArgs e): Обробка закриття форми.

3.4.3 Клас FAQForm : Form {}

Призначення: відображення відповідей на найпоширеніші питання.

Функції:

- FAQForm(): Конструктор, який ініціалізує компоненти форми та записує лог про успішне завантаження форми.

3.4.4 Діаграма класів:

Рис. 3.15 – Діаграма класів програми SimpleCleaner

3.5 Реалізація фіч програми

3.5.1 Видалення кешу браузера

Опис: Функція видалення кешу браузера видаляє тимчасові файли, створені веб-браузерами для прискорення доступу до веб-сторінок.

Реалізація:

- Використовується бібліотека для взаємодії з файловою системою, наприклад, System.IO.
- Кеш браузерів зберігається в конкретних директоріях, які визначаються на основі використовуваних браузерів (Chrome, Firefox, Edge тощо).
- Функція ClearBrowserCache видаляє файли з цих директорій, викликаючи метод Delete для кожного файлу.

3.5.2 Видалення тимчасових файлів

Опис: Видалення тимчасових файлів звільняє місце на диску, видаляючи непотрібні файли, створені операційною системою та додатками.

Реалізація:

- Використовується клас Directory для отримання списку файлів у тимчасових директоріях, таких як %TEMP%.
- Метод ClearTempFilesFolder видаляє всі файли у зазначених директоріях за допомогою циклу, що викликає File.Delete.

3.5.3 Очищення кошика

Опис: Очищення кошика видаляє всі файли, що знаходяться в кошику, остаточно звільняючи місце на диску.

Реалізація:

- Використовується Windows API, зокрема функції з бібліотеки shell32.dll, щоб взаємодіяти з кошиком.
- Метод EmptyRecycleBin викликає функцію SHEmptyRecycleBin, яка очищує кошик.

3.5.4 Створення бекапів

Опис: Функція створення бекапів копіює важливі файли перед їх видаленням, забезпечуючи можливість відновлення.

Реалізація:

- Використовується клас File для копіювання файлів із оригінальних директорій до резервної директорії.
- Метод BackupFiles виконує копіювання файлів, забезпечуючи, що всі важливі файли будуть збережені перед видаленням.

3.5.5 Пошук та видалення дублікатів файлів

Опис: Ця функція знаходить файли-дублікати у вибраній користувачем директорії та видаляє їх, звільняючи місце на диску.

Реалізація:

- Користувач вибирає директорію через діалогове вікно, реалізоване за допомогою FolderBrowserDialog.
- Використовується клас Directory для отримання списку всіх файлів у вибраній директорії.
- Метод SearchForDuplicates перевіряє хеші файлів для виявлення дублікатів, використовуючи алгоритм хешування (наприклад, MD5 або SHA-256).

- Дублікати відображаються у списку, і користувач може вибрати файли для видалення.
- Метод DeleteDuplicates видаляє вибрані файли.

3.5.6 Обрахунок кількості файлів різних типів

Опис: Функція обрахунку кількості файлів різних типів надає статистику щодо файлів у вибраній директорії.

Реалізація:

- Користувач вибирає директорію через діалогове вікно.
- Метод CountFileTypes використовує клас Directory для отримання списку файлів у директорії.
- Файли групуються за розширеннями, і підраховується кількість кожного типу файлів.
- Результати відображаються у вигляді звіту або діаграми.

3.5.7 Графічний інтерфейс користувача

Опис: Інтерфейс програми надає користувачу можливість взаємодії з функціями програми через кнопки та діалогові вікна.

Реалізація:

- Використовується Windows Forms для створення форм та елементів управління.
- Кожна функція прив'язана до відповідної кнопки на формі.
- Використовуються GroupBox для організації кнопок та RichTextBox для виведення інформації.

3.6 Експорт

Програма "SimpleCleaner" була експортована у виконавчий файл формату .exe за допомогою вбудованих інструментів Visual Studio. Це дозволяє запускати програму без необхідності попереднього встановлення середовища розробки .NET Framework та всіх необхідних бібліотек.

Переваги використання .exe файлу:

- Простота розповсюдження: Кінцевий користувач отримує готовий до запуску файл, що спрощує процес інсталяції та використання програми.
- Інтеграція всіх залежностей: Всі бібліотеки та модулі, необхідні для роботи програми, включені всередині .exe файлу. Це забезпечує стабільну роботу програми на будь-якій системі Windows без додаткових налаштувань.

Доступ до захищених ресурсів:

Для отримання доступу до системних файлів та інших захищених ресурсів, режим запуску програми за замовчуванням можна встановити як «Запуск від імені адміністратора». Це дозволяє програмі отримувати всі необхідні привілеї для коректної роботи з системними API та іншими критичними ресурсами.

4. Тестування

Тестування є критично важливим етапом розробки програмного забезпечення, який допомагає забезпечити якість і надійність додатка. Цей розділ описує процес тестування програми SimpleCleaner, включаючи методи тестування, використовувані інструменти та результати тестування. Тестування функціональності включає перевірку основних функцій програми, таких як очищення сміттєвих файлів, робота з дублікатами та обчислення кількості файлів різних типів.

Для тестування було підготовлено середовище з наступними компонентами:

- Операційна система: Windows 10
- .NET Framework 4.8
- Visual Studio 2019

4.1 Тестування кнопок

4.1.1 CleaningForm

- **Search All**: Пошук абсолютно всіх сміттєвих файлів

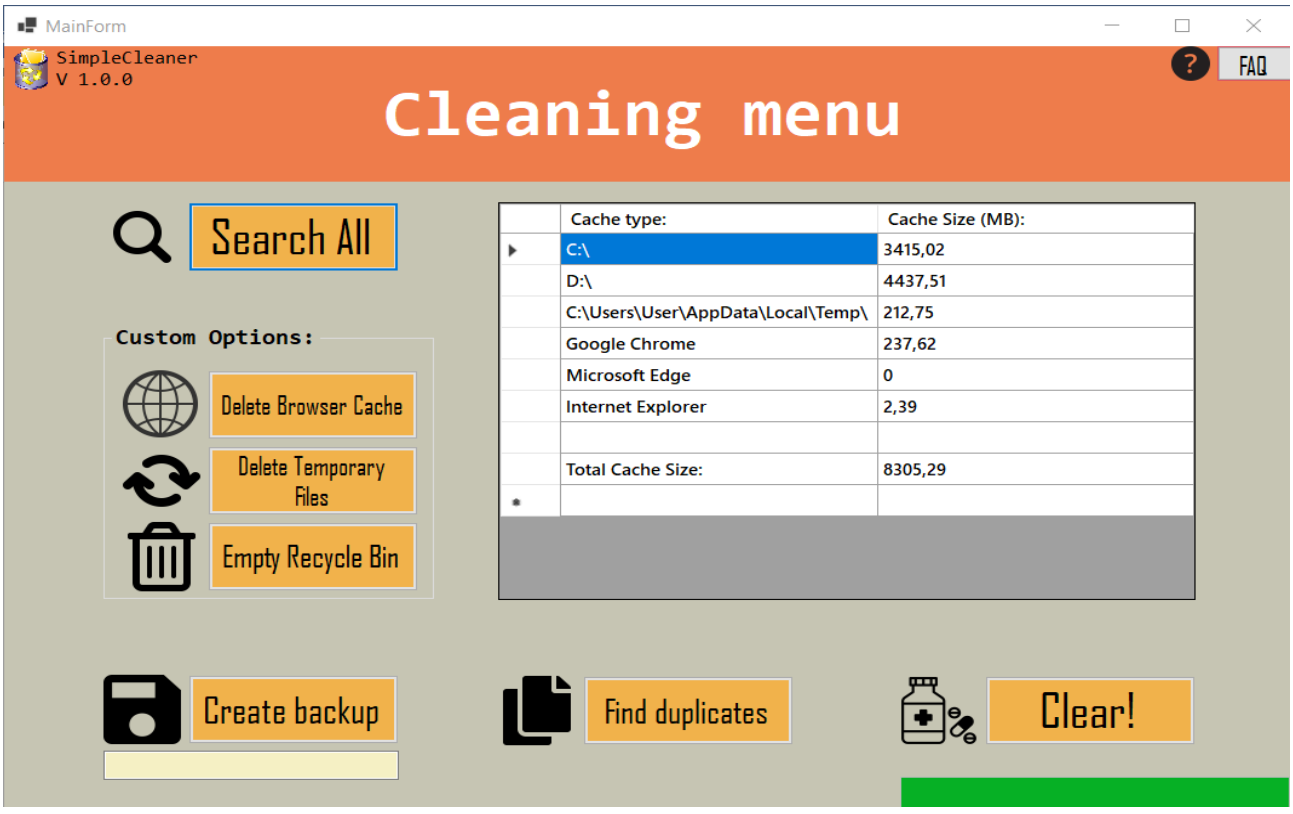

Рис. 4.1 Результат програми після натискання «Search All»

- **Delete Browser Cache**: Пошук файлів кешу браузера

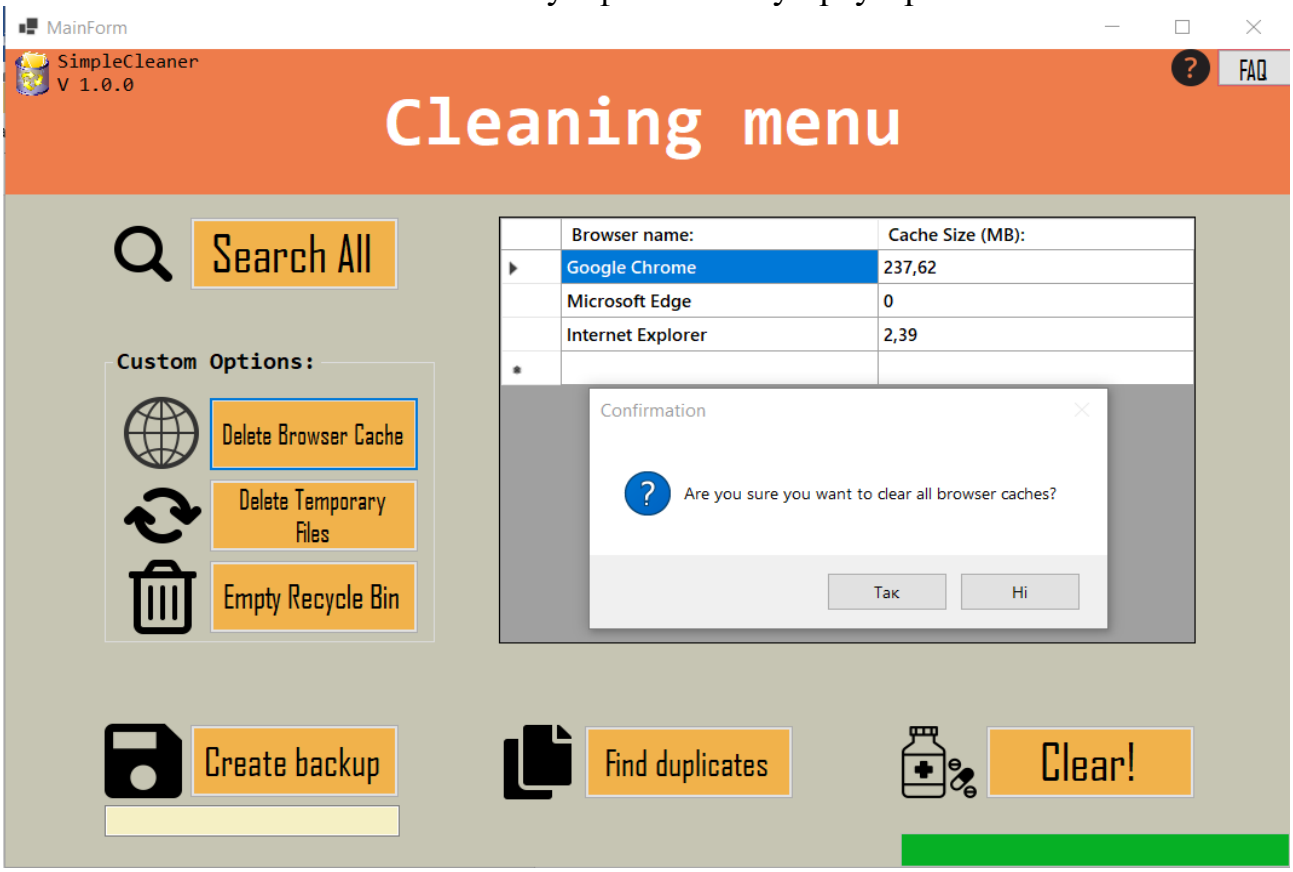

Рис. 4.2 Результат програми після натискання «Delete Browser Cache»

- **Delete Temporary Files**: Пошук тичасових файлів системи

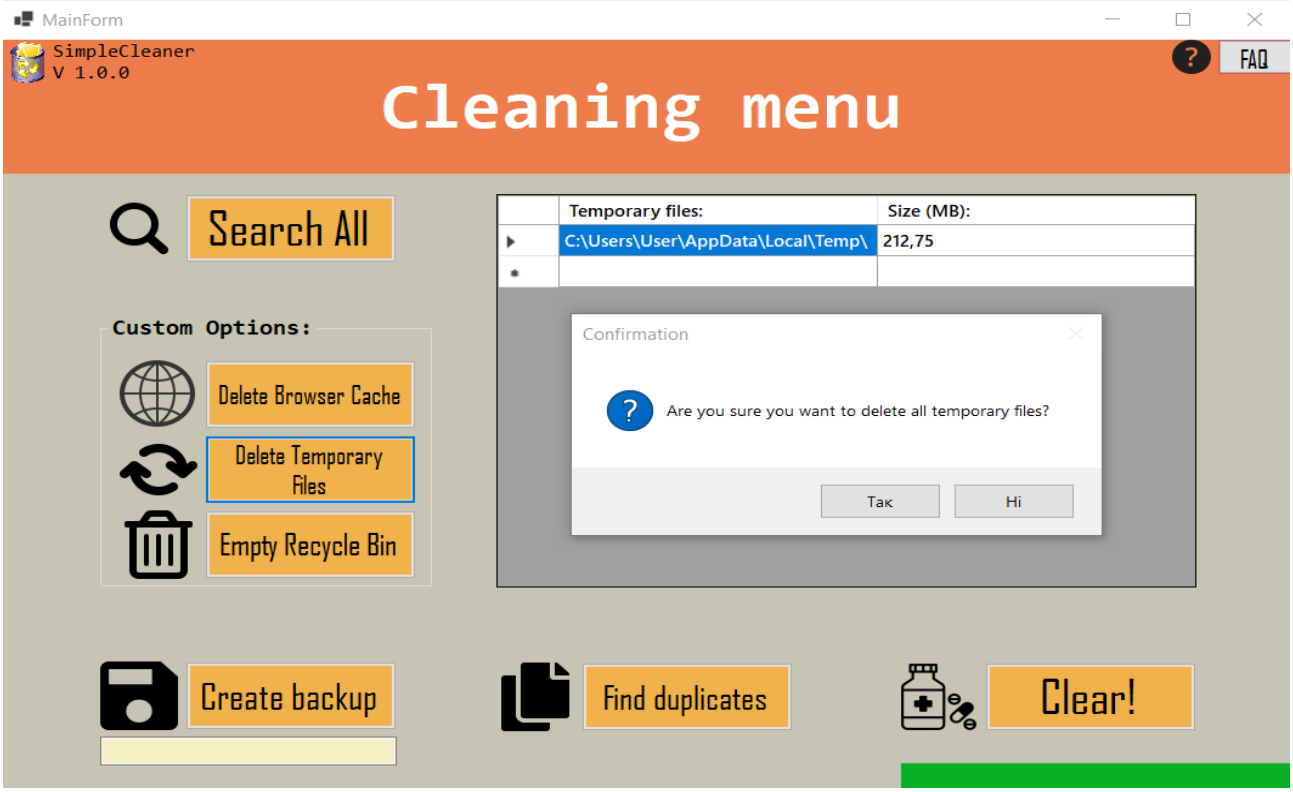

Рис. 4.3 Результат програми після натискання «Delete Temporary Files»

- **Empty Recycle Bin**: Очищення корзини

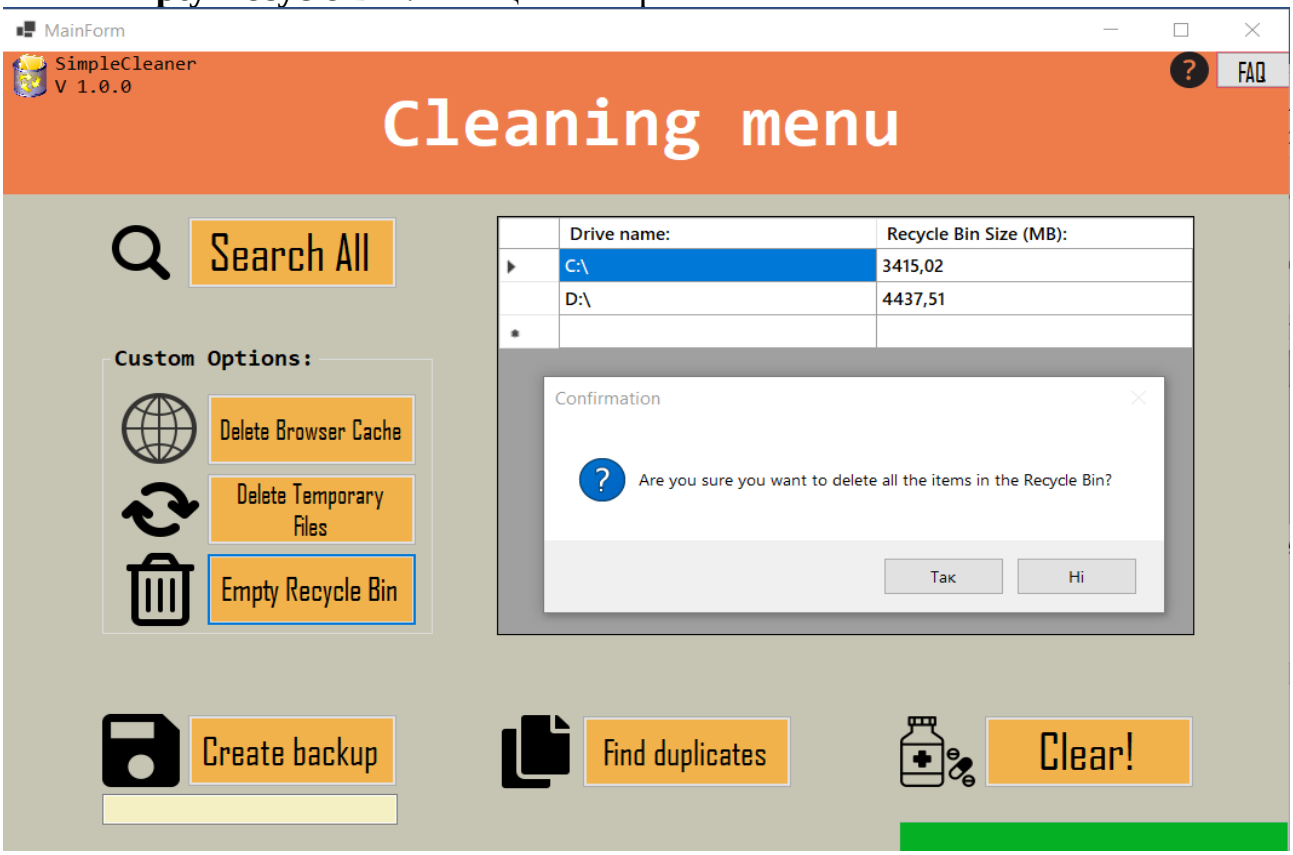

Рис. 4.4 Результат програми після натискання «Empty Recycle Bin»

- **Create backup**: Створення бекапу файлів перед їх видаленням

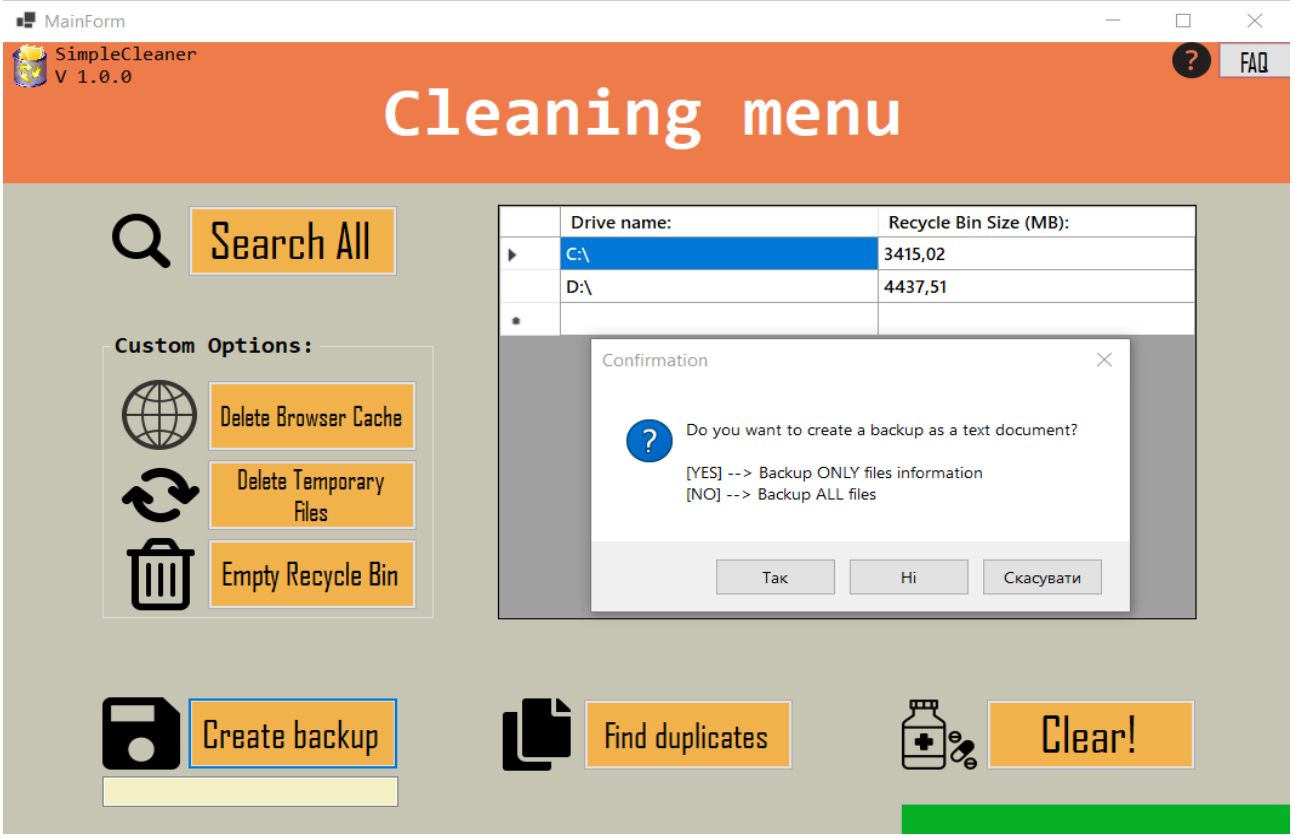

Рис. 4.5 Результат програми після натискання «Create backup»

- **Clear!:** Очищення обраних файлів

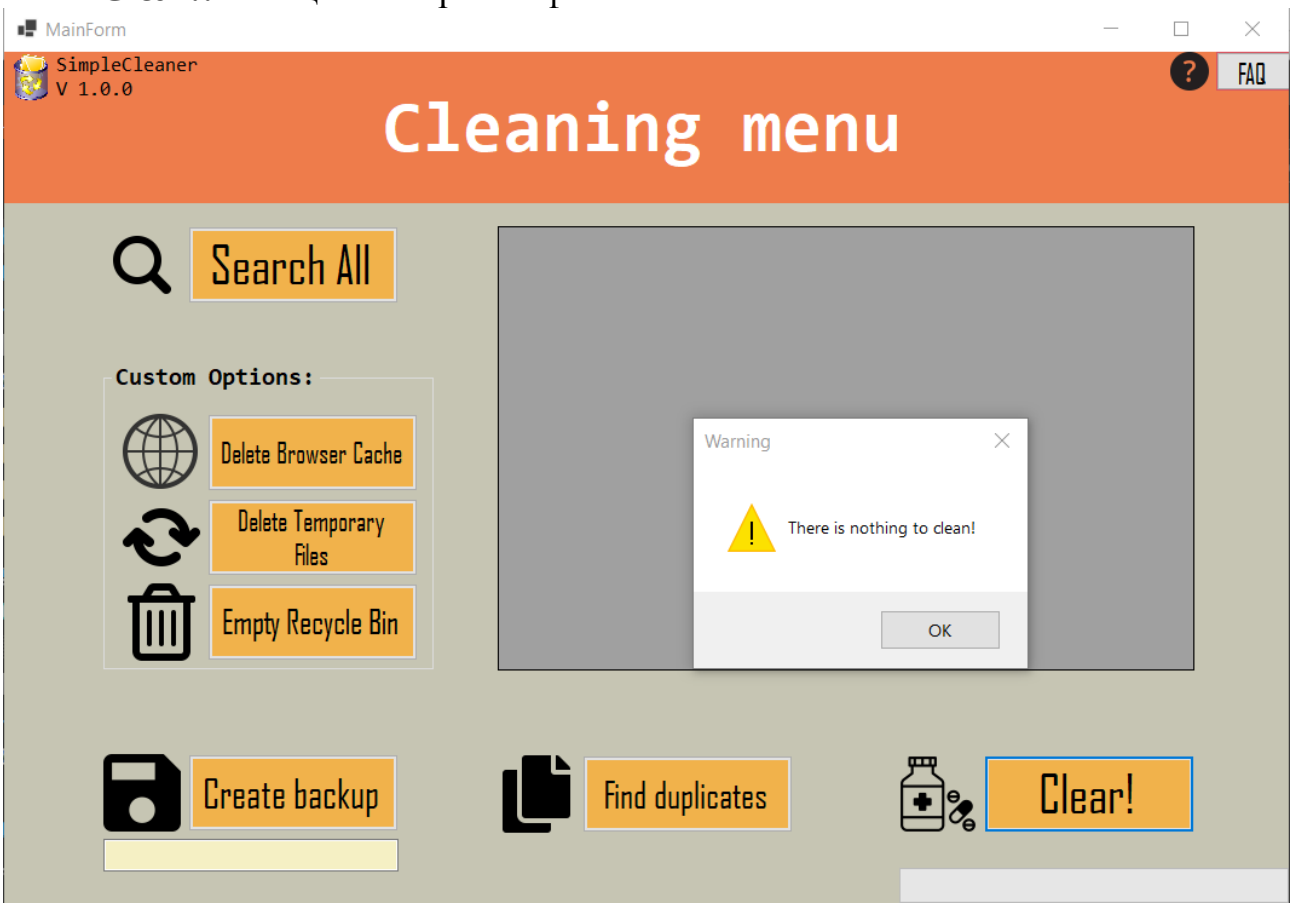

Рис. 4.6 Результат програми після натискання «Clear»

4.1.2 DuplicatesForm

- **Choose directory** обрання директорії для пошуку дублікатів

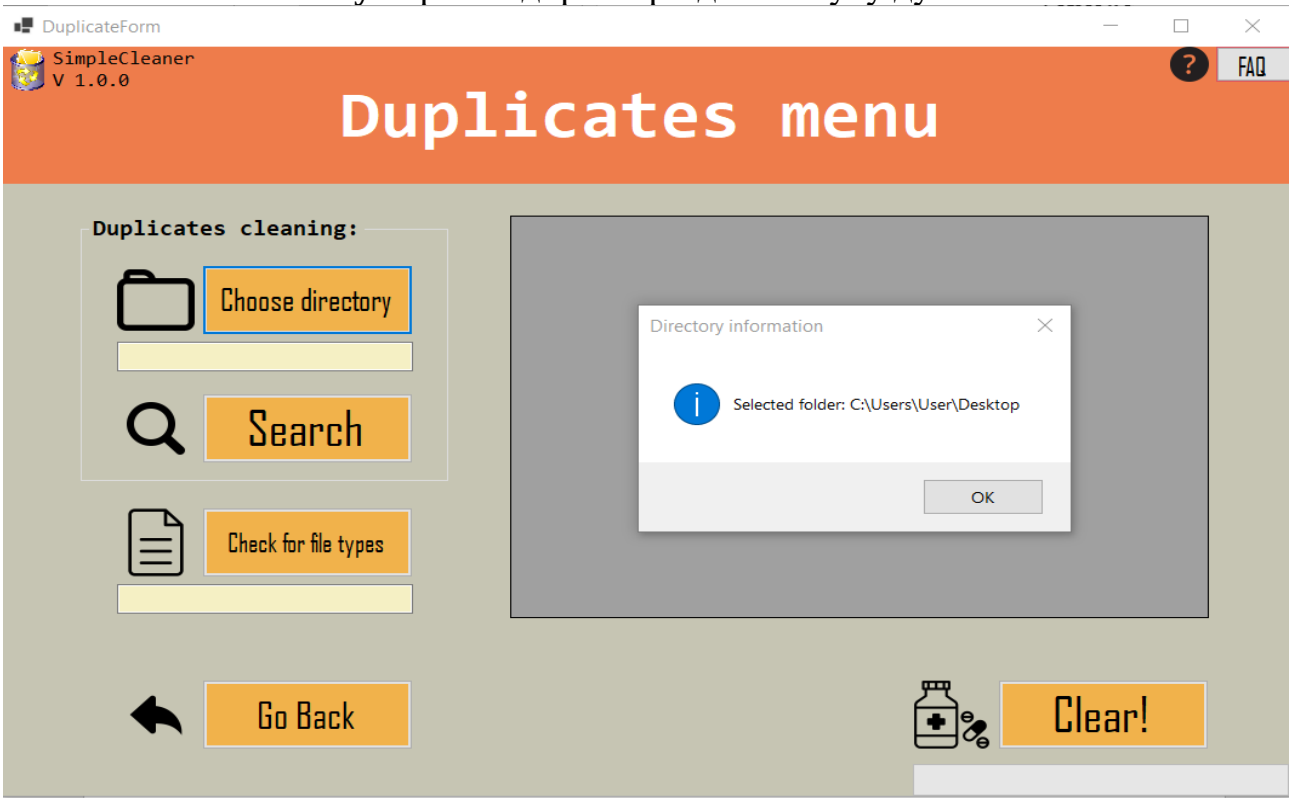

Рис. 4.7 Результат програми після натискання «Choose directory»

- **Search**: пошук дублікатів у вибраній папці

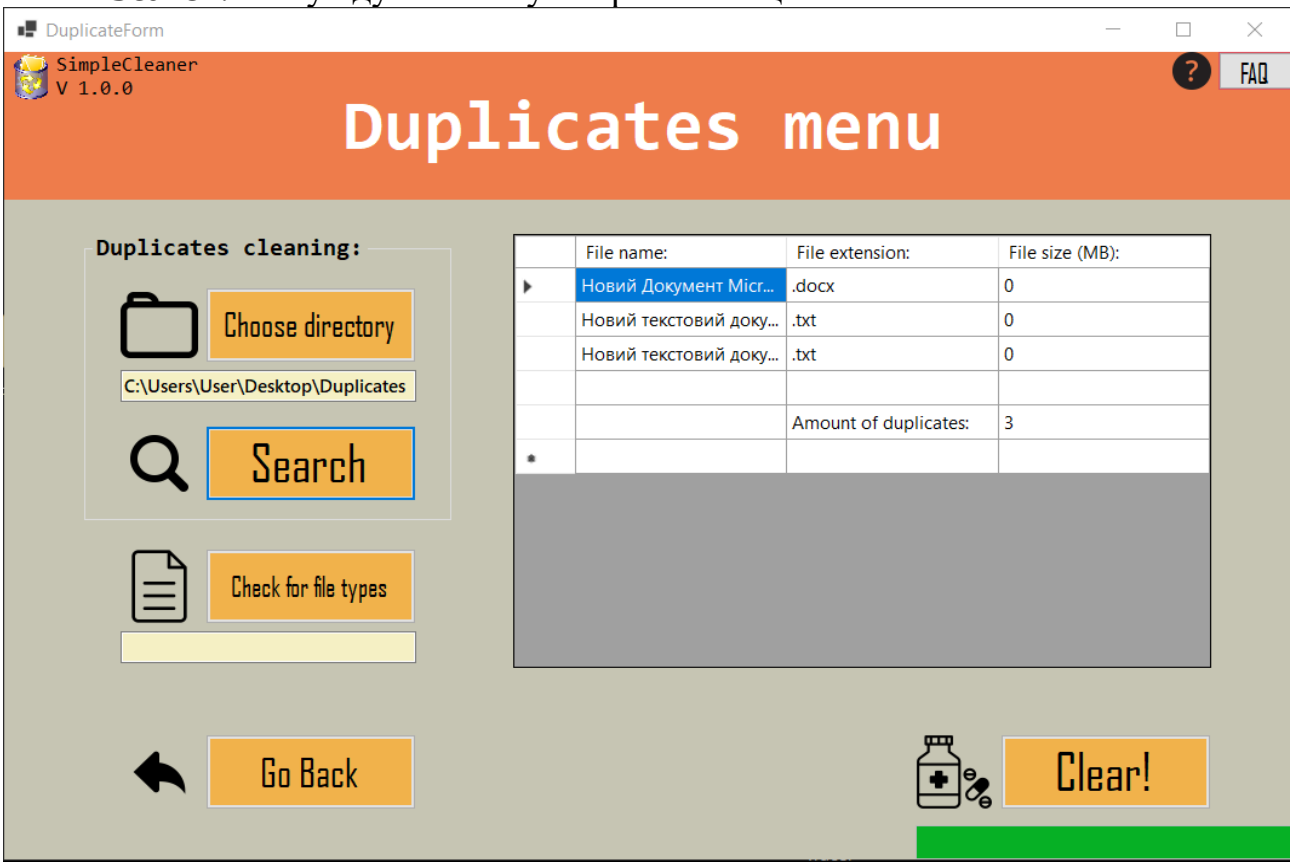

Рис. 4.8 Результат програми після натискання «Search»

- **Check for file types:** обрахунок кількості файлів різних типів

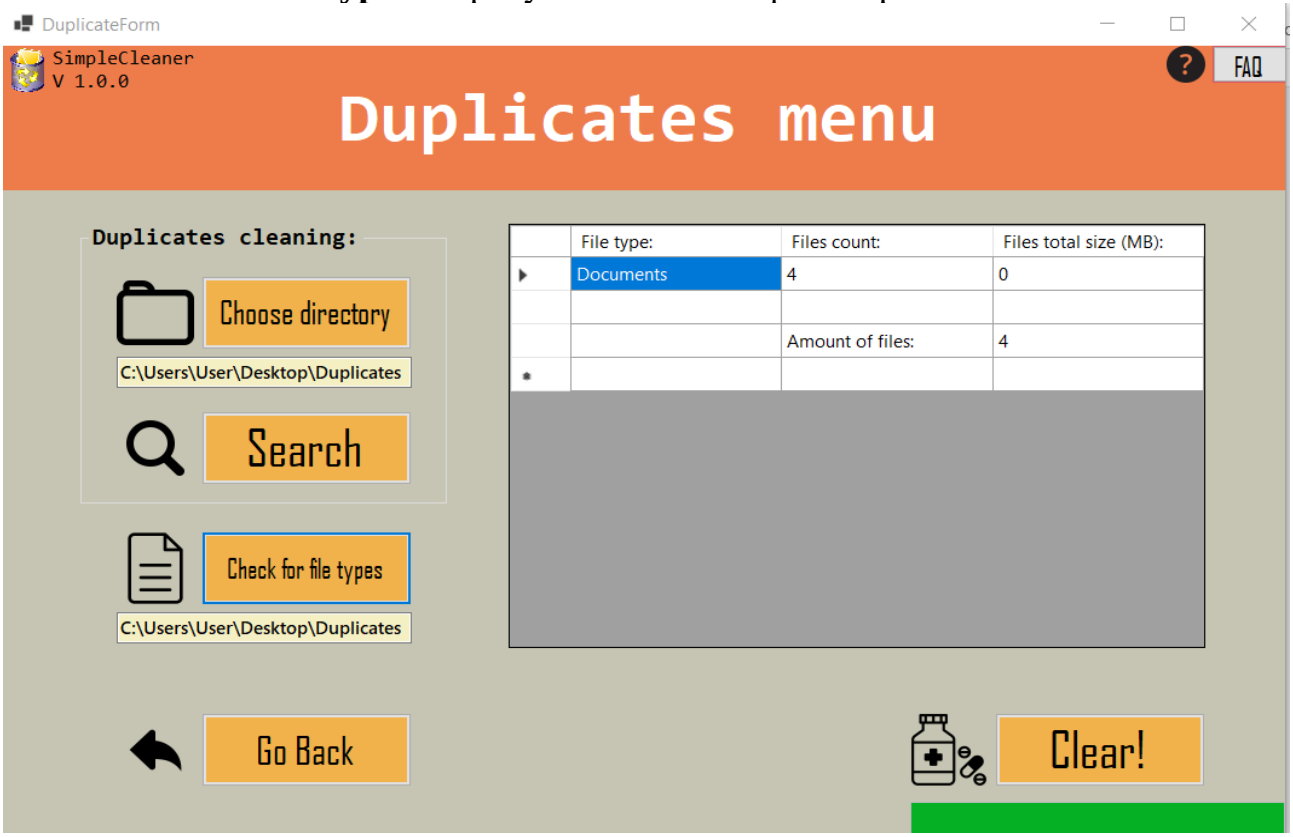

Рис. 4.9 Результат програми після натискання «Check for file types»

- **Clear!** Очищення обраних дублікатів

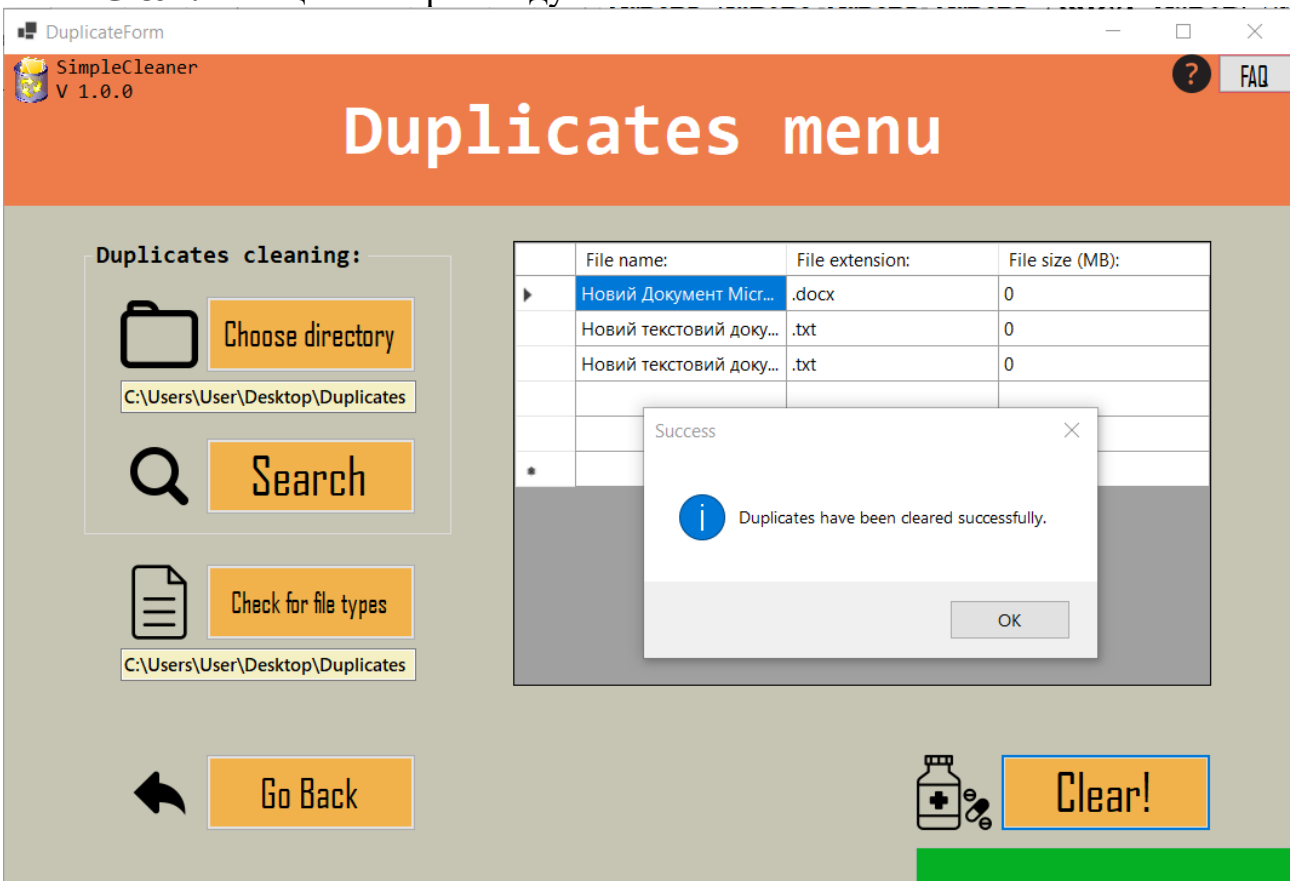

Рис. 4.10 Результат програми після натискання «Clear»

Як бачимо, кожна функція програми огорнута в try-catch, для обробки помилок. Помилки перехоплюються програмою і не відбувається вимкнення програми. Наприклад при натисканні кнопки «Clear!» для файлів різних типів:

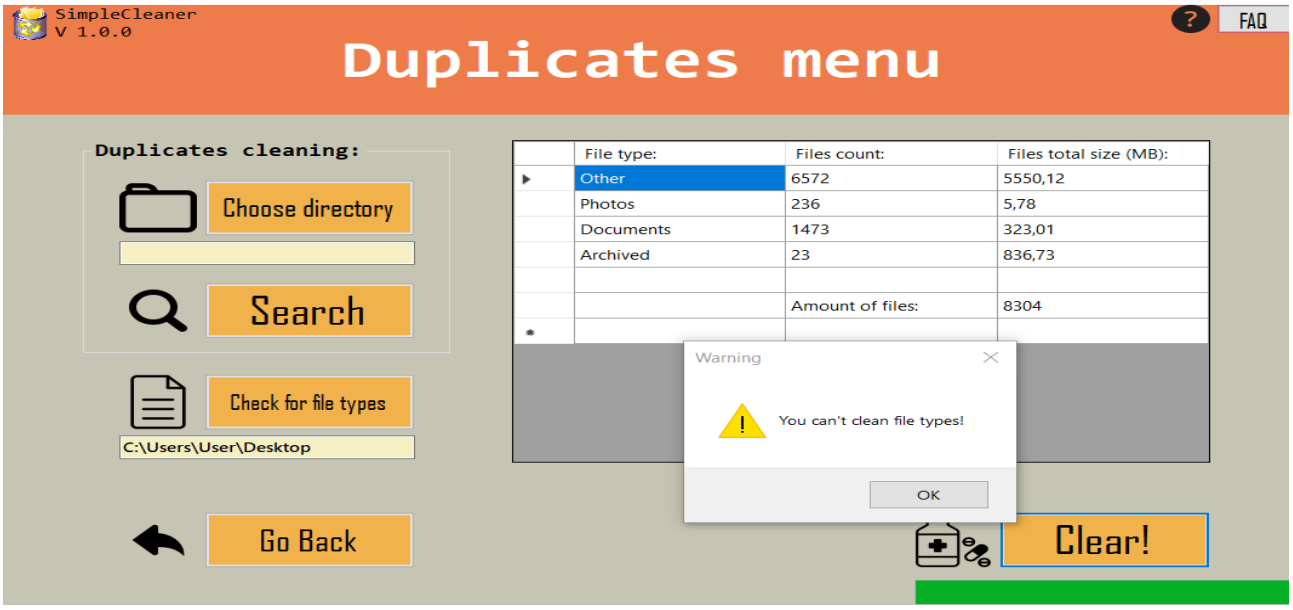

Рис. 4.11 Результат перехоплення помилок і поява message box

4.2 Логування подій програми

Логування є критично важливим для забезпечення стабільної, безпечної та продуктивної роботи програми. Воно допомагає розробникам та адміністраторам системи ефективно виявляти та усувати проблеми, аналізувати продуктивність, забезпечувати безпеку та надавати підтримку користувачам.

У даному випадку, кожна форма створює свій лог-файл, та записує інформацію про всі функції, які були задіяні в ній, під час її виконання.

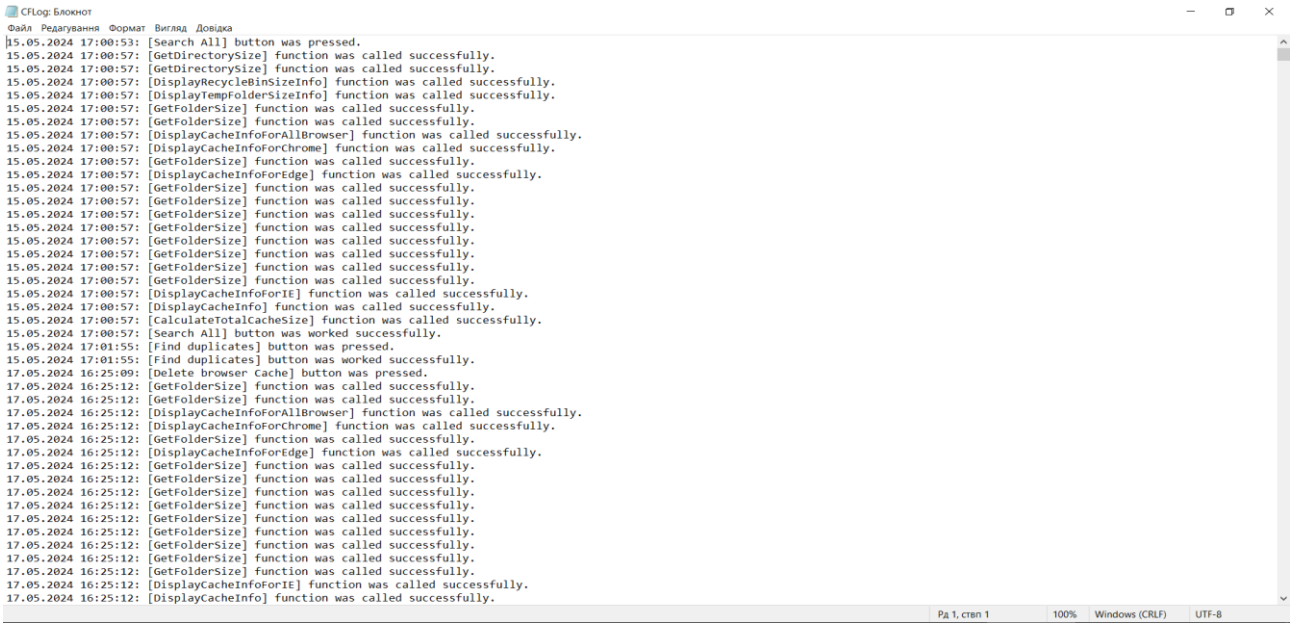

Рис. 4.12 Лог-файл форми CleaningForm

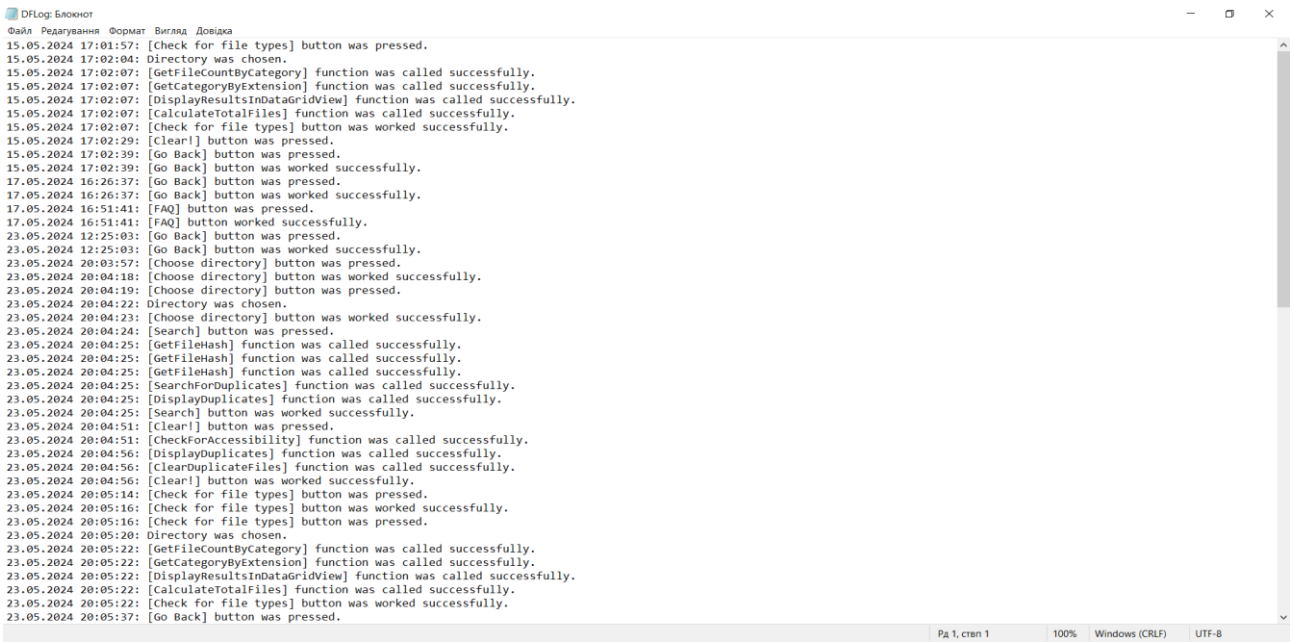

Рис. 4.13 Лог-файл форми DuplicatesForm

5. Висновки

Курсовий проект на C#, SimpleCleaner, є Windows Forms додатком для очищення системи від сміттєвих файлів, тимчасових файлів, кешу браузерів та управління файлами-дублікатами. Основні результати та функціональність програми:

- Компоненти інтерфейсу: кнопки для різних функцій очищення, відображення інформації про знайдені файли, та можливість перегляду і видалення дублікатів.
- Очищення системи: пошук та видалення сміттєвих і тимчасових файлів, очищення кешу браузерів та корзини, управління дублікованими файлами.
- Моніторинг та аналіз: аналіз та підрахунок файлів за категоріями, відображення результатів користувачеві.
- Інтерфейс користувача: зручний та інтуїтивний дизайн, налаштування відповідно до потреб користувача.
- Обробка помилок: надійна обробка винятків, забезпечення стабільності роботи.
- Структура коду: асинхронні операції, що забезпечують плавну роботу інтерфейсу.
- Приклад використання: функції для пошуку та видалення файлів, створення резервних копій, очищення системи.

Програма SimpleCleaner повністю відповідає вимогам, забезпечуючи ефективне очищення системи та управління файлами у зручному інтерфейсі.

6. Список використаних джерел

1. Glarysoft. (n.d.). Glary Utilities. Веб-сайт. URL: https://www.glarysoft.com/ (Дата зверення: 14.05.2024)
2. BleachBit. (n.d.). Веб-сайт. URL: https://www.bleachbit.org/ (Дата звернення: 14.05.2024)
3. Piriform. (n.d.). CCleaner. Веб-сайт. URL: https://www.ccleaner.com/ (Дата звернення: 14.05.2024)
4. Microsoft. (n.d.). C# and .NET documentation. Веб-сайт. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (Дата звернення: 23.04.2024)
5. Microsoft. (n.d.). Windows Forms documentation. Веб-сайт. URL: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/?view=netdesktop-8.0 (Дата звернення: 23.04.2024)
6. Дударь, Г. (2018). Відео уроки C# .NET Windows Forms. YouTube. URL: https://www.youtube.com/watch?v=gp2rA0rgq_0&list=PL0lO_mIqDDFWOMqSKFaLypANf1W7-o87q&ab_channel=%D0%93%D0%BE%D1%88%D0%B0%D0%94%D1%83%D0%B4%D0%B0%D1%80%D1%8C (Дата звернення: 21.04.2024)
7. GitHub. (n.d.). Windows API Code Pack. Веб-сайт. URL: https://github.com/contre/Windows-API-Code-Pack-1.1 (Дата звернення: 02.05.2024)
8. Microsoft. (n.d.). Asynchronous programming in C#. Веб-сайт. URL: https://learn.microsoft.com/en-us/dotnet/csharp/asynchronous-programming/async-scenarios (Дата звернення: 03.05.2024)
9. Microsoft. (n.d.). System.IO documentation. Веб-сайт. URL: https://learn.microsoft.com/en-us/dotnet/api/system.io?view=net-8.0 (Дата звернення: 03.05.2024)
10. Microsoft. (n.d.). Exception handling in C#. Веб-сайт. URL: https://learn.microsoft.com/en-us/dotnet/standard/exceptions/ (Дата звернення: 03.05.2024)
11. Microsoft. (n.d.). Process.Start documentation. Веб-сайт. URL: https://learn.microsoft.com/enus/dotnet/api/system.diagnostics.process.start?view=net-8.0 (Дата звернення: 06.05.2024)
12. Microsoft. (n.d.). System.Net.NetworkInformation documentation. Веб-сайт. URL: https://learn.microsoft.com/enus/dotnet/api/system.net.networkinformation?view=net-8.0 (Дата звернення: 24.05.2024)
13. Microsoft. (n.d.). User interface development with Windows Forms. Веб-сайт. URL: https://learn.microsoft.com/enus/dotnet/desktop/winforms/controls/overview?view=netdesktop-7.0 (Дата звернення: 21.04.2024)
14. Microsoft. (n.d.). Windows system calls and API documentation. Веб-сайт. URL: https://learn.microsoft.com/en-us/windows/win32/api/ (Дата звернення: 06.05.2024)

15. Microsoft. (n.d.). FileInfo documentation for duplicate monitoring. Веб-сайт. URL: https://learn.microsoft.com/enus/dotnet/api/system.io.fileinfo?view=net-8.0 (Дата звернення: 07.05.2024)
16. Stack Overflow. (n.d.). File I/O in C#. Веб-сайт. URL: https://stackoverflow.com/questions/tagged/file-io+c%23 (Дата звернення: 12.05.2024)
17. CodeProject. (n.d.). Creating backup solutions in C#. Веб-сайт. URL: https://www.codeproject.com/Articles/5370966/Writing-a-Simple-Csharp-Files-Backup-Solution-for (Дата звернення: 14.05.2024)
18. GitHub. (n.d.). Browser cache cleaning tools. Веб-сайт. URL:: https://github.com/topics/browser-cache-cleaner (Дата звернення: 17.05.2024)
19. Microsoft. (n.d.). Advanced user settings and configuration management. Веб-сайт. URL: https://learn.microsoft.com/enus/dotnet/api/system.configuration.configurationmanager?view=net-8.0 (Дата звернення: 18.05.2024)
20. Microsoft. (n.d.). File access control in .NET. Веб-сайт. URL: https://learn.microsoft.com/en-us/dotnet/standard/io/file-access-control (Дата звернення: 06.05.2024)

Додатки

Додаток 1 – Оцінка виконання програми за допомогою User Story

User Story 1: Очищення браузерного кешу

Оцінка часу виконання: 4 дні
Я, як користувач, хочу мати можливість очищувати кеш усіх встановлених у моїй системі браузерів в один клік.

Користь для користувача:

- Забезпечення можливості аналізу та очищення кешу різноманітних браузерів для забезпечення оптимальної продуктивності та вільного місця на носії.

Критерії готовності:

- Функції очищення браузерного кешу реалізовані.
- Створений інтерфейс для відображення цієї інформації.
- Пройдено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

User Story 2: Очищення тимчасових файлів сисетми

Оцінка часу виконання: 4 дні
Я, як користувач, хочу мати можливість очищувати тимчасові файли операційної системи з метою економії власного часу та вільного простору на носії.

Користь для користувача:
- Надання можливості очищення тимчасових файлів системи “в один клік”.

Критерії готовності:
- Реалізовані функції очищення тимчасових файлів системи.
- Інтерфейс для відображення цих даних створений.
- Проведено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

User Story 3: Очищення кошика

Оцінка часу виконання: 2 дні
Я, як користувач, хочу бачити інформацію про розмір файлів у кошику та мати зручний інструмент для їх очищення з метою уникнення проблем із зберіганням даних.

Користь для користувача:

- Забезпечення можливості відображення та очищення кошика системи в один клік. Також ця процедура істотно збільшить об’єм вільного простору на носії.

Критерії готовності:

- Розроблені функції для очищення кошика.
- Інтерфейс для відображення цих даних реалізований.
- Проведено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

User Story 4: Створення резервної копії файлів перед їх видаленням

Оцінка часу виконання: 5 днів
Я, як користувач, хочу мати можливість створити резервну копію файлів, які буду видаляти.

Користь для користувача:

- Надання можливості створення бекапу файлів перед їх остаточним видаленням з метою запобігання видалення важливих файлів.

Критерії готовності:

- Написані функції для створення резервних копій.
- Інтерфейс для відображення цих даних створений.
- Проведено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

User Story 5: Пошук файлів-дублікатів

Оцінка часу виконання: 5 дні
Я, як користувач, хочу перевірити певні папки на наявність файлів-дублікатів з метою економії дискового простору.

Користь для користувача:
- Забезпечення можливості перевірки директорій на наявність файлів-дублікатів.

Критерії готовності:
- Реалізовані функції для отримання інформації про файли-дублікати.
- Інтерфейс для відображення цих даних розроблений.
- Проведено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

User Story 6: Аналіз дисків

Оцінка часу виконання: 4 дні
Я, як користувач, хочу бачити кількість файлів у певній категорії. Наприклад, мені цікаво, скільки у мене пісень у форматі .mp3 знаходиться на комп’ютері.

Користь для користувача:
- Надання можливості обрахунку кількості файлів певної категорії.

Критерії готовності:
- Функції для аналізу дисків реалізовані.
- Створений інтерфейс для відображення цих даних.
- Пройдено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

User Story 7: Очищення абсолютно всіх сміттєвих файлів

Оцінка часу виконання: 6 днів
Я, як користувач, хочу мати можливість очищати всі сміттєві файли у “один клік”.

Користь для користувача:

- Забезпечення можливості очищення всіх сміттєвих файлів шляхом об’єднання різного функціоналу.

Критерії готовності:

- Функції для очищення абсолютно всіх сміттєвих файлів реалізовані.
- Інтерфейс для відображення цих даних створений.
- Пройдено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

User Story 8: Відображення відповідей на найпоширеніші питання

Оцінка часу виконання: 2 дні
Я, як користувач, хочу бачити окреме вікно, в якому відображені найпоширеніші питання щодо користування програмою та відповіді на них.

Користь для користувача:

- Надання можливості отримання відповідей на найпоширеніші питання щодо користування програмою.

Критерії готовності:

- Функції для відображення відповідей на питання реалізовані.
- Створений інтерфейс для відображення цих даних.
- Проведено тестування та відлагодження коду.
- Оновлена документація з описом нового функціоналу.

Додаток 2 – Вихідний код програми

CleaningForm.cs

```
using System;
using System.IO;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Security;
using System.Security.Permissions;


namespace SimpleCleaner
{
    public partial class CleaningForm : Form
    {
        ///FOR RECYCLE BIN CLEANING:///
        enum RecycleFlags : uint
        {
            SHRB_NOCONFIRMATION = 0x00000001, // Don't ask confirmation
            SHRB_NOPROGRESSUI = 0x00000002, // Don't show any windows dialog
            SHRB_NOSOUND = 0x00000004 // Don't make sound, ninja mode enabled :v
        }
        [DllImport("Shell32.dll", CharSet = CharSet.Unicode)]
        static extern uint SHEmptyRecycleBin(IntPtr hwnd, string pszRootPath, RecycleFlags
dwFlags);
        ////////////////////////////////

        Logger logger = new Logger("CFLog.txt");

        public CleaningForm()
        {
            InitializeComponent();
            //logger.Log("[CleaningForm] was loaded successfully.");
        }

        //GOTO ANOTHER WINDOWS:
        private void GoToDuplicatesButton_Click(object sender, EventArgs e)
        {
            logger.Log("[Find duplicates] button was pressed.");
            this.Visible = false;
            DuplicateForm newForm = new DuplicateForm();
            newForm.Show();
            logger.Log("[Find duplicates] button was worked successfully.");
        }

        private bool FAQFormOpened = false;
        private void FaqButton_Click(object sender, EventArgs e)
        {
            logger.Log("[FAQ] button was pressed.");
            if (!FAQFormOpened)
            {
                FAQForm newForm = new FAQForm();
                newForm.FormClosed += (s, args) => FAQFormOpened = false; // Set FAQFormOpened to
false when the form is closed
                newForm.Show();
                FAQFormOpened = true;
                logger.Log("[FAQ] button worked successfully.");
            }
            else
            {
                logger.Log("[FAQ] button was already pressed once.");
```

```
            }
        }
        /// ///////////

        //DEFAULT OPTIONS:
        private async void SearchButton_Click(object sender, EventArgs e)
        {
            logger.Log("[Search All] button was pressed.");
            ProgressBar.Value = 0;
            await SimulateLoadingAsync(1500);

            ResultGridView.Rows.Clear();
            ResultGridView.ColumnCount = 2;
            ResultGridView.Columns[0].Name = "Cache type:";
            ResultGridView.Columns[1].Name = "Cache Size (MB):";
            ResultGridView.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;

            //ResultGridView.Rows.Add("->RECYCLE BIN:", "");
            DisplayRecycleBinSizeInfo();
            string tempFolderPath = Path.GetTempPath();
            //ResultGridView.Rows.Add("->TEMPORARY FILES:", "");
            DisplayTempFolderSizeInfo(tempFolderPath);
            //ResultGridView.Rows.Add("->BROWSER CACHE:", "");
            DisplayCacheInfo();

            double totalCacheSizeMB = CalculateTotalCacheSize();
            ResultGridView.Rows.Add("", "");
            ResultGridView.Rows.Add("Total Cache Size:", totalCacheSizeMB.ToString("0.##"));

            logger.Log("[Search All] button was worked successfully.");
        }
        private async void BackupButton_Click(object sender, EventArgs e)
        {
            logger.Log("[Create backup] button was pressed.");
            if (ResultGridView.RowCount == 0)
            {
                MessageBox.Show("There is nothing to backup!", "Warning", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
                return;
            }

            using (var folderBrowserDialog = new FolderBrowserDialog())
            {
                DialogResult result = folderBrowserDialog.ShowDialog();

                if (result == DialogResult.OK &&
!string.IsNullOrWhiteSpace(folderBrowserDialog.SelectedPath))
                {
                    string selectedFolderPath = folderBrowserDialog.SelectedPath;
                    MessageBox.Show($"Selected folder: {selectedFolderPath}", "Backup
information", MessageBoxButtons.OK, MessageBoxIcon.Information);
                    DialogResult backupTypeResult = MessageBox.Show("Do you want to create a
backup as a text document?\n\n[YES] --> Backup ONLY files information\n[NO] --> Backup ALL
files", "Confirmation", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question);
                    logger.Log("Backup folder was chosen.");
                    ProgressBar.Value = 0;
                    await SimulateLoadingAsync(1000);

                    try
                    {
                        switch (backupTypeResult)
                        {
                            case DialogResult.Yes:
                                if (IsBrowserInstalled("Chrome"))
                                    BrowserCacheInfo(selectedFolderPath, BrowserType.Chrome);
                                if (IsBrowserInstalled("Firefox"))
                                    BrowserCacheInfo(selectedFolderPath, BrowserType.Firefox);
                                //if (IsBrowserInstalled("Edge"))
```

```
                                //    BackupBrowserCacheInfo(selectedFolderPath,
BrowserType.Edge);
                                //if (IsBrowserInstalled("InternetExplorer"))
                                //    BackupBrowserCacheInfo(selectedFolderPath,
BrowserType.InternetExplorer);
                                if (IsBrowserInstalled("Opera"))
                                    BrowserCacheInfo(selectedFolderPath, BrowserType.Opera);
                                RecycleBinInfo(selectedFolderPath);
                                TempFilesInfo(selectedFolderPath);
                                MessageBox.Show($"Information backup completed successfully!",
"Backup Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
                                break;

                            case DialogResult.No:
                                if (IsBrowserInstalled("Chrome"))
                                    BackupBrowserCache(selectedFolderPath, BrowserType.Chrome);
                                if (IsBrowserInstalled("Firefox"))
                                    BackupBrowserCache(selectedFolderPath, BrowserType.Firefox);
                                //if (IsBrowserInstalled("Edge"))
                                //    BackupBrowserCache(selectedFolderPath, BrowserType.Edge);
                                //if (IsBrowserInstalled("InternetExplorer"))
                                //    BackupBrowserCache(selectedFolderPath,
BrowserType.InternetExplorer);
                                  if (IsBrowserInstalled("Opera"))
                                    BackupBrowserCache(selectedFolderPath, BrowserType.Opera);
                                BackupRecycleBin(selectedFolderPath);
                                BackupTempFiles(selectedFolderPath);
                                MessageBox.Show($" Total backup completed successfully!", "Backup
Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
                                break;

                            case DialogResult.Cancel:
                                this.Close();
                                break;
                            default:
                                break;
                        }
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show($"Backup failed: {ex.Message}", "Backup Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    }
                }
            }
            logger.Log("[Create backup] button was worked successfully.");
        }
        private async void ClearButton_Click(object sender, EventArgs e)
        {
            logger.Log("[Clear!] button was pressed.");
            if (ResultGridView.RowCount == 0)
            {
                MessageBox.Show("There is nothing to clean!", "Warning", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
                return;
            }

            DialogResult result = MessageBox.Show("Do you want to create a backup before
cleaning?", "Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
            if (result == DialogResult.Yes)
            {
                BackupButton_Click(null, EventArgs.Empty);
                ProgressBar.Value = 0;
                await SimulateLoadingAsync(2500);
                ClearBrowserCache();
                ClearTempFolder();
                EmptyRecycleBin();
            }
            if (result == DialogResult.No)
```

```
    {
        ProgressBar.Value = 0;
        await SimulateLoadingAsync(1100);
        ClearBrowserCache();
        ClearTempFolder();
        EmptyRecycleBin();
    }

    logger.Log("[Clear!] button was worked successfully.");
}
/// ///////////

//CUSTOM OPTIONS:
private async void RecycleBinButton_Click(object sender, EventArgs e)
{
    logger.Log("[Empty Recycle Bin] button was pressed.");
    ProgressBar.Value = 0;
    await SimulateLoadingAsync(1100);

    ResultGridView.Rows.Clear();
    ResultGridView.ColumnCount = 2;
    ResultGridView.Columns[0].Name = "Drive name:";
    ResultGridView.Columns[1].Name = "Recycle Bin Size (MB):";
    ResultGridView.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;

    DisplayRecycleBinSizeInfo();
    EmptyRecycleBin();

    logger.Log("[Empty Recycle Bin] button was worked successfully.");
}
private async void TemporaryFilesButton_Click(object sender, EventArgs e)
{
    logger.Log("[Delete Temporary Files] button was pressed.");
    ProgressBar.Value = 0;
    await SimulateLoadingAsync(1500);

    ResultGridView.Rows.Clear();
    ResultGridView.ColumnCount = 2;
    ResultGridView.Columns[0].Name = "Temporary files:";
    ResultGridView.Columns[1].Name = "Size (MB):";
    ResultGridView.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;

    string tempFolderPath = Path.GetTempPath();
    DisplayTempFolderSizeInfo(tempFolderPath);
    ClearTempFolder();

    logger.Log("[Delete Temporary Files] button was worked successfully.");
}
private async void BrowserCacheButton_Click(object sender, EventArgs e)
{
    logger.Log("[Delete browser Cache] button was pressed.");
    ProgressBar.Value = 0;
    await SimulateLoadingAsync(1500);

    ResultGridView.Rows.Clear();
    ResultGridView.ColumnCount = 2;
    ResultGridView.Columns[0].Name = "Browser name:";
    ResultGridView.Columns[1].Name = "Cache Size (MB):";
    ResultGridView.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;

    DisplayCacheInfo();
    ClearBrowserCache();

    logger.Log("[Delete browser Cache] button was worked successfully.");
}
/// ///////////

private async Task SimulateLoadingAsync(int durationMilliseconds)
{
```

```
            const int updateInterval = 10;
            int numUpdates = durationMilliseconds / updateInterval;
            int percentIncrement = 10 / numUpdates;

            for (int i = 0; i < numUpdates; i++)
            {
                int percentComplete = (i + 1) * percentIncrement;
                ProgressBar.PerformStep();

                await Task.Delay(updateInterval);
            }
        }
        private bool IsBrowserInstalled(string browserName)
        {
            switch (browserName.ToLower())
            {
                case "chrome":
                    string chromePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
                                                    @"Google\Chrome");
                    return Directory.Exists(chromePath);

                case "firefox":
                    string firefoxPath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
                                                     @"Mozilla\Firefox");
                    return Directory.Exists(firefoxPath);

                case "edge":
                    string edgePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
                                                   @"Microsoft\Edge");
                    return Directory.Exists(edgePath);

                case "internetexplorer":
                    string iePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ProgramFiles),
                                                  @"Internet Explorer");
                    return Directory.Exists(iePath);

                case "opera":
                    string operaPath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData),
                                                   @"Opera Software\Opera Stable");
                    return Directory.Exists(operaPath);

                default:
                    logger.Log("[IsBrowserInstalled] function was called successfully.");
                    return false;

            }
        }

        //INFO OUTPUT:
        //BROWSERS:
        private void DisplayCacheInfo()
        {
            //ResultGridView.Rows.Clear();

            if (IsBrowserInstalled("Chrome"))
                DisplayCacheInfoForChrome();

            if (IsBrowserInstalled("Firefox"))
                DisplayCacheInfoForFirefox();

            if (IsBrowserInstalled("Edge"))
                DisplayCacheInfoForEdge();

            if (IsBrowserInstalled("InternetExplorer"))
```

```
                DisplayCacheInfoForIE();

            if (IsBrowserInstalled("Opera"))
                DisplayCacheInfoForOpera();

            logger.Log("[DisplayCacheInfo] function was called successfully.");
        }
        private void DisplayCacheInfoForAllBrowser(string browserName, string cachePath)
        {
            long cacheSizeBytes = GetFolderSize(cachePath);
            double cacheSizeMB = (double)cacheSizeBytes / (1024 * 1024);
            ResultGridView.Rows.Add(browserName, cacheSizeMB.ToString("0.##"));
            logger.Log("[DisplayCacheInfoForAllBrowser] function was called successfully.");
        }
        private void DisplayCacheInfoForChrome()
        {
            //ResultGridView.Rows.Clear();

            DisplayCacheInfoForAllBrowser("Google Chrome",
                Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData)
                + @"\Google\Chrome\User Data\Default\Cache");
            logger.Log("[DisplayCacheInfoForChrome] function was called successfully.");
        }
        private void DisplayCacheInfoForEdge()
        {
            string edgeCachePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
                                                @"Microsoft\Edge\User Data\Default\Cache");
            long cacheSizeBytes = GetFolderSize(edgeCachePath);
            double cacheSizeMB = (double)cacheSizeBytes / (1024 * 1024);
            ResultGridView.Rows.Add("Microsoft Edge", cacheSizeMB.ToString("0.##"));
            logger.Log("[DisplayCacheInfoForEdge] function was called successfully.");
        }
        private void DisplayCacheInfoForIE()
        {
            string ieCachePath =
Environment.GetFolderPath(Environment.SpecialFolder.InternetCache);
            long cacheSizeBytes = GetFolderSize(ieCachePath);
            double cacheSizeMB = (double)cacheSizeBytes / (1024 * 1024);
            ResultGridView.Rows.Add("Internet Explorer", cacheSizeMB.ToString("0.##"));
            logger.Log("[DisplayCacheInfoForIE] function was called successfully.");
        }
        private void DisplayCacheInfoForFirefox()
        {
            string firefoxCachePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
                                                  @"Mozilla\Firefox\Profiles");
            string[] profiles = Directory.GetDirectories(firefoxCachePath, "*.default*");

            foreach (string profile in profiles)
            {
                string cachePath = Path.Combine(profile, "cache2");
                long cacheSizeBytes = GetFolderSize(cachePath);
                double cacheSizeMB = (double)cacheSizeBytes / (1024 * 1024);
                string profileName = Path.GetFileName(profile);
                ResultGridView.Rows.Add($"Firefox ({profileName})",
cacheSizeMB.ToString("0.##"));
            }
            logger.Log("[DisplayCacheInfoForFirefox] function was called successfully.");
        }
        private void DisplayCacheInfoForOpera()
        {
            string operaCachePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData),
                                                @"Opera Software\Opera Stable\Cache");
            long cacheSizeBytes = GetFolderSize(operaCachePath);
            double cacheSizeMB = (double)cacheSizeBytes / (1024 * 1024);
            ResultGridView.Rows.Add("Opera", cacheSizeMB.ToString("0.##"));
            logger.Log("[DisplayCacheInfoForOpera] function was called successfully.");
```

```
        }
        //RECYCLE BIN:
        private void DisplayRecycleBinSizeInfo()
        {
            //ResultGridView.Rows.Clear();

            try
            {

                DriveInfo[] drives = DriveInfo.GetDrives();

                foreach (DriveInfo drive in drives)
                {
                    if (drive.DriveType == DriveType.Fixed || drive.DriveType ==
DriveType.Removable)
                    {
                        string recycleBinPath = Path.Combine(drive.RootDirectory.FullName,
"$Recycle.Bin");
                        long recycleBinSizeBytes = GetDirectorySize(recycleBinPath);
                        double recycleBinSizeMB = (double)recycleBinSizeBytes / (1024 * 1024);
                        ResultGridView.Rows.Add(drive.Name, recycleBinSizeMB.ToString("0.##"));
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error retrieving Recycle Bin size information: " + ex.Message,
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            logger.Log("[DisplayRecycleBinSizeInfo] function was called successfully.");
        }
        //TEMPORARY FILES:
        private void DisplayTempFolderSizeInfo(string tempFolderPath)
        {
            //ResultGridView.Rows.Clear();
            try
            {
                string[] tempFiles = Directory.GetFiles(tempFolderPath);
                long totalSizeBytes = 0;

                foreach (string tempFile in tempFiles)
                {
                    FileInfo fileInfo = new FileInfo(tempFile);
                    totalSizeBytes += fileInfo.Length;
                }

                double totalSizeMB = (double)totalSizeBytes / (1024 * 1024);
                ResultGridView.Rows.Add(tempFolderPath, totalSizeMB.ToString("0.##"));
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error retrieving temporary files size information: " +
ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            logger.Log("[DisplayTempFolderSizeInfo] function was called successfully.");
        }
        private double CalculateTotalCacheSize()
        {
            double totalSizeMB = 0.0;

            foreach (DataGridViewRow row in ResultGridView.Rows)
            {
                if (row.Cells[0].Value != null && row.Cells[1].Value != null)
                {
                    string cacheType = row.Cells[0].Value.ToString();
                    string sizeString = row.Cells[1].Value.ToString();
```

```
                    if (double.TryParse(sizeString.Replace(" MB", ""), out double sizeMB))
                    {
                        totalSizeMB += sizeMB;
                    }
                }
            }

            logger.Log("[CalculateTotalCacheSize] function was called successfully.");
            return totalSizeMB;
        }
        /// ///////////

        //CLEANING:
        private void ClearBrowserCache()
        {
            if (MessageBox.Show("Are you sure you want to clear all browser caches?",
"Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
            {
                if (IsBrowserInstalled("Chrome"))
                {
                    ClearChromeCache();
                }

                if (IsBrowserInstalled("Firefox"))
                {
                    ClearFirefoxCache();
                }

                if (IsBrowserInstalled("Edge"))
                {
                    ClearEdgeCache();
                }

                if (IsBrowserInstalled("InternetExplorer"))
                {
                    ClearIECache();
                }

                if (IsBrowserInstalled("Opera"))
                {
                    ClearOperaCache();
                }
                logger.Log("[ClearBrowserCache] function was called successfully.");

                MessageBox.Show("Browser caches have been cleared successfully.", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                DisplayCacheInfo();
            }
        }
        private void ClearCacheFolder(string folderPath)
        {
            try
            {
                DirectoryInfo di = new DirectoryInfo(folderPath);
                foreach (FileInfo file in di.GetFiles())
                {
                    file.Delete();
                }
                foreach (DirectoryInfo dir in di.GetDirectories())
                {
                    dir.Delete(true);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Error clearing cache: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
```

```
            logger.Log("[ClearCacheFolder] function was called successfully.");
        }
        private void ClearChromeCache()
        {
            string chromeCachePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
                                                    @"Google\Chrome\User Data\Default\Cache");
            ClearCacheFolder(chromeCachePath);
            logger.Log("[ClearChromeCache] function was called successfully.");
        }
        private void ClearFirefoxCache()
        {
            string firefoxCachePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
                                                     @"Mozilla\Firefox\Profiles");
            string[] profiles = Directory.GetDirectories(firefoxCachePath, "*.default*");

            foreach (string profile in profiles)
            {
                string cachePath = Path.Combine(profile, "cache2");
                ClearCacheFolder(cachePath);
            }

            logger.Log("[ClearFirefoxCache] function was called successfully.");
        }
        private void ClearEdgeCache()
        {
            string edgeCachePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
                                                   @"Microsoft\Edge\User Data\Default\Cache");
            ClearCacheFolder(edgeCachePath);

            logger.Log("[ClearEdgeCache] function was called successfully.");
        }
        private void ClearIECache()
        {
            string ieCachePath =
Environment.GetFolderPath(Environment.SpecialFolder.InternetCache);
            ClearCacheFolder(ieCachePath);

            logger.Log("[ClearIECache] function was called successfully.");
        }
        private void ClearOperaCache()
        {
            string operaCachePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData),
                                                    @"Opera Software\Opera Stable\Cache");
            ClearCacheFolder(operaCachePath);
            logger.Log("[ClearOperaCache] function was called successfully.");
        }

        private void EmptyRecycleBin()
        {
            DialogResult result = MessageBox.Show("Are you sure you want to delete all the items
in the Recycle Bin?", "Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

            // If accepted, continue with the cleaning
            if (result == DialogResult.Yes)
            {
                try
                {
                    // Execute the method with the required parameters
                    uint isSuccess = SHEmptyRecycleBin(IntPtr.Zero, null,
RecycleFlags.SHRB_NOCONFIRMATION);

                    if (isSuccess == 0)
                    {
                        MessageBox.Show("The Recycle Bin has been successfully emptied!",
"Confirmation", MessageBoxButtons.OK, MessageBoxIcon.Information);
```

```
                    }
                    else
                    {
                        MessageBox.Show("Failed to empty the Recycle Bin.", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    }
                }
                catch (Exception ex)
                {
                    // Handle exceptions
                    MessageBox.Show("The Recycle Bin couldn't be emptied: " + ex.Message,
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }

            logger.Log("[EmptyRecycleBin] function was called successfully.");
        }
        private void ClearTempFolder()
        {
            DialogResult result = MessageBox.Show("Are you sure you want to delete all temporary
files?", "Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
            if (result == DialogResult.Yes)
            {
                try
                {

                    string tempFolderPath = Path.GetTempPath();
                    string[] tempFiles = Directory.GetFiles(tempFolderPath);


                    foreach (string tempFile in tempFiles)
                    {
                        File.Delete(tempFile);
                    }

                    MessageBox.Show("Temporary files have been cleaned up successfully.",
"Cleanup Complete", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Error cleaning up temporary files: " + ex.Message, "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }

            logger.Log("[ClearTempFolder] function was called successfully.");
        }
        /// ////////////

        private long GetFolderSize(string folderPath)
        {
            long size = 0;

            try
            {
                DirectoryInfo di = new DirectoryInfo(folderPath);
                foreach (FileInfo file in di.GetFiles())
                {
                    size += file.Length;
                }
                foreach (DirectoryInfo dir in di.GetDirectories())
                {
                    size += GetFolderSize(dir.FullName);
                }
            }
            catch (Exception)
            {
                // Ignore exceptions for folder size calculation
            }
```

```
    logger.Log("[GetFolderSize] function was called successfully.");
    return size;
}
private long GetDirectorySize(string directoryPath)
{
    long directorySize = 0;

    try
    {

        DirectoryInfo dirInfo = new DirectoryInfo(directoryPath);
        foreach (FileInfo fileInfo in dirInfo.GetFiles("*", SearchOption.AllDirectories))
        {
            directorySize += fileInfo.Length;
        }
    }
    catch (Exception)
    {
        // Ignore exceptions for folder size calculation
    }

    logger.Log("[GetDirectorySize] function was called successfully.");
    return directorySize;
}
private void CopyDirectory(string sourceDirPath, string destDirPath)
{

    if (!Directory.Exists(destDirPath))
    {
        Directory.CreateDirectory(destDirPath);
    }

    foreach (string filePath in Directory.GetFiles(sourceDirPath))
    {
        string fileName = Path.GetFileName(filePath);
        string destFilePath = Path.Combine(destDirPath, fileName);
        File.Copy(filePath, destFilePath, true);
    }

    foreach (string subDirPath in Directory.GetDirectories(sourceDirPath))
    {
        string subDirName = new DirectoryInfo(subDirPath).Name;
        string destSubDirPath = Path.Combine(destDirPath, subDirName);
        CopyDirectory(subDirPath, destSubDirPath);
        logger.Log("[CopyDirectory] function was called successfully.");
    }

    logger.Log("[CopyDirectory] function was called successfully.");
}

//MAKING BACKUPS:
private enum BrowserType
{
    Chrome,
    Firefox,
    Edge,
    InternetExplorer,
    Opera
}
private void BackupBrowserCache(string backupFolderPath, BrowserType browserType)
{
    string cachePath = "";

    switch (browserType)
    {
        case BrowserType.Chrome:
```

```
                    cachePath =
Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + @"\Google\Chrome\User
Data\Default\Cache";
                    logger.Log("[GetFolderPath] function was called successfully.");
                    break;
                case BrowserType.Firefox:
                    cachePath =
Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) +
@"\Mozilla\Firefox\Profiles";
                    logger.Log("[GetFolderPath] function was called successfully.");
                    break;
                case BrowserType.Edge:
                    cachePath =
Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) +
@"\Microsoft\Edge\User Data\Default\Cache";
                    logger.Log("[GetFolderPath] function was called successfully.");
                    break;
                case BrowserType.InternetExplorer:
                    cachePath =
Environment.GetFolderPath(Environment.SpecialFolder.InternetCache);
                    logger.Log("[GetFolderPath] function was called successfully.");
                    break;
                case BrowserType.Opera:
                    cachePath =
Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + @"\Opera
Software\Opera Stable\Cache";
                    logger.Log("[GetFolderPath] function was called successfully.");
                    break;
                default:
                    break;
            }

            if (!string.IsNullOrEmpty(cachePath))
            {
                string destinationPath = Path.Combine(backupFolderPath, browserType.ToString() +
"Cache");
                CopyDirectory(cachePath, destinationPath);
                logger.Log("[CopyDirectory] function was called successfully.");
            }

            logger.Log("[BackupBrowserCache] function was called successfully.");
        }
        private void BackupTempFiles(string backupFolderPath)
        {
            string tempFolderPath = Path.GetTempPath();
            string tempBackupPath = Path.Combine(backupFolderPath, "TempFiles");

            CopyDirectory(tempFolderPath, tempBackupPath);
            logger.Log("[CopyDirectory] function was called successfully.");
            logger.Log("[BackupTempFiles] function was called successfully.");
        }
        private void BackupRecycleBin(string backupFolderPath)
        {
            DriveInfo[] drives = DriveInfo.GetDrives();
            foreach (DriveInfo drive in drives)
            {
                if (drive.DriveType == DriveType.Fixed || drive.DriveType == DriveType.Removable)
                {
                    string recycleBinPath = Path.Combine(drive.RootDirectory.FullName,
"$Recycle.Bin");
                    string backupRecycleBinPath = Path.Combine(backupFolderPath, "RecycleBin");
                    CopyDirectory(recycleBinPath, backupRecycleBinPath);
                }
            }
            logger.Log("[BackupRecycleBin] function was called successfully.");
        }
        ////////////////////

        //MAKING TXT INFO OF FILES:
```

```
        private void BrowserCacheInfo(string backupFolderPath, BrowserType browserType)
        {
            string cachePath = "";

            switch (browserType)
            {
                case BrowserType.Chrome:
                    cachePath =
Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + @"\Google\Chrome\User
Data\Default\Cache";
                    logger.Log("[GetFolderPath] function was called successfully.");
                    break;
                case BrowserType.Firefox:
                    cachePath =
Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) +
@"\Mozilla\Firefox\Profiles";
                    logger.Log("[GetFolderPath] function was called successfully.");
                    break;
                case BrowserType.Edge:
                    cachePath =
Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) +
@"\Microsoft\Edge\User Data\Default\Cache";
                    logger.Log("[GetFolderPath] function was called successfully.");
                    break;
                case BrowserType.InternetExplorer:
                    cachePath =
Environment.GetFolderPath(Environment.SpecialFolder.InternetCache);
                    logger.Log("[GetFolderPath] function was called successfully.");
                    break;
                case BrowserType.Opera:
                    cachePath =
Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + @"\Opera
Software\Opera Stable\Cache";
                    logger.Log("[GetFolderPath] function was called successfully.");
                    break;
                default:
                    break;
            }

            if (!string.IsNullOrEmpty(cachePath))
            {
                string cacheInfoFilePath = Path.Combine(backupFolderPath, browserType.ToString()
+ "CacheInfo.txt");
                WriteFileInfoToText(cachePath, cacheInfoFilePath);
                logger.Log("[WriteFileInfoToText] function was called successfully.");
            }

            logger.Log("[BrowserCacheInfo] function was called successfully.");
        }
        private void TempFilesInfo(string backupFolderPath)
        {
            string tempFolderPath = Path.GetTempPath();
            string tempInfoFilePath = Path.Combine(backupFolderPath, "TempFilesInfo.txt");

            WriteFileInfoToText(tempFolderPath, tempInfoFilePath);
            logger.Log("[WriteFileInfoToText] function was called successfully.");

            logger.Log("[TempFilesInfo] function was called successfully.");
        }
        private void RecycleBinInfo(string backupFolderPath)
        {
            DriveInfo[] drives = DriveInfo.GetDrives();
            foreach (DriveInfo drive in drives)
            {
                if (drive.DriveType == DriveType.Fixed || drive.DriveType == DriveType.Removable)
                {
                    string recycleBinPath = Path.Combine(drive.RootDirectory.FullName,
"$Recycle.Bin");
```

```
                    string recycleBinInfoFilePath = Path.Combine(backupFolderPath,
"RecycleBinInfo.txt");
                    WriteFileInfoToText(recycleBinPath, recycleBinInfoFilePath);
                }
            }

            logger.Log("[RecycleBinInfo] function was called successfully.");
        }
        private void WriteFileInfoToText(string directoryPath, string outputFilePath)
        {
            try
            {
                using (StreamWriter writer = new StreamWriter(outputFilePath))
                {
                    writer.WriteLine($"Files in directory: {directoryPath}");
                    WriteFilesInDirectoryToText(directoryPath, writer);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Failed to write file info to {outputFilePath}: {ex.Message}",
"Backup Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            logger.Log("[WriteFileInfoToText] function was called successfully.");
        }
        private void WriteFilesInDirectoryToText(string directoryPath, StreamWriter writer)
        {
            foreach (string file in Directory.GetFiles(directoryPath))
            {
                FileInfo fileInfo = new FileInfo(file);
                writer.WriteLine($"Name: |{fileInfo.Name}| Size: |{fileInfo.Length}| bytes");
            }

            foreach (string subDirectory in Directory.GetDirectories(directoryPath))
            {
                WriteFilesInDirectoryToText(subDirectory, writer);
            }

            logger.Log("[WriteFilesInDirectoryToText] function was called successfully.");
        }
        ////////////////////
    }
}
```

DuplicateForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Security.Cryptography;

namespace SimpleCleaner
{
    public partial class DuplicateForm : Form
    {
        private string selectedFolderPath;
        private List<string> duplicateFilePaths = new List<string>();

        Logger logger = new Logger("DFLog.txt");

        public DuplicateForm()
```

```
        {
            InitializeComponent();
            this.FormClosing += new FormClosingEventHandler(FormClosingFunction);
            //logger.Log("[DuplicateForm] was loaded successfully.");
        }

        private void BackButton_Click(object sender, EventArgs e)
        {
            logger.Log("[Go Back] button was pressed.");
            this.Visible = false;
            CleaningForm newForm = new CleaningForm();
            newForm.Show();
            logger.Log("[Go Back] button was worked successfully.");
        }

        private bool FAQFormOpened = false;
        private void FaqButton_Click(object sender, EventArgs e)
        {
            logger.Log("[FAQ] button was pressed.");
            if (!FAQFormOpened)
            {
                FAQForm newForm = new FAQForm();
                newForm.FormClosed += (s, args) => FAQFormOpened = false; // Set FAQFormOpened to
false when the form is closed
                newForm.Show();
                FAQFormOpened = true;
                logger.Log("[FAQ] button worked successfully.");
            }
            else
            {
                logger.Log("[FAQ] button was already pressed once.");
            }
        }

        private void ChooseDirectoryButton_Click(object sender, EventArgs e)
        {
            logger.Log("[Choose directory] button was pressed.");
            using (var folderBrowserDialog = new FolderBrowserDialog())
            {
                DialogResult result = folderBrowserDialog.ShowDialog();

                if (result == DialogResult.OK &&
!string.IsNullOrWhiteSpace(folderBrowserDialog.SelectedPath))
                {
                    selectedFolderPath = folderBrowserDialog.SelectedPath;
                    logger.Log("Directory was chosen.");
                    MessageBox.Show($"Selected folder: {selectedFolderPath}", "Directory
information", MessageBoxButtons.OK, MessageBoxIcon.Information);
                    DirectoryPathInfo.Text = selectedFolderPath;
                }
            }
            logger.Log("[Choose directory] button was worked successfully.");
        }
        private async void ChooseDriveButton_Click(object sender, EventArgs e)
        {
            logger.Log("[Check for file types] button was pressed.");
            using (var folderBrowserDialog = new FolderBrowserDialog())
            {
                DialogResult result = folderBrowserDialog.ShowDialog();

                if (result == DialogResult.OK &&
!string.IsNullOrWhiteSpace(folderBrowserDialog.SelectedPath))
                {
                    selectedFolderPath = folderBrowserDialog.SelectedPath;
                    logger.Log("Directory was chosen.");
                    MessageBox.Show($"Selected folder: {selectedFolderPath}", "Directory
information", MessageBoxButtons.OK, MessageBoxIcon.Information);
                    DrivePathInfo.Text = selectedFolderPath;
```

```
                    DFProgressBar.Value = 0;
                    await SimulateLoadingAsync(300);

                    DFResultGridView.Rows.Clear();
                    DFResultGridView.ColumnCount = 3;
                    DFResultGridView.Columns[0].Name = "File type:";
                    DFResultGridView.Columns[1].Name = "Files count:";
                    DFResultGridView.Columns[2].Name = "Files total size (MB):";
                    DFResultGridView.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;

                    Dictionary<string, (int count, double totalSize)> fileCountByCategory =
GetFileCountByCategory(selectedFolderPath);
                    logger.Log("[GetCategoryByExtension] function was called successfully.");
                    DisplayResultsInDataGridView(fileCountByCategory);

                    DFResultGridView.Rows.Add("", "", "");
                    double totalSize = CalculateTotalFiles();
                    DFResultGridView.Rows.Add("", "Amount of files:", totalSize.ToString());
                }
            }

            logger.Log("[Check for file types] button was worked successfully.");
        }
        private async void SearchButton_Click(object sender, EventArgs e)
        {
            logger.Log("[Search] button was pressed.");
            if (string.IsNullOrEmpty(selectedFolderPath))
            {
                MessageBox.Show("Choose directory at first!", "Warning", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
                return;
            }

            DFProgressBar.Value = 0;
            await SimulateLoadingAsync(400);

            DFResultGridView.Rows.Clear();
            DFResultGridView.ColumnCount = 3;
            DFResultGridView.Columns[0].Name = "File name:";
            DFResultGridView.Columns[1].Name = "File extension:";
            DFResultGridView.Columns[2].Name = "File size (MB):";
            DFResultGridView.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;

            SearchForDuplicates(selectedFolderPath);
            DisplayDuplicates(duplicateFilePaths);
            int totalDuplicates = (DFResultGridView.RowCount - 1);
            DFResultGridView.Rows.Add("", "", "");
            DFResultGridView.Rows.Add("", "Amount of duplicates:", totalDuplicates);

            logger.Log("[Search] button was worked successfully.");
        }
        private async void ClearButton_Click(object sender, EventArgs e)
        {
            logger.Log("[Clear!] button was pressed.");
            if (DFResultGridView.RowCount == 0 || DFResultGridView.RowCount <= 3)
            {
                MessageBox.Show("There is nothing to clean!", "Warning", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
                return;
            }
            if (CheckForAccessibility(DFResultGridView))
            {
                return;
            }

            DialogResult result = MessageBox.Show("Do you want to backup duplicates before
cleaning?", "Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
            if (result == DialogResult.Yes)
            {
```

```
                BackupDuplicateFiles(duplicateFilePaths);
                DFProgressBar.Value = 0;
                await SimulateLoadingAsync(350);
                ClearDuplicateFiles(duplicateFilePaths);
                int totalDuplicates = (DFResultGridView.RowCount - 1);
                DFResultGridView.Rows.Add("", "", "");
                DFResultGridView.Rows.Add("", "Amount of duplicates:", totalDuplicates);
            }
            if (result == DialogResult.No)
            {
                DFProgressBar.Value = 0;
                await SimulateLoadingAsync(350);
                ClearDuplicateFiles(duplicateFilePaths);
                int totalDuplicates = (DFResultGridView.RowCount - 1);
                DFResultGridView.Rows.Add("", "", "");
                DFResultGridView.Rows.Add("", "Amount of duplicates:", totalDuplicates);
            }

            logger.Log("[Clear!] button was worked successfully.");
        }

        private async Task SimulateLoadingAsync(int durationMilliseconds)
        {
            const int updateInterval = 10;
            int numUpdates = durationMilliseconds / updateInterval;
            int percentIncrement = 10 / numUpdates;

            for (int i = 0; i < numUpdates; i++)
            {
                int percentComplete = (i + 1) * percentIncrement;
                DFProgressBar.PerformStep();

                await Task.Delay(updateInterval);
            }
        }

        //WORK WITH DUPLICATES:
        private void SearchForDuplicates(string directory)
        {
            Dictionary<string, List<string>> filesByHash = new Dictionary<string,
List<string>>();
            duplicateFilePaths.Clear();

            foreach (string filePath in Directory.GetFiles(directory, "*",
SearchOption.AllDirectories))
            {
                try
                {
                    using (var stream = File.OpenRead(filePath))
                    {
                        string fileHash = GetFileHash(stream);

                        if (!filesByHash.ContainsKey(fileHash))
                        {
                            filesByHash[fileHash] = new List<string>();
                        }
                        else
                        {
                            duplicateFilePaths.Add(filePath);
                        }

                        filesByHash[fileHash].Add(filePath);
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show($"Scanning failed: {ex.Message}", "Scanning Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    return;
```

```
                }
            }

            logger.Log("[SearchForDuplicates] function was called successfully.");
        }
        private string GetFileHash(Stream stream)
        {
            using (var md5 = MD5.Create())
            {
                byte[] hash = md5.ComputeHash(stream);
                logger.Log("[GetFileHash] function was called successfully.");
                return BitConverter.ToString(hash).Replace("-", "").ToLowerInvariant();
            }
        }
        private void DisplayDuplicates(List<string> duplicateFilePaths)
        {
            DFResultGridView.Rows.Clear();

            foreach (string filePath in duplicateFilePaths)
            {
                string fileName = Path.GetFileName(filePath);
                long fileSizeBytes = new FileInfo(filePath).Length;
                double fileSizeMB = Math.Round((double)fileSizeBytes / (1024 * 1024), 2);
                string fileExtension = Path.GetExtension(filePath);
                DFResultGridView.Rows.Add(fileName, fileExtension, fileSizeMB);
            }

            logger.Log("[DisplayDuplicates] function was called successfully.");
        }
        /// ////////////////

        //WORK WITH FILE TYPES:
        private Dictionary<string, (int count, double totalSize)> GetFileCountByCategory(string
diskPath)
        {
            Dictionary<string, (int count, double totalSize)> fileCountAndSizeByCategory = new
Dictionary<string, (int, double)>();

            try
            {
                string[] allFiles = Directory.GetFiles(diskPath, "*.*",
SearchOption.AllDirectories);

                foreach (string filePath in allFiles)
                {
                    try
                    {
                        string extension = Path.GetExtension(filePath).ToLower();
                        string category = GetCategoryByExtension(extension);
                        long fileSizeInBytes = new FileInfo(filePath).Length;
                        double fileSizeInMegabytes = fileSizeInBytes / (1024.0 * 1024.0);

                        if (fileCountAndSizeByCategory.ContainsKey(category))
                        {
                            var (count, totalSize) = fileCountAndSizeByCategory[category];
                            fileCountAndSizeByCategory[category] = (count + 1, totalSize +
fileSizeInMegabytes);
                        }
                        else
                        {
                            fileCountAndSizeByCategory[category] = (1, fileSizeInMegabytes);
                        }
                    }
                    catch (UnauthorizedAccessException)
                    {
                        // Ignore exceptions for folder size calculation
                    }
                }
            }
```

```
            catch (Exception ex)
            {
                MessageBox.Show($"Error: {ex.Message}", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }

            logger.Log("[GetFileCountByCategory] function was called successfully.");
            return fileCountAndSizeByCategory;
        }
        private string GetCategoryByExtension(string extension)
        {
            switch (extension)
            {
                case ".mp3":
                case ".wav":
                case ".flac":
                case ".aac":
                case ".m4a":
                case ".ogg":
                case ".wma":
                case ".aiff":
                    return "Music";
                case ".jpg":
                case ".jpeg":
                case ".jfif":
                case ".pjpeg":
                case ".pjp":
                case ".png":
                case ".gif":
                case ".apng":
                case ".avif":
                case ".svg":
                case ".webp":
                case ".bmp":
                case ".ico":
                case ".cur":
                case ".tif":
                case ".tiff":
                    return "Photos";
                case ".txt":
                case ".doc":
                case ".docx":
                case ".html":
                case ".htm":
                case ".odt":
                case ".pdf":
                case ".xls":
                case ".xlsx":
                case ".ods":
                case ".ppt":
                case ".pptx":
                    return "Documents";
                case ".mp4":
                case ".avi":
                case ".mov":
                case ".wmv":
                case ".mkv":
                case ".flv":
                case ".mpeg":
                case ".m4v":
                case ".webm":
                    return "Videos";
                case ".zip":
                case ".rar":
                case ".7z":
                case ".tar":
                case ".gz":
                case ".bz2":
                case ".xz":
```

```
                case ".tgz":
                case ".tar.gz":
                case ".tar.bz2":
                case ".tar.xz":
                    return "Archived";
                default:
                    return "Other";
            }
        }
        private void DisplayResultsInDataGridView(Dictionary<string, (int count, double
totalSize)> results)
        {
            DFResultGridView.Rows.Clear();

            foreach (var pair in results)
            {
                DFResultGridView.Rows.Add(pair.Key, pair.Value.count,
pair.Value.totalSize.ToString("0.##"));
            }

            logger.Log("[DisplayResultsInDataGridView] function was called successfully.");
        }
        private double CalculateTotalFiles()
        {
            int totalFilesAmount = 0;

            foreach (DataGridViewRow row in DFResultGridView.Rows)
            {
                if (row.Cells[0].Value != null && row.Cells[1].Value != null)
                {
                    string cacheType = row.Cells[0].Value.ToString();
                    string sizeString = row.Cells[1].Value.ToString();

                    if (int.TryParse(sizeString.Replace(" MB", ""), out int sizeMB))
                    {
                        totalFilesAmount += sizeMB;
                    }
                }
            }

            logger.Log("[CalculateTotalFiles] function was called successfully.");
            return totalFilesAmount;
        }
        /// ////////////////

        //CLEANING:
        private void ClearDuplicateFiles(List<string> duplicateFilePaths)
        {
            foreach (string filePath in duplicateFilePaths)
            {
                try
                {
                    File.Delete(filePath);
                }
                catch (Exception ex)
                {
                    MessageBox.Show($"Failed to delete file {filePath}: {ex.Message}", "Deletion
Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
            MessageBox.Show("Duplicates have been cleared successfully.", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information);
            duplicateFilePaths.Clear();
            DisplayDuplicates(duplicateFilePaths);

            logger.Log("[ClearDuplicateFiles] function was called successfully.");
        }
        private bool CheckForAccessibility(DataGridView dataGridView)
        {
```

```
            bool column0Exists = false;
            bool column1Exists = false;
            bool column2Exists = false;

            foreach (DataGridViewColumn column in dataGridView.Columns)
            {
                if (column.Name == "File type:")
                {
                    column0Exists = true;
                }
                else if (column.Name == "Files count:")
                {
                    column1Exists = true;
                }
                else if (column.Name == "Files total size (MB):")
                {
                    column2Exists = true;
                }
            }

            if (column0Exists || column1Exists || column2Exists)
            {
                MessageBox.Show("You can't clean file types!", "Warning", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
                return true;
            }

            logger.Log("[CheckForAccessibility] function was called successfully.");
            return false;
        }
        /// ////////////////

        //MAKING BACKUPS:
        private void BackupDuplicateFiles(List<string> duplicateFilePaths)
        {
            using (var folderBrowserDialog = new FolderBrowserDialog())
            {
                DialogResult result = folderBrowserDialog.ShowDialog();

                if (result == DialogResult.OK &&
!string.IsNullOrWhiteSpace(folderBrowserDialog.SelectedPath))
                {
                    string backupDirectory = folderBrowserDialog.SelectedPath;

                    try
                    {
                        //backupDirectory = Path.Combine(backupDirectory, "DuplicatesBackup_" +
DateTime.Now.ToString("yyyyMMdd_HHmmss"));
                        Directory.CreateDirectory(backupDirectory);

                        // Копіюємо кожен файл-дублікат у директорію бекапу
                        foreach (string filePath in duplicateFilePaths)
                        {
                            string fileName = Path.GetFileName(filePath);
                            string destFilePath = Path.Combine(backupDirectory, fileName);
                            File.Copy(filePath, destFilePath, true);
                        }

                        MessageBox.Show($"Backup completed successfully!", "Backup Information",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show($"Backup failed: {ex.Message}", "Backup Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    }
                }
            }
```

```
            logger.Log("[BackupDuplicateFiles] function was called successfully.");
        }
        /// ////////////////

        //FORM CLOSER:
        private void FormClosingFunction(object sender, FormClosingEventArgs e)
        {
            if (e.CloseReason == CloseReason.UserClosing)
            {
                Application.Exit();
            }
        }
    }
}
```

FAQForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace SimpleCleaner
{
    public partial class FAQForm : Form
    {
        Logger logger = new Logger("FQLog.txt");

        public FAQForm()
        {
            InitializeComponent();
            logger.Log("[FAQForm] was loaded successfully.");
        }

    }
}
```

Logger.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SimpleCleaner
{
    public class Logger
    {
        private string logFilePath;

        public Logger(string logFilePath)
        {
            this.logFilePath = logFilePath;
        }

        public void Log(string message)
        {
            try
            {
                using (StreamWriter writer = File.AppendText(logFilePath))
                {
```

```
                    writer.WriteLine($"{DateTime.Now}: {message}");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error occurred while logging: {ex.Message}");
            }
        }
    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace SimpleCleaner
{
    static class Program
    {
        /// <summary>
        ///  The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.SetHighDpiMode(HighDpiMode.SystemAware);
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new CleaningForm());
            //Application.Run(new DuplicateForm());
            //Application.Run(new FAQForm());
        }
    }
}
```