Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка програмного забезпечення системи аналізу файлів на диску і побудови звіту»

Виконав: студент 4 курсу
групи 4ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Назарчук Б.Ю.
(прізвище та ініціали)
Керівник: к.т.н., доц., каф. ПЗ Черноволик Г.С.
(прізвище та ініціали)
Рецензент: к.т.н., доц., каф.ОТ Тарновський М.Г.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
« 21 » березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Назарчуку Богдану Юрійовичу

1. Тема роботи – «Розробка програмного забезпечення системи аналізу файлів на диску і побудови звіту»

Керівник роботи: Черноволик Галина Олександрівна, к.т.н., доцент каф. ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

2. Строк подання студентом роботи 2 червня 2025 р.

3. Вихідні дані до роботи: мова програмування C++ та інтегроване середовище розробки Visual Studio 2022.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз розвитку програмного застосунку для аналізу файлів та побудови звіту; розробка алгоритму програмного застосунку; розробка програмних функцій та методів реалізації програмного додатку для аналізу файлів; тестування програмного забезпечення ; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження, наукова новизна, практична цінність одержаних результатів; порівняльний аналіз аналогів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	к.т.н., доц., каф. ПЗ Черноволик Г.О.	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз розвитку програмного застосунку для аналізу файлів та побудови звіту	25.03.25 – 03.04.25	вик
2	Розробка моделі та алгоритм програмного застосунку	04.04.25 – 15.04.25	вик
3	Варіантний аналіз та обгрунтування вибору засобів програмної реалізації	16.04.25 – 17.04.25	вик
4	Розробка програмних модулів та методів реалізації програмного додатку для аналізу файлів	18.04.25 – 27.04.25	вик
5	Тестування програмного забезпечення	28.04.25 – 11.05.25	вик
6	Оформлення матеріалів до захисту БКР	12.05.25 – 30.05.25	вик

Студент _____ Назарчук Б.Ю.
(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи _____ Черноволик Г.О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.932.2:004.93

Назарчук Б.Ю. Розробка програмного забезпечення системи аналізу файлів на диску і побудови звіту: бакалаврська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 78 с.

На укр. мові. Бібліогр. : 28 назв ; рис. : 24 ; табл. 4.

Було розроблено програмне забезпечення системи аналізу файлів на диску і побудови звіту. Розроблено програмне забезпечення, яке відображає проаналізовані файли у головному вікні розробленого застосунку. Для реалізації програми використовуються мова програмування C++ та інтегроване середовище розробки Visual Studio 2022.

Ключові слова: аналіз, диск, звіт.

ABSTRACT

UDC 004.932.2:004.93

Nazarchuk B.Y. Development of software for the system of analysing files on a disc and building a report: bachelor's qualification work in speciality 121 - software engineering, educational programme - software engineering. Vinnytsia: VNTU, 2025. 78 c.

In Ukrainian. Bibliography: 28 titles; fig.: 24; tab. 4.

Software for analysing files on a disc and generating a report was developed. The software displays the analysed files in the main window of the developed application. The C++ programming language and the Visual Studio 2022 integrated development environment are used to implement the application.

Keywords: analysis, disc, report.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Розробка програмного застосунку для аналізу файлів на диску і побудови звіту є актуальною через зростання обсягу даних, що створює потребу в ефективних інструментах для їх управління [2]. Такий застосунок допомагає оптимізувати використання дискового простору, виявляючи великі та непотрібні файли, а також підвищує безпеку даних, ідентифікуючи потенційно небезпечні файли. Автоматизація процесів аналізу файлів зекономить час і зусилля користувачів.

Проте існує кілька проблем, які необхідно вирішити. Зокрема, різноманітність файлових форматів потребує розробки універсального інструменту, що може працювати з різними типами файлів [3]. Велика кількість файлів вимагає оптимізації алгоритмів для забезпечення високої продуктивності програми. Важливо також створити зручний інтерфейс, щоб користувачі могли легко використовувати програму і отримувати необхідну інформацію. Програма повинна бути точною і надійною, щоб уникнути помилок при аналізі файлів і формуванні звітів. Оскільки аналіз файлів може включати роботу з конфіденційною інформацією, необхідно забезпечити високий рівень безпеки даних [4].

Тому актуальною є розробка програмного забезпечення, яке дозволяє автоматизовано аналізувати вміст файлової системи, зручно відображати ключові параметри файлів та формувати статистичні звіти. У контексті зростання обсягів цифрових даних, що зберігаються на персональних та робочих комп’ютерах, виникає необхідність у доступних інструментах для контролю за використанням дискового простору [5]. Створення подібного застосунку сприятиме підвищенню продуктивності користувача, оптимізації збереження інформації та своєчасному виявленню надлишкових або непотрібних файлів.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є покращення організації та ефективності роботи з файловою системою шляхом створення інструменту для швидкого аналізу вмісту директорій, виявлення зайвих або великих файлів, а також побудови інформативного звіту на основі зібраних даних.

Основними завданнями дослідження є:

- розробити архітектуру програмного застосунку для аналізу файлів;
- розробити структуру додатку;
- розробити дизайн додатку;
- розробити методи, що відповідають за графічний інтерфейс та логіку програми;
- розробити алгоритм роботи системи;
- розробити функціонал системи;
- провести тестування розробленої системи.

Об’єкт дослідження – процес розробки програмного забезпечення для аналізу файлової системи та генерації звітів.

Предмет дослідження – методи та інструменти розробки програмного забезпечення, зокрема принципи об’єктно-орієнтованого програмування мовою C++, а також засоби пошуку й аналізу файлів на диску, що використовуються у створенні застосунку.

Методи дослідження. У процесі досліджень використовувались: дискретна математика, теорія графів, теорія алгоритмів для побудови ефективних засобів аналізу файлової структури; методи структурного аналізу даних для класифікації та оцінки характеристик файлів; а також принципи об’єктно-орієнтованого програмування для розробки програмного забезпечення з модульною архітектурою і засобами формування звітів.

Новизна отриманих результатів.

Подальшого розвитку набув метод детермінованого аналізу файлових об’єктів, у якому всі етапи виконуються послідовно та передбачувано. Новизна полягає у використанні ітеративного методу сканування замість рекурсивного, що дозволяє стабільно працювати навіть при великій кількості вкладених папок.

Також був застосований простий спосіб пошуку дублікатів за іменем і розміром файлів, що дає змогу швидко отримувати результати без великих витрат ресурсів.

Ще однією особливістю є відображення результатів у вигляді простої таблиці без дерева папок, що робить інтерфейс зрозумілішим. Звіт формується прямо в програмі без PDF або CSV, що спрощує реалізацію та використання. Усе це разом створює ефективний та зручний підхід до аналізу файлів на диску.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що розроблено програмне забезпечення, алгоритми та методи якого дозволяють здійснювати автоматизований аналіз файлів на диску, класифікувати їх за визначеними критеріями та формувати структуровані звіти з візуалізацією результатів, що значно спрощує управління даними та оптимізує процеси їх обробки.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належить «Використання програмного забезпечення системи аналізу файлів на диску».

Апробація матеріалів бакалаврської кваліфікаційної роботи. Основні положення БКР доповідалися та обговорювалися на Міжнародній конференції «Молодь в науці: дослідження, проблеми, перспективи Вінницького національного технічного університету (МН ВНТУ–2025)».

Публікації. Основні результати дослідження були опубліковані в матеріалах конференції [1].

1 АНАЛІЗ РОЗВИТКУ ПРОГРАМНОГО ЗАСТОСУНКУ ДЛЯ АНАЛІЗУ ФАЙЛІВ ТА ПОБУДОВИ ЗВІТУ

1.1 Аналіз стану програмних застосунків для аналізу файлів на диску

У сучасних інформаційних системах обсяг даних, що зберігаються на комп'ютерах, серверах та інших носіях, стрімко зростає. Цей процес супроводжується ускладненням структури збережених даних, зростанням кількості тимчасових, дубльованих, застарілих або навіть небезпечних файлів. Така ситуація створює додаткові труднощі для користувачів, адміністраторів та спеціалістів із кібербезпеки, оскільки ефективне управління файлами та оптимізація дискового простору потребує точного, швидкого і наочного аналізу великого обсягу інформації. Програмні застосунки для аналізу файлів на диску відіграють у цьому процесі ключову роль, забезпечуючи автоматизацію рутинних завдань і створення детальної аналітики щодо вмісту файлової системи.

Розвиток таких програм зумовлений необхідністю підвищення продуктивності як окремих комп’ютерів, так і цілих інформаційних систем. Раніше процес управління файлами відбувався вручну або за допомогою консольних інструментів, що вимагало від користувача високого рівня технічної обізнаності. Натомість сучасні інструменти для аналізу файлової системи мають інтуїтивно зрозумілий інтерфейс і надають можливість за декілька хвилин отримати повну картину використання простору, виявити проблемні файли або каталоги, оцінити їхню значимість та визначити, які з них доцільно видалити або перемістити.

З технічної точки зору, подібні програмні рішення функціонують за допомогою стандартних інтерфейсів операційної системи, що дозволяє їм отримувати доступ до структур каталогів і метаданих файлів. У більшості випадків застосовується алгоритм ітеративного або рекурсивного обходу файлового дерева, під час якого зчитується інформація про розміри, дати створення та останньої модифікації, розширення файлів тощо. Зібрані дані

структуруються і подаються у зручному для користувача форматі: у вигляді таблиць, списків, графіків або інтерактивних панелей. Такий підхід дозволяє швидко визначити, які файли або папки займають найбільше місця, які дублюються, які мають нехарактерні або застарілі властивості.

Програмне забезпечення для аналізу файлів також нерідко виконує функції автоматичного формування звітів, які можуть зберігатися в різних форматах – наприклад, CSV для подальшого використання у табличних редакторах, PDF для представлення результатів керівництву або HTML для інтеграції в інформаційні панелі. Деякі системи мають функціонал, що дозволяє надсилати ці звіти на електронну пошту або зберігати у хмарному сховищі. Таке розширення можливостей є особливо актуальним для великих компаній або IT-відділів, які працюють з великим масивом даних і повинні звітувати про їхній стан у регламентованому форматі.

Значну роль відіграє і візуальна складова аналізу. Зокрема, графічна інтерпретація використання простору – за допомогою кольорових діаграм, блок-схем або прямокутних карт – дозволяє спростити сприйняття великої кількості інформації та швидко виявити критичні області на диску. Завдяки інтерактивності таких інструментів користувачі можуть миттєво отримувати додаткову інформацію при наведенні або натисканні на елемент, що значно прискорює процес ухвалення рішень щодо подальших дій із файлами.

Іншою важливою характеристикою є підтримка роботи з мережевими сховищами, зовнішніми носіями та інтеграція із системами моніторингу або управління. Це дозволяє використовувати такі застосунки не лише для локального аналізу, але й у масштабах великих інфраструктур, де потрібно оцінювати стан сотень і тисяч пристроїв. Виявлення дубльованих файлів у різних точках системи, розрахунок потенційно зайвих обсягів даних, а також оцінка ступеня фрагментації або розкиданості інформації – усе це можна реалізувати за допомогою професійних інструментів аналізу файлової системи [6].

Суттєвим аспектом, який визначає зручність та ефективність використання таких програм, є можливість налаштування критеріїв аналізу відповідно до

потреб користувача. Сучасні інструменти дозволяють здійснювати детальну фільтрацію даних за різними параметрами: типами файлів, діапазоном розмірів, датами створення або останнього доступу. Це дозволяє зосередитись лише на релевантних елементах і не витрачати час на обробку другорядної інформації. У деяких випадках також реалізовано підтримку регулярних виразів або шаблонів, що забезпечує гнучке формулювання запитів при великій кількості об’єктів.

Ще однією корисною функцією є можливість попереднього перегляду вмісту файлів або отримання короткої інформації без відкриття сторонніх застосунків. Це особливо важливо під час аналізу великої кількості документів, коли потрібно швидко оцінити зміст або тип даних. Наприклад, у програмах із підтримкою мультимедійного аналізу можливе розпізнавання типу контенту (аудіо, відео, текст) навіть без розширення файлу.

Не менш важливою є підтримка багатоплатформенності. Чимало сучасних застосунків мають версії для Windows, Linux, macOS, а також мобільні додатки або веб-інтерфейси. Це дає змогу виконувати аналіз диску не лише з локального пристрою, а й віддалено — через браузер чи серверну платформу. Такий підхід дозволяє централізовано обслуговувати мережі комп’ютерів або корпоративні системи, що складаються з багатьох вузлів.

У контексті корпоративного використання варто згадати також про роль логування та журналювання подій. Застосунки можуть зберігати історію дій користувача, зміни у структурі файлів, попередні результати сканування або створені звіти. Це дозволяє у разі потреби провести аналіз динаміки змін, порівняти попередній та поточний стан файлової системи, а також відновити певні операції або структуру у разі помилки.

Варто також зазначити, що останнім часом такі застосунки дедалі частіше інтегруються з технологіями штучного інтелекту або машинного навчання. Це дозволяє автоматично виявляти нетипову активність, наприклад появу великої кількості однотипних файлів у короткий проміжок часу або збереження даних у нехарактерних місцях. Також можливе автоматичне формування рекомендацій – наприклад, пропозицій видалити тимчасові файли, об'єднати дублікати або

архівувати старі документи. Такий підхід значно підвищує ефективність управління інформацією та дозволяє зменшити ризики, пов’язані з людським фактором.

Окрему увагу заслуговують аспекти інформаційної безпеки. Виявлення файлів із підозрілими розширеннями, прихованими властивостями або тими, що з’явилися на диску без явної участі користувача, може вказувати на потенційну загрозу. У цьому випадку програмні засоби аналізу можуть бути інтегровані з антивірусними модулями, системами виявлення вторгнень або навіть реагування на інциденти. Таким чином, простий на перший погляд інструмент для аналізу структури даних перетворюється на важливий елемент комплексної системи кіберзахисту.

У підсумку можна стверджувати, що програмні засоби для аналізу файлів на диску відіграють дедалі важливішу роль у забезпеченні ефективності, продуктивності та безпеки цифрових систем. Їхнє використання дозволяє не лише зекономити місце на накопичувачах, а й оптимізувати процеси зберігання, запобігти дублюванню даних і своєчасно реагувати на потенційні загрози. Надалі розвиток таких систем передбачає ще глибшу інтеграцію з інтелектуальними механізмами, розширення можливостей адаптивного інтерфейсу, підтримку роботи з великими обсягами даних у режимі реального часу та розробку спеціалізованих рішень для різних галузей – від освіти до промисловості

1.2 Аналіз аналогів і їх основні характеристики

Під час розробки власного додатка Disk Analyzer було проведено детальний аналіз існуючих програмних рішень у сфері візуалізації використання дискового простору. Метою було визначення найкращих підходів до реалізації функціональності, зручності інтерфейсу та продуктивності.

Серед розглянутих варіантів для порівняння були відібрані такі популярні аналоги, а також власна розробка – Disk Analyzer:

1. WinDirStat;
2. WizTree;

3. Disk Inventory X;

4. TreeSize Free;

5. Disk Analyzer.

Першим розглянемо аналог «WinDirStat» – це безкоштовна програма для аналізу використання місця на диску в ОС Windows [7]. Вона показує дерево каталогів з використанням диска у відсотках та список розширень файлів з зазначенням у відсотках, скільки місця займає кожне розширення.. Розробником WinDirStat є компанія WinDirStat Team, а останнє оновлення програми відбулося 14 жовтня 2023 року. Головне вікно даного програмного застосунку зображено на рисунку 1.1.

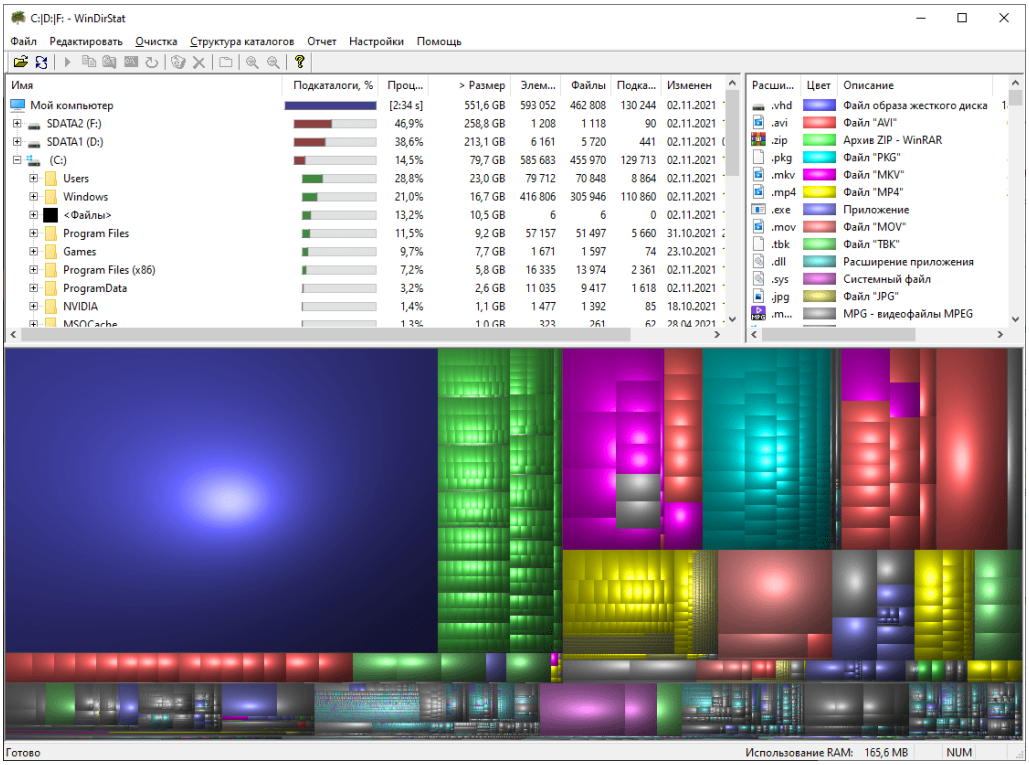


Рисунок 1.1 – Зображення головного вікна WinDirStat

Наступним розглянемо аналог, що зображено на рисунку 1.2, «WizTree» – це платна програма для Windows, яка використовується для аналізу дискового простору. Програма сканує диски та папки, а потім представляє дані у вигляді таблиці, де можна побачити розмір, ім'я, тип файлу та дату останньої модифікації кожного файлу. WizTree також має карту дерева, яка візуалізує дані у вигляді

кольорових прямокутників. Останнє оновлення програми відбулося у листопаді 2023 року [8].

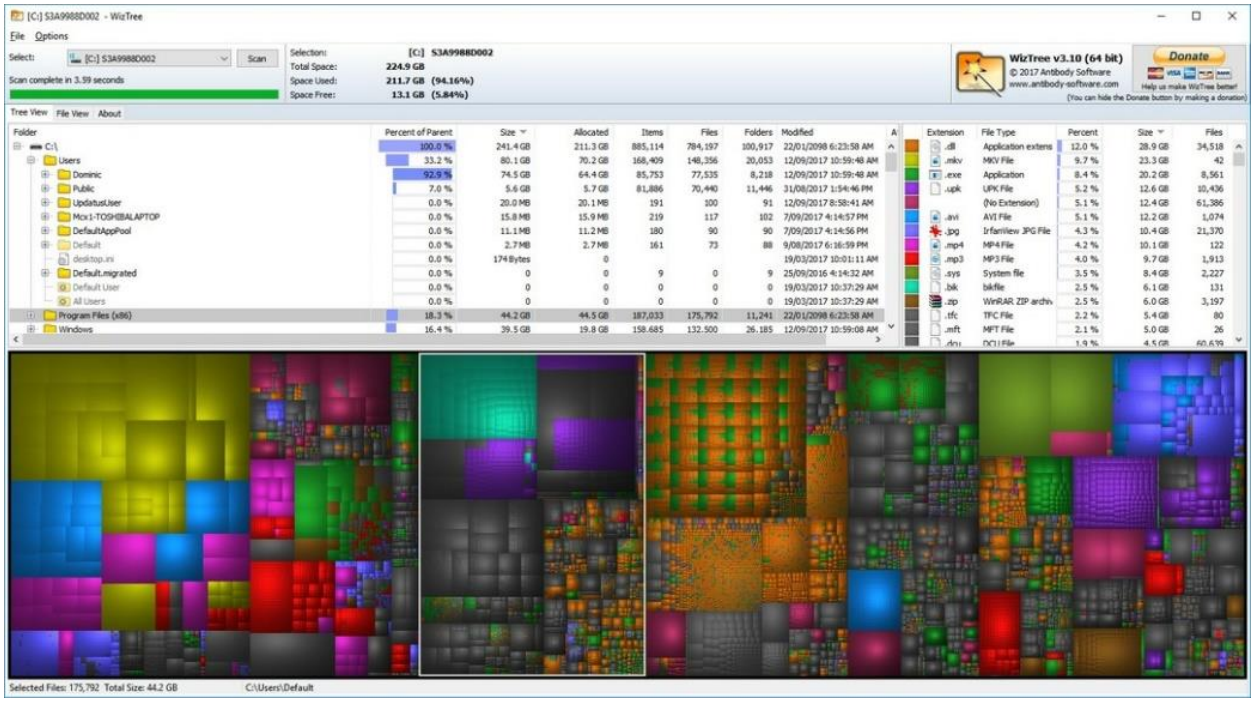


Рисунок 1.2 – Зображення головного вікна WizTree

Disk Inventory X – це безкоштовна програма для macOS, яка використовується для аналізу дискового простору [9]. Програма сканує диски та папки, а потім представляє дані у вигляді таблиці, де можна побачити розмір, ім'я, тип файлу та дату останньої модифікації кожного файлу. Disk Inventory X також має карту дерева, яка візуалізує дані у вигляді кольорових прямокутників. В останнє оновлювалась у 2023 р., а її розробником є Derlien.com. Інтерфейс даного програмного застосунку зображено на рисунку 1.3. Крім основних функцій аналізу дискового простору, Disk Inventory X дозволяє користувачам здійснювати сортування та фільтрацію файлів за різними критеріями. Це значно спрощує процес пошуку великих файлів. Програма також підтримує функцію видалення файлів безпосередньо з інтерфейсу, що робить її зручною для очищення дискового простору.

Завдяки своїй простоті у використанні та наочності, вона особливо корисна для користувачів, які хочуть швидко звільнити місце на жорсткому диску. Disk

Inventory X є надійним інструментом як для досвідчених користувачів, так і для новачків.

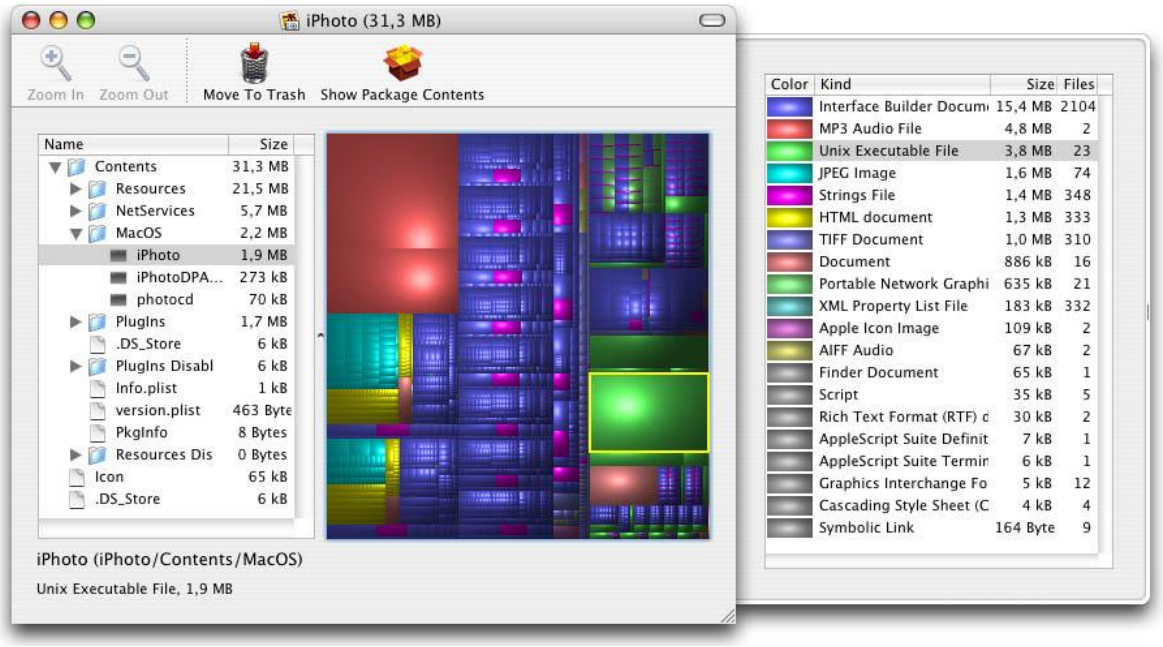


Рисунок 1.3 – Зображення головного вікна Disk Inventory X

TreeSize Free – це безкоштовна програма для Windows, яка використовується для аналізу дискового простору [10]. Як і його аналоги, програма сканує диски та папки, а потім представляє дані у вигляді таблиці, де можна побачити розмір, ім'я, тип файлу та дату останньої модифікації кожного файлу. TreeSize Free також має карту дерева, яка візуалізує дані у вигляді кольорових прямокутників. Головне вікно можна побачити на рисунку 1.4. TreeSize Free дозволяє швидко виявити великі файли та папки, що займають найбільше місця на диску, завдяки гнучкому сортуванню та зручному відображенню інформації. Програма підтримує інтеграцію з контекстним меню провідника Windows, що дає змогу запускати аналіз прямо з файлової системи. Вона також може працювати з мережевими дисками та зовнішніми накопичувачами, що робить її універсальним інструментом для адміністраторів і звичайних користувачів. TreeSize Free має простий та інтуїтивно зрозумілий інтерфейс, який підходить навіть для недосвідчених користувачів. Незважаючи

на свою безкоштовну ліцензію, програма забезпечує достатній обсяг функціональності для ефективного аналізу та очищення дисків.

Рисунок 1.4 – Зображення головного вікна TreeSize Free

Аналіз функціональностей аналогів та власного застосунку, що показано у таблиці 1.1, дозволяє виявити як спільні, так і унікальні особливості кожного з програмних продуктів. Такий порівняльний підхід є важливою складовою етапу проєктування, оскільки він дає змогу об'єктивно оцінити конкурентне середовище та краще зрозуміти потреби кінцевих користувачів. Це допомагає визначити переваги та недоліки власного застосунку порівняно з аналогами, зокрема в аспектах функціональності, зручності інтерфейсу, продуктивності, гнучкості налаштувань та можливостей розширення.

На основі такого аналізу можна зробити висновки щодо ключових напрямків удосконалення програмного забезпечення. Наприклад, якщо деякі аналоги пропонують більш гнучку систему фільтрації або мають вбудовану підтримку хмарних сховищ — це може стати поштовхом для розширення можливостей власного рішення. Також важливим є врахування відгуків користувачів конкурентних продуктів — це дозволяє не лише запобігти

повторенню чужих помилок, а й впровадити перевірені рішення.

Таким чином, порівняльний аналіз сприяє стратегічному плануванню розвитку додатку, підвищує його конкурентоспроможність та дає змогу створити більш досконалий і корисний для користувача продукт.

Таблиця 1.1 – Порівняння характеристик функціоналу аналогів

Критерії оцінювання	Disk Analyzer (Власна розробка)	WizTree	WinDirStat	TreeSize Free	Disk Inventor y X
Сканування та аналіз файлів та папок	1	1	1	1	1
Створення детальних звітів	1	0	1	1	1
Фільтрація та сортування результатів	1	1	0	0	1
Пошук дублікатів файлів	0	1	0	0	0
Збереження звіту у файл	0	0	0	1	0
Включення додаткової інформації про файли	1	0	0	0	0
Сумарний коефіцієнт	4	3	2	3	3

Проаналізувавши та порівнявши властивості аналогів, можна зробити висновок про актуальність розробки власного програмного забезпечення. Створення власної програми дозволить врахувати специфічні потреби та вимоги користувачів, які можуть не бути реалізовані в існуючих рішеннях.

1.3 Аналіз методів розв'язання задачі

Задача аналізу файлів на диску охоплює декілька взаємопов'язаних етапів: сканування файлової системи, обробку отриманих даних, виявлення певних закономірностей та формування підсумкового звіту для користувача. Кожен із цих етапів має свою специфіку реалізації та може бути виконаний різними методами, які відрізняються за рівнем складності, швидкодією, точністю, вимогами до ресурсів і ступенем адаптації до різних середовищ. У цьому розділі розглядаються найбільш поширені підходи, що використовуються у програмному забезпеченні для аналізу дискового простору, а також обґрунтовується вибір методів, застосованих у рамках даної бакалаврської кваліфікаційної роботи.

Перший і базовий етап – це сканування вмісту файлової системи. Його мета полягає у повному проходженні всіх файлів і папок, починаючи з заданого кореневого каталогу. Така операція потребує ефективного алгоритму обходу структури директорій, що зазвичай реалізується двома основними способами: рекурсивним або ітеративним [11].

Рекурсивний підхід є класичним методом, який реалізується через послідовні виклики функцій самого себе. Це дозволяє створювати простий, короткий і зрозумілий код, який легко підтримується. Однак, на практиці при роботі з глибоко вкладеними структурами каталогів рекурсія може стати джерелом проблем. Наприклад, у випадку занадто великої глибини вкладеності виникає ризик переповнення стеку викликів, що може призвести до аварійного завершення роботи програми або навіть системи. Особливо це актуально в мовах програмування, де існують жорсткі обмеження на глибину рекурсії.

Натомість ітеративні методи обходу структури директорій реалізуються за

допомогою циклів і структур даних, таких як черги або стеки, що дозволяє уникнути проблем, пов’язаних із переповненням стеку. Такий підхід є більш гнучким, стабільним і краще масштабується в умовах великої кількості папок і файлів. Саме тому в рамках цієї роботи було обрано ітеративний алгоритм обходу директорій. Він забезпечив оптимальний баланс між надійністю, швидкістю виконання та простотою реалізації.

Крім технічного аспекту, важливо зазначити і вплив вибору алгоритму на взаємодію з користувачем. Наприклад, ітеративні методи краще підходять для реалізації багатопоточності або асинхронного сканування, що дозволяє паралельно обробляти великі масиви даних без блокування інтерфейсу користувача.

Після завершення етапу сканування отримані дані потребують обробки та структуризації. Саме цей крок дозволяє витягти корисну інформацію з великої кількості необроблених елементів файлової системи. Найчастіше користувач очікує побачити найбільші файли, дізнатися, які типи файлів займають найбільше місця, а також отримати інструмент для виявлення дублікатів.

Процес обробки включає кілька ключових операцій. По-перше, це сортування – наприклад, за розміром файлів, що дозволяє швидко виявити найбільш ресурсоємні об’єкти. По-друге, групування – за типами, розширеннями або шляхами. Така класифікація дозволяє користувачу краще зрозуміти структуру збережених даних [12].

Окремим завданням є пошук дублікатів. Найпростіший підхід – це знаходження файлів з однаковими іменами та розмірами. Хоча цей метод не забезпечує абсолютну точність, він досить ефективний у разі необхідності швидкого аналізу. Його перевагою є висока швидкодія, відсутність необхідності в додатковій обробці вмісту файлів. Водночас, для підвищення точності можна застосувати обчислення хеш-сум – наприклад, алгоритмів MD5 або SHA-1. Це дозволяє виявити повні дублі навіть у разі, якщо імена чи розширення були змінені. Проте такий підхід суттєво ускладнює реалізацію та підвищує навантаження на процесор і пам’ять.

У розробленому застосунку для збереження простоти реалізації та забезпечення швидкого аналізу було реалізовано саме базовий метод виявлення дублікатів за іменем і розміром. Такий компроміс дозволив досягти задовільного рівня точності при збереженні високої продуктивності.

Третім важливим компонентом системи є візуалізація результатів. Як показує практика, саме цей етап найбільше впливає на користувацький досвід. Від того, наскільки зручно та інтуїтивно подано результати, залежить, наскільки ефективно користувач зможе інтерпретувати інформацію та прийняти подальші рішення [13].

У багатьох подібних застосунках застосовуються деревоподібні структури для відображення ієрархії папок. Це дійсно може бути зручно, однак при великій кількості вкладених директорій користувач може легко загубитися в обсязі інформації. В межах цієї роботи було вирішено не використовувати ієрархічні структури, а натомість реалізувати просту табличну форму подання. Такий підхід дозволяє швидко охопити поглядом усю ключову інформацію та забезпечує легку сортування і фільтрацію.

Кожен рядок у таблиці відповідає окремому файлу, а стовпці містять найважливіші параметри: назву, тип, розмір, дату створення або змінення. При цьому таблиця адаптується до розміру вікна, підтримує пошук за ключовими словами та інтерактивні фільтри, що значно покращує взаємодію з додатком.

Завершальний етап – це створення підсумкового звіту. У сучасному програмному забезпеченні для цього можуть використовуватись різні формати: HTML, PDF, CSV тощо. Кожен із них має свої переваги та недоліки. CSV – легкий і простий, зручний для подальшої обробки у табличних редакторах, проте має обмежені можливості з візуального оформлення. PDF, навпаки, є статичним і придатним для друку та презентації, але його генерація потребує використання спеціальних бібліотек і шаблонів [14].

У межах цієї роботи було прийнято рішення реалізувати звіт безпосередньо в інтерфейсі програми. Це дозволяє обійтися без зовнішніх залежностей, спростити підтримку коду, а також надати користувачу

адаптивний, зрозумілий та структурований підсумок виконаного аналізу. Такий підхід значно підвищує гнучкість програми, особливо з урахуванням того, що вона може працювати на різних платформах.

У подальшому, якщо виникне необхідність, можна реалізувати експорт у популярні формати, проте в базовій версії цього не потрібно для задоволення основних функціональних вимог.

У результаті аналізу наявних підходів та практичного тестування різних реалізацій було обґрунтовано вибір основних методів для створення програмного забезпечення аналізу файлів. До них належать: ітеративне сканування файлової системи, базова обробка даних з використанням простого сортування та фільтрації, візуалізація результатів у табличному форматі та генерація звіту безпосередньо в інтерфейсі програми. Обрані методи забезпечують прийнятний баланс між простотою реалізації, продуктивністю і якістю користувацького досвіду.

У майбутньому можливе розширення функціоналу, наприклад, шляхом додавання експорту звітів у PDF/CSV, використання хеш-функцій для точнішого пошуку дублікатів, інтеграції з хмарними сервісами або реалізації багатопоточності для пришвидшення обробки великих обсягів даних. Але навіть у своєму базовому вигляді запропоноване рішення повністю відповідає поставленим вимогам до системи та може бути успішно використане для аналізу структури файлової системи.

1.4 Постановка задач дослідження

Провівши аналіз методів розв’язання поставленої задачі, аналіз стану програмних застосунків для аналізу файлів на диску, порівняння аналогів було поставлено такі задачі дослідження:

- розробити архітектуру програмного застосунку для аналізу файлів;
- розробити структуру додатку;
- розробити дизайн додатку;
- розробити методи, що відповідають за графічний інтерфейс та логіку

програми;

- розробити алгоритм роботи системи;
- розробити функціонал системи;
- провести тестування розробленої системи.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі бакалаврської кваліфікаційної роботи було проаналізовано сучасний стан програмних застосунків, призначених для аналізу файлів на диску. Розглянуто їхнє функціональне призначення, сфери використання та особливості реалізації.

Було проведено детальний аналіз існуючих аналогів, визначено їхні ключові характеристики, переваги й недоліки, що дозволило виділити вимоги до майбутнього застосунку.

Окрему увагу приділено методам розв’язання задачі аналізу файлової системи, включаючи алгоритмічні та програмні підходи. На завершення розділу сформульовано постановку задачі дослідження, яка стала основою для подальшого проєктування та реалізації програмного додатку.

2 РОЗРОБКА СТРУКТУРНИХ СХЕМ ТА АЛГОРИТМІВ ПРОГРАМНОГО ЗАСТОСУНКУ

2.1 Розробка структурних схем застосунку

Структурна схема програмного застосунку містить основні логічні блоки, які є візуалізацією модулів, що використані в реалізації застосунку. Кожен блок представляє окремий функціональний компонент системи — інтерфейс користувача, модуль обробки даних, модуль сортування, генератор звітів, а також допоміжні елементи, такі як вікно статистики чи інформаційне вікно «Про програму». Схема наочно демонструє взаємозв’язки між компонентами, порядок обміну даними та логіку їх взаємодії. Такий підхід дозволяє краще зрозуміти внутрішню структуру програмного забезпечення та спростити процес його підтримки, тестування й подальшої модернізації. Структурна схема зображена на рисунку 2.1.

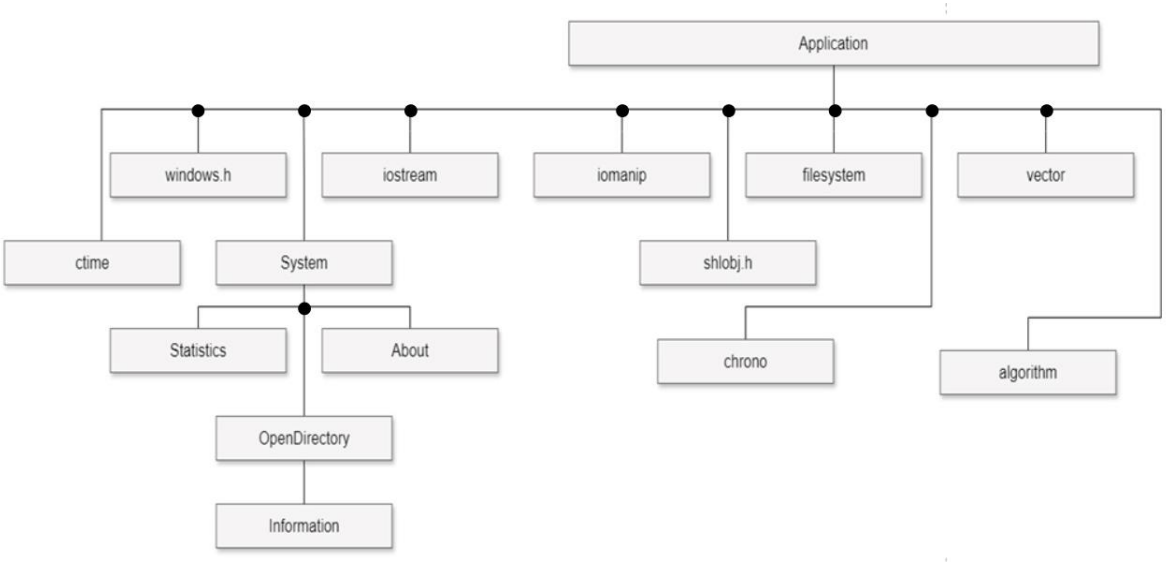


Рисунок 2.1 – Структурна схема програмного застосунку

На структурній схемі головного вікна, зображено основні компоненти, які задіяні для виконання головних задач програмного застосунку (див. рисунок 2.2).

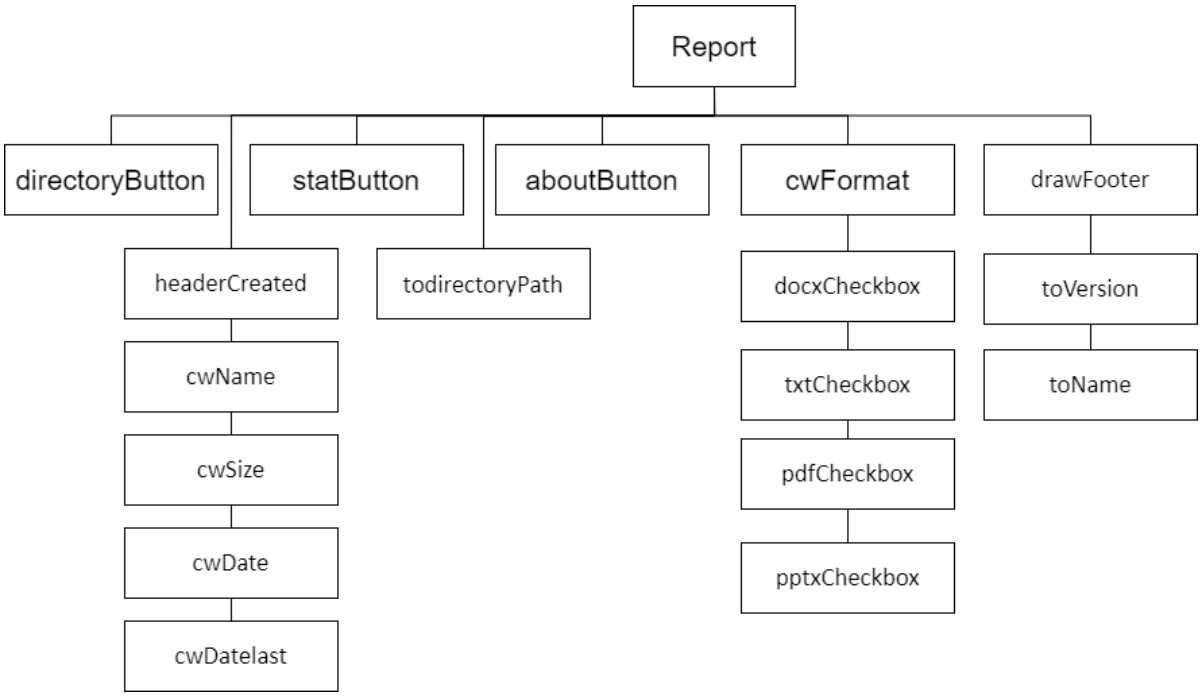


Рисунок 2.2 – Структурна схема головного вікна програмного застосунку

Основна інформація про програмний застосунок буде представлена у вікні «Про програму» (див. рисунок 2.3).

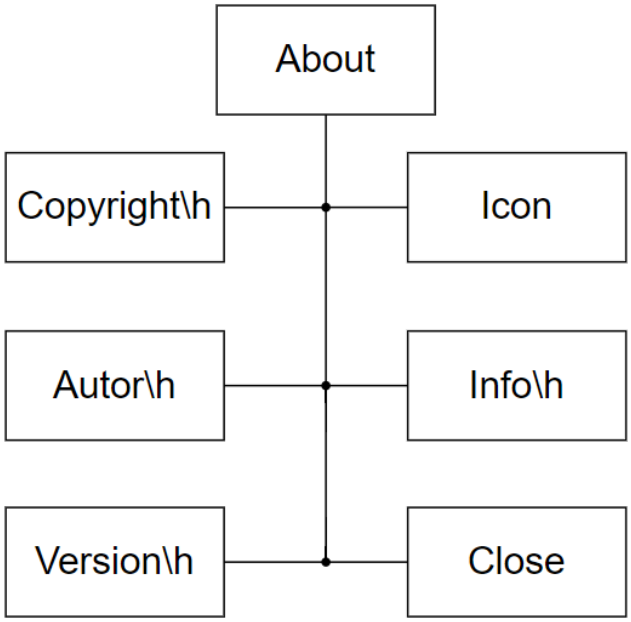


Рисунок 2.3 – Структурна схема вікна «Про програму»

Інтерфейс користувача складається із двох вікон: основного та "Про програму". У основному вікні знаходиться робоча область, а також панель меню інструментів, необхідних для користування програмним застосунком. Вікно "Про програму" містить усю необхідну інформацію про програму, включаючи назву, версію, автора та тему розробки.

Головне вікно програмного застосунку має такі елементи :

1) Report – головне вікно програми;
2) directoryButton – кнопка «Вибрати Директорію»;
3) statButton – кнопка «Статистика»;
4) aboutButton – кнопка «Про програму»;
5) todirectoryPath – відображає шлях до поточної директорії;
6) headerCreated – виведення заголовків;
7) cwName – кнопка «Ім’я файлу»;
8) cwSize – кнопка «Розмір файлу»;
9) cwDate – кнопка «Дата створення »;
10) cwDatelast – кнопка «Дата останньої модифікації»;
11) cwFormat – відображає заголовок Формат;
12) docxCheckbox – кнопка «.docx»;
13) txtCheckbox – кнопка «.txt»;
14) pdfCheckbox – кнопка «.pdf»;
15) pptxCheckbox – кнопка «.pptx»;
16) drawFooter – роздільний контейнер нижнього простору
17) toVersion – відображає автора програми
18) toName – відображає назву програми

На рисунку 2.4 зображено схему головного вікна програмного застосунку з позначеними елементами інтерфейсу, що відображають основні функціональні блоки системи. Тут можна побачити розташування панелі керування, області виводу результатів аналізу, навігаційного меню, а також додаткових елементів, що забезпечують взаємодію користувача з програмою.

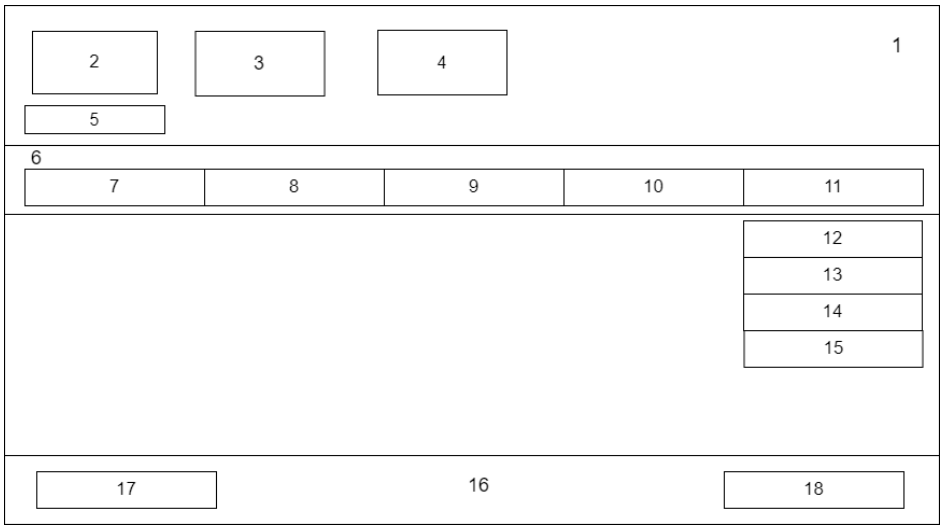


Рисунок 2.4 – Графічна схема головного вікна застосунку

Вікно, з головною інформацією «Про програму» містить (див. рисунок 2.5):

1) About – форма «Про програму»;
2) Icon – компонент із зображенням логотипу.
3) Info\h – текстовий компонент для відображення загальної інформації про програму;
4) Author\h – текст для відображення інформації про розробника;
5) Copyright\ h – текстовий компонент відображення прав на застосунок;
6) Version\h – текст для відображення версії застосунку;
7) Close – кнопка «OK» для закриття форми.

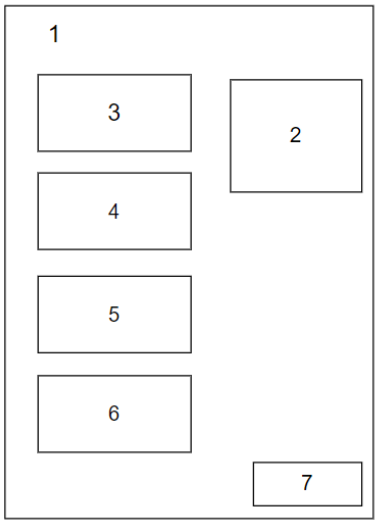


Рисунок 2.5 – Графічна схема вікна «Про програму»

2.2 Метод детермінованого аналізу файлів на диску

Метод детермінованого аналізу файлів на диску, який було реалізовано в межах цієї роботи, базується на послідовному виконанні чітко визначених кроків, що дозволяє досягти стабільного та надійного результату. Кожен етап виконується згідно з попередньо визначеним алгоритмом, використовуючи перевірені технічні підходи, що гарантують відтворюваність та узгодженість дій у процесі аналізу.

Першим і ключовим етапом є сканування файлової системи. У даному випадку використано ітеративний метод обходу директорій, що дозволяє уникнути проблем, пов’язаних з обмеженнями рекурсивних алгоритмів. Такий підхід особливо ефективний при обробці складних структур із великою глибиною вкладеності, де традиційне рекурсивне проходження може призводити до помилок, таких як переповнення стеку або надмірне використання ресурсів системи. Ітеративна реалізація гарантує стабільну роботу навіть у найскладніших конфігураціях файлової ієрархії.

Після завершення сканування система переходить до етапу обробки зібраної інформації. Цей процес включає сортування файлів за їх розміром, групування за типами з урахуванням розширень, а також виявлення дублікатів. Для останнього завдання використано простий, але ефективний метод – порівняння імен та розмірів файлів. Такий підхід дає змогу досягти балансу між швидкістю обробки та точністю результатів, що є важливим фактором у роботі з великими обсягами даних.

Результати аналізу подаються у зручному для сприйняття вигляді – у форматі звичайної табличної структури. Відмова від побудови складних деревоподібних відображень каталогів була свідомим рішенням для спрощення інтерфейсу користувача. Це дозволяє швидко зорієнтуватися у результатах навіть при великій кількості даних та полегшує взаємодію з системою. Підсумкова інформація автоматично виводиться безпосередньо у вікні програми у вигляді адаптивного інтерактивного звіту. Такий спосіб подання даних дозволяє обійтися

без необхідності експорту у сторонні формати, наприклад PDF чи CSV, зберігаючи при цьому гнучкість та доступність результатів для користувача.

На рисунку 2.6 наочно представлено послідовність роботи методу, що ілюструє описані вище етапи.

Рисунок 2.6 – Діаграма діяльності методу

Отже, реалізований у даній роботі підхід можна охарактеризувати як приклад класичного детермінованого методу. Усі кроки виконуються у строго визначеному порядку, що дозволяє забезпечити передбачуваність та стабільність результатів. Застосовані рішення є не лише ефективними, але й достатньо

простими у реалізації, що робить їх придатними для подальшої модифікації та масштабування в межах більш складних систем.

2.3 Розробка алгоритму системи

При створенні програмного забезпечення для додатка, орієнтованого для роботи з аналізом файлів, важливо детально спланувати всі компоненти системи. У цьому контексті була розроблена UML діаграма діяльності, яка ілюструє ключові взаємодії користувача з додатком (див.рисунок 2.7).

Рисунок 2.7 – Діаграма діяльності розробленого додатка

На діаграмі діяльності представлено процес взаємодії користувача з програмним додатком для виведення аналізу файлів. Процес починається з вибору форматів фалів для пошуку і вибору директорії. Після вибору директорії програма аналізує вміст директорії та виводить оброблені дані на екран

користувача.

Користувач може вибрати чи виводити статистику. Весь процес завершується отриманням результату роботи аналізу файлів у вигляді звіту.

Розроблений програмний застосунок працює наступним чином. При завантажені необхідного файлу, користувач отримує доступ до роботи з розробленим додатком.

Далі користувач вибирає відповідну директорію, яку потрібно опрацювати розробленим додатком.

Блок-схема алгоритму роботи застосунку представлена на рисунку 2.8.

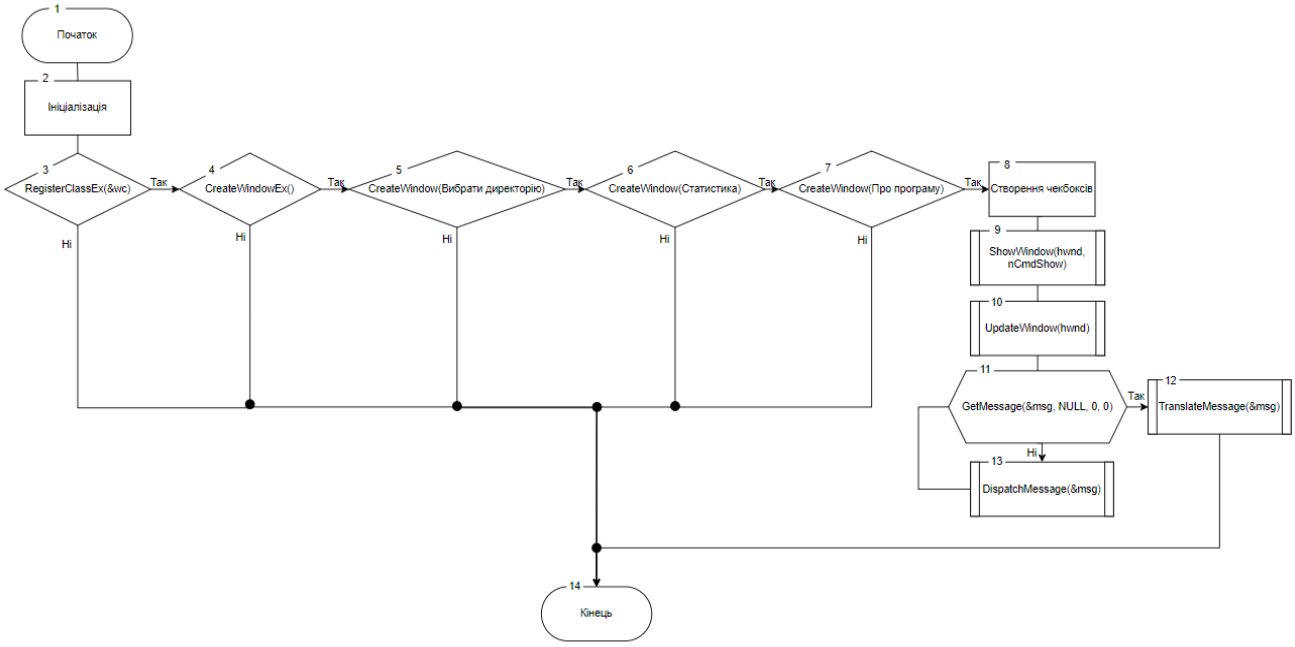


Рисунок 2.8 – Блок-схема алгоритму роботи застосунку

Після вибору директорії програма миттєво зчитує всі файли в обраній директорії (без вкладених папок) і відображає їх у табличному вигляді.

Є можливість фільтрації за форматами файлів – користувач може обрати, які саме типи файлів він хоче бачити у звіті (.docx, .pdf тощо). Дані оновлюються динамічно відповідно до обраних фільтрів.

Також по натисканні кнопки для виклику статистики виводиться статистика файлів у директорії.

Всі описані вище дії не потребують ніяких додаткових дій від користувача, окрім вибору директорії, що дає легкий користувацький досвід в сфері аналізу данних програмним застосунком.

2.4 Висновки

У процесі дослідження було розглянуто метод структурованого аналізу, який дозволив сформувати точну і логічну основу для аналізу поведінки системи та її компонентів. Цей метод забезпечив передбачуваність результатів, що особливо важливо для систем із чітко визначеними вхідними даними та очікуваними виходами.

На основі отриманих аналітичних даних була розроблена структурна схема застосунку, що деталізує основні модулі системи, їхні функції та взаємозв’язки. Це стало фундаментом для подальшого проектування логіки роботи програмного забезпечення та забезпечило цілісне уявлення про архітектуру системи.

Завершальним етапом стало створення алгоритму функціонування системи, який відображає послідовність дій під час виконання основних операцій. Розроблений алгоритм дозволяє ефективно реалізувати ключові функції, забезпечує узгодженість між компонентами та оптимізує обробку даних у межах запропонованої моделі.

У сукупності виконані етапи дозволили закласти надійну основу для реалізації програмного продукту, що відповідає вимогам до стабільності, логічної побудови та функціональності.

3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ ТА МЕТОДІВ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ДОДАТКУ ДЛЯ АНАЛІЗУ ФАЙЛІВ

3.1 Варіантний аналіз та обгрунтування вибору засобів програмної реалізації

Для порівняльного аналізу мов програмування обрано C++, Java та C#. Цей вибір дозволяє оцінити їхні можливості, ефективність та застосування в різних сферах розробки програмного забезпечення, також це дозволить обрати найзручнішу мову для розробки власного ПЗ.

C++ забезпечує високу продуктивність завдяки низькорівневому доступу до апаратного забезпечення та ручному управлінню пам'яттю [15]. Мова є крос-платформною, тому використовується для розробки застосунків на Windows, Linux, macOS, а також для вбудованих систем. C++ підтримує об'єктно-орієнтоване, процедурне та генеричне програмування, що дозволяє гнучко управляти апаратним забезпеченням і ресурсами. Велика і активна спільнота забезпечує багатий набір ресурсів для навчання та підтримки.

Java має нижчу продуктивність порівняно з C++ через оверхед віртуалізації JVM, проте забезпечує автоматичне управління пам'яттю, що спрощує розробку. Вона є крос-платформною завдяки JVM, дозволяючи запускати застосунки на Windows, Linux, macOS і Android. Java проста у використанні, має великий набір стандартних бібліотек і підтримує об'єктно-орієнтоване програмування [16]. Велика спільнота та численні ресурси для навчання сприяють розвитку і підтримці розробників. Крім того, Java широко використовується в корпоративних застосунків, веб-розробці та мобільних застосунках завдяки своїй надійності та масштабованості.

C# демонструє хорошу продуктивність на платформі .NET з певним оверхедом і автоматичним управлінням пам'яттю [17]. З виходом .NET Core та .NET 5+ мова стала крос-платформною, підтримуючи Windows, Linux та macOS. C# поєднує простоту Java з потужними можливостями C++, підтримуючи об'єктно-орієнтоване програмування, асинхронне програмування та LINQ.

Активно розвивається завдяки підтримці Microsoft, і спільнота продовжує зростати, забезпечуючи більше ресурсів для навчання та розвитку.

Детальне порівняння функціональності кожної із мов програмування наведено в таблиці 3.1.

Таблиця 3.1 – Порівняння характеристик мов програмування

Характеристика	C++	Java	C#
Продуктивність	1	0	0
Крос-платформеність	1	1	1
Гнучкість	1	0	0
Робота з графічним інтерфейсом	0	1	1
Управління пам’ятю	1	0	0
Наявність обширної кількості бібліотек	1	1	1
Швидкість виконання	1	0	0
Сумарний коефіцієнт	6	3	3

За результатами порівняльного аналізу, C++ було обрано як мову програмування для розробки програмного застосунку «Disk Analyzer».

При розробці програмного додатку постало питання вибору відповідного середовища програмування.

Для вибору середовища програмування розглянемо такі засоби розробки:

Microsoft Visual Studio, Qt Creator, Code::Blocks, які є популярними серед розробників.

Microsoft Visual Studio - це інтегроване середовище розробки (IDE), яке надає широкі можливості для створення різноманітних застосунків на платформі Windows [18]. Воно підтримує різні мови програмування, включаючи C++, C#, Visual Basic і інші. Visual Studio має багатофункціональний інтерфейс, який дозволяє розробникам працювати з проектами будь-якої складності. Крім того, воно має розширену підтримку для платформи .NET, що робить його ідеальним вибором для розробки програм для Windows.

Qt Creator - інтегроване середовище розробки, спеціалізоване на розробці застосунків з використанням фреймворку Qt [19]. Це середовище розробки призначене для створення крос-платформних застосунків, що можуть працювати на різних операційних системах. Qt Creator підтримує мови програмування, такі як C++, QML, JavaScript і Python. Воно має простий інтерфейс та розширені можливості для роботи з графічним інтерфейсом.

Code::Blocks - це вільне та відкрите середовище розробки, яке призначене для створення програм на C, C++ і Fortran [20]. Воно підтримує крос-платформну розробку і має простий інтерфейс. Code::Blocks не має розширеної підтримки для .NET і не такий гнучкий як Visual Studio чи Qt Creator, але воно надає просте та швидке середовище для розробки малих і середніх проектів.

Детальніше порівняння функціональності кожного із середовищ розробки наведено у таблиці 3.2.

Таблиця 3.2 Порівняльний аналіз середовища розробки

Характеристика	Visual Studio	Qt Creator	Code:: Blocks
1	2	3	4
Відкритий вихідний код та спільнота підтримки	0	1	1
Розширена підтримка .NET	1	0	0

Продовження таблиці 3.2

1	2	3	4
Масштабованість проектів	1	1	0
Інтегрованість інструментів розробки	1	0	1
Багатофункціональність	1	1	0
Сумарний коефіцієнт	4	3	2

Для розробки було обрано середовище Microsoft Visual Studio через його зручність та найбільшу кількість переваг у порівнянні з іншими доступними середовищами.

3.2 Розробка функцій програмного додатку

Одна з функцій програмного застосунку є «GetDirectoryPathFromUser». Блок-схема функції зображена на рисунку 3.1.

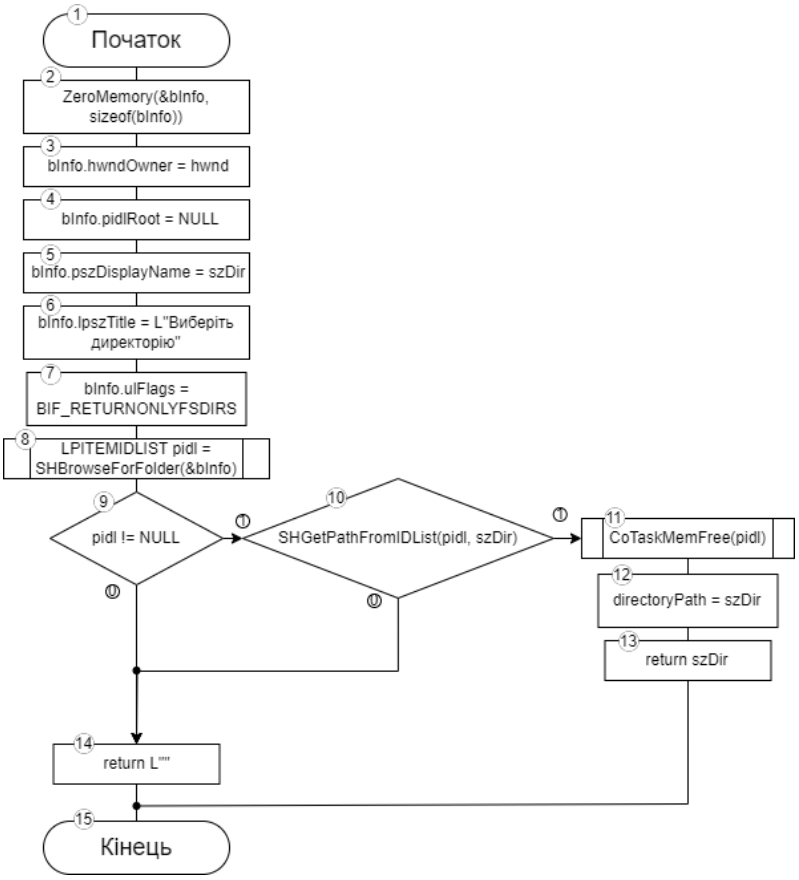


Рисунок 3.1 – Блок-схема функції «GetDirectoryPathFromUser»

Ця функція відкриває діалогове вікно для вибору директорії і повертає шлях до вибраної директорії. Коли користувач натискає кнопку "Вибрати директорію", відкривається діалогове вікно, де він може вибрати потрібну директорію. Після вибору, функція отримує шлях до цієї директорії та повертає його. Якщо користувач не вибрав директорію або сталася помилка, функція повертає порожній рядок.

Функції, реалізовані у програмному забезпеченні «Disk Analyzer», виконують конкретні функції, які забезпечують логіку взаємодії користувача з додатком, а також реалізацію функціоналу збору та відображення інформації про файли. Нижче наведено опис основних методів, що використовуються у застосунку.

1. generateReport() — основна функція, що відповідає за генерацію та виведення звіту на основі аналізу вибраної директорії. Функція активується після того, як користувач обирає директорію та підтверджує запуск процесу. Внутрішньо ця функція звертається до функцій збору статистики, сортування файлів, формує структуру звіту, що включає перелік файлів з їх характеристиками (назва, розмір, дата створення тощо), а також викликає функції відображення отриманих результатів у головному вікні програми. Звіт будується в інтерактивному вигляді з можливістю візуального аналізу інформації. [21]. (див. рисунок 3.2).

```
void generateReport(HWND hwnd) {
    // Отримуємо контекст виведення для роботи з вікном
    HDC hdc = GetDC(hwnd);
    if (hdc == NULL) {
        MessageBox(hwnd, L"Помилка отримання контексту виведення", L"Помилка", MB_OK | MB_ICONERROR);
        return;
    }

    // Налаштування шрифту для виведення тексту
    HFONT hFont = CreateFont(16, 0, 0, 0, FW_NORMAL, FALSE, FALSE, FALSE, ANSI_CHARSET, OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH | FF_DONTCARE, L"Arial");
    SelectObject(hdc, hFont);

    int headerY = 100; // Початкова у-координата заголовка
    int statisticY = headerY + 20; // Початкова у-координата статистики

    // Виведення заголовків
    static bool headerCreated = false;
    if (!headerCreated) {
        CreateWindow(L"STATIC", L"Ім'я файлу", WS_VISIBLE | WS_CHILD, 20, headerY, 180, 20, hwnd, (HMENU)1, NULL, NULL);
        CreateWindow(L"STATIC", L"Розмір файлу", WS_VISIBLE | WS_CHILD, 180, headerY, 150, 20, hwnd, (HMENU)2, NULL, NULL);
        CreateWindow(L"STATIC", L"Дата створення", WS_VISIBLE | WS_CHILD, 330, headerY, 150, 20, hwnd, (HMENU)3, NULL, NULL);
        CreateWindow(L"STATIC", L"Дата останньої модифікації", WS_VISIBLE | WS_CHILD, 480, headerY, 200, 20, hwnd, (HMENU)4, NULL, NULL);
        CreateWindow(L"STATIC", L"Формат", WS_VISIBLE | WS_CHILD, 680, headerY, 80, 20, hwnd, (HMENU)5, NULL, NULL);
        headerCreated = true;
    }
```

Рисунок 3.2 – функція «generateReport()»

2. SortFiles() - функція, яка здійснює сортування файлів у вибраній директорії згідно з критерієм, заданим користувачем. Параметрами сортування можуть бути: ім’я файлу, розмір, дата створення або дата останньої модифікації. Також функція враховує порядок сортування — за зростанням або спаданням. Сортування реалізується із застосуванням стандартних алгоритмів сортування, що адаптовані до структури даних про файли. Після сортування відбувається оновлення виведеної таблиці зі списком файлів. [22]. Функція зображена на рисунку 3.3.

```
void SortFiles(int criteria, bool reverse) {
    switch (criteria) {
    case 1:
        if (reverse) {
            sort(files.begin(), files.end(), [](const FileData& a, const FileData& b) { return a.name > b.name; });
        }
        else {
            sort(files.begin(), files.end(), [](const FileData& a, const FileData& b) { return a.name < b.name; });
        }
        break;
    case 2:
        if (reverse) {
            sort(files.begin(), files.end(), [](const FileData& a, const FileData& b) { return a.size > b.size; });
        }
        else {
            sort(files.begin(), files.end(), [](const FileData& a, const FileData& b) { return a.size < b.size; });
        }
        break;
    case 3:
        if (reverse) {
            sort(files.begin(), files.end(), [](const FileData& a, const FileData& b) { return a.creationTime > b.creationTime; });
        }
        else {
            sort(files.begin(), files.end(), [](const FileData& a, const FileData& b) { return a.creationTime < b.creationTime; });
        }
        break;
    case 4:
        if (reverse) {
            sort(files.begin(), files.end(), [](const FileData& a, const FileData& b) { return a.lastModifiedTime > b.lastModifiedTime; });
        }
        else {
            sort(files.begin(), files.end(), [](const FileData& a, const FileData& b) { return a.lastModifiedTime < b.lastModifiedTime; });
        }
        break;
    }
}
```

Рисунок 3.3 – Функція «SortFiles()»

3. showStatistics() – ця функція виконує обчислення загальної статистики по файлах, що містяться у вибраній директорії. Результатом виконання є такі показники, як: загальна кількість файлів, сумарний розмір усіх файлів, розміри найбільшого та найменшого файлів, а також середній розмір. Функція працює зі збереженим масивом об’єктів-файлів, здійснюючи обхід цього масиву з метою збору необхідних значень. Після розрахунків статистика відображається у відповідному вікні. [23]. (див. рисунок 3.4).

```
void showStatistics(HWND hwnd) {
    if (files.empty()) {
        MessageBox(hwnd, L"Статистика недоступна, оскільки файли ще не завантажені.", L"Помилка", MB_OK | MB_ICONERROR);
        return;
    }

    long long totalSize = 0;
    long long maxSize = 0;
    long long minSize = LLONG_MAX;

    for (const auto& file : files) {
        totalSize += file.size;
        maxSize = max(maxSize, file.size);
        minSize = min(minSize, file.size);
    }

    long long averageSize = files.empty() ? 0 : totalSize / files.size();

    wstring stats = L"Статистика:\nКількість файлів: " + to_wstring(files.size()) +
        L"\nЗагальний розмір файлів: " + to_wstring(totalSize) +
        L"\nНайбільший файл: " + to_wstring(maxSize) +
        L"\nНайменший файл: " + to_wstring(minSize) +
        L"\nСередній розмір файлу: " + to_wstring(averageSize);

    MessageBox(hwnd, stats.c_str(), L"Статистика", MB_OK);
}
```

Рисунок 3.4 – Функція «showStatistics()»

4. ShowAboutDialog() – допоміжна функція, яка викликає діалогове вікно з інформацією про програму, такою як назва, версія, авторство, короткий опис функціональності і призначення. Діалогове вікно викликається через пункт меню «Про програму» та слугує елементом інтерфейсу для ознайомлення користувача з додатком. Функція реалізована у вигляді окремого вікна з кнопкою «Закрити» для повернення до головного інтерфейсу. [24]. (див. рисунок 3.5).

```
public:
    void ShowAboutDialog(HWND hwnd) {
        DialogBoxParam(GetModuleHandle(NULL), MAKEINTRESOURCE(IDD_ABOUTBOX), hwnd, AboutDlgProc, 0);
    }

private:
    static INT_PTR CALLBACK AboutDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {
        switch (message) {
        case WM_INITDIALOG:
            SetDlgItemText(hDlg, IDC_ABOUTTEXT, L"Ця програма реалізовує аналіз файлів на диску і побудову звіту.\nЗастосунок виконанс
            return (INT_PTR)TRUE;
        case WM_COMMAND:
            if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL) {
                EndDialog(hDlg, LOWORD(wParam));
                return (INT_PTR)TRUE;
            }
            break;
        case WM_DROPFILES: {
            HDROP hDrop = (HDROP)wParam;
            WCHAR szFileName[MAX_PATH];
            DragQueryFile(hDrop, 0, szFileName, MAX_PATH);
            DragFinish(hDrop);
                        break;
        }
        }
        return (INT_PTR)FALSE;
```

Рисунок 3.5 – Функція «ShowAboutDialog()»

5. drawFooter() – функція, яка відповідає за побудову нижньої частини вікна (колонтитула), де виводиться службова інформація про програму — зокрема назва додатку та його версія. Функція автоматично викликається при

перезавантаженні або оновленні вікна, що дозволяє зберігати сталість інформації незалежно від дій користувача. Вона також може викликатися у випадках зміни теми або роздільної здатності, щоб адаптувати відображення інтерфейсу. [25]. (див. рисунок 3.6).

```
void drawFooter(HWND hwnd) {
    HDC hdc = GetDC(hwnd);
    if (hdc == NULL) {
        MessageBox(hwnd, L"Помилка отримання контексту виведення", L"Помилка", MB_OK | MB_ICONERROR);
        return;
    }

    HFONT hFont = CreateFont(16, 0, 0, 0, FW_NORMAL, FALSE, FALSE, FALSE, ANSI_CHARSET, OUT_DEFAULT_PRECI
    SelectObject(hdc, hFont);

    RECT rc;
    GetClientRect(hwnd, &rc);

    int footerHeight = 30;
    int footerY = rc.bottom - footerHeight - 5;

    // Зафарбовуємо фон footer'а
    HBRUSH hBrush = CreateSolidBrush(RGB(220, 220, 220)); // Сірий колір
    RECT footerRect = { 0, footerY, rc.right, rc.bottom };
    FillRect(hdc, &footerRect, hBrush);
    DeleteObject(hBrush);

    // Малюємо рамочку
    HPEN hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 0)); // Чорна рамка
    SelectObject(hdc, hPen);
    MoveToEx(hdc, 0, footerY, NULL);
    LineTo(hdc, rc.right, footerY);
    DeleteObject(hPen);

    int textLeftMargin = 20;
    int textRightMargin = 200;
    int textHeight = 20;
    int textY = footerY + (footerHeight - textHeight) / 2;
    SetBkMode(hdc, TRANSPARENT);
    SetTextColor(hdc, RGB(0, 0, 0));
    // Виведення імені автора
    TextOut(hdc, 20, footerY + 7, L"Версія 1.0", wcslen(L"Версія 1.0"));

    // Виведення назви програми
    TextOut(hdc, rc.right - 200, footerY + 7, L"Disk Analyzer", wcslen(L"Disk Analyzer"));

    ReleaseDC(hwnd, hdc);
    DeleteObject(hFont);
}
```

Рисунок 3.6 – Функція «drawFooter()»

Описані вище функції формують основу функціональної логіки застосунку «Disk Analyzer» і забезпечують його інтерактивність, зручність користування та ефективну роботу. Кожна з функція виконує чітко визначене завдання в рамках загальної архітектури програми, що дозволяє досягти високого рівня структурованості коду та полегшує подальше масштабування і підтримку системи. Завдяки модульному підходу реалізації, програмне забезпечення можна легко адаптувати до нових вимог або розширити додатковими функціями, не порушуючи загальної логіки. Це створює надійну основу для подальшого розвитку продукту та його вдосконалення

3.3 Розробка та обґрунтування вибору інтерфейсу

У сучасному світі, який стрімко розвивається в напрямку цифровізації, практично всі програмні продукти мають графічний інтерфейс користувача. Це стало не лише стандартом, а й очікуваною нормою з боку користувачів. Графічний інтерфейс користувача, або ж GUI (Graphical User Interface), – це зручний і ефективний спосіб взаємодії людини з комп'ютером або будь-яким іншим цифровим пристроєм, що ґрунтується на використанні візуальних елементів, таких як вікна, кнопки, меню, іконки тощо [26].

Вікна – прямокутні області на екрані, які можуть містити різний вміст, такий як текст, зображення та інші графічні елементи.

Значки – це невелике графічне зображення, що представляє програму, файл або інший об'єкт. Це полегшує швидкий доступ до цих об'єктів.

Меню – це список опцій або команд, які користувач може вибрати. Меню може бути випадаючим, контекстним або головним. У кожного з них різні функції і спосіб застосування.

Кнопки – це графічний елемент, який можна натиснути, щоб виконати певні дії, такі як запуск програми або збереження файлу.

Панель інструментів – це панель, що містить кнопки та інші елементи керування, які дозволяють швидко виконувати часто використовувані команди.

Форми та поля введення – це поля, які дозволяють користувачам вводити такі дані, як текст або необхідні файли.

Завдяки впровадженню графічного інтерфейсу користувача більше не потрібно запам’ятовувати та вручну вводити складні текстові команди, що було звичним у перші роки розвитку обчислювальної техніки, коли основною формою взаємодії з комп’ютером були командні рядки. Тодішні системи вимагали від користувача знань синтаксису команд і точного їх написання, що значно ускладнювало процес роботи з комп’ютером, особливо для новачків або непрофесійних користувачів. З появою графічного інтерфейсу цей бар’єр було усунуто — тепер усі основні дії можна виконувати за допомогою миші або сенсорного екрана, використовуючи кнопки, меню, вікна та інші візуальні

елементи.

Цей підхід суттєво полегшує роботу з програмами та операційними системами, роблячи її доступнішою для широкого кола користувачів, незалежно від рівня їхньої технічної підготовки. Інтерфейс стає інтуїтивно зрозумілим: навіть люди, які вперше стикаються з новим програмним забезпеченням, можуть швидко розібратися в його функціоналі без потреби звертатися до об’ємної документації. Саме тому вибір графічного інтерфейсу для розробленого програмного забезпечення є цілком логічним та виправданим — він забезпечує не лише зручність і простоту у використанні, але й сприяє підвищенню загальної ефективності роботи користувача.

Варто також враховувати, що якість графічного інтерфейсу прямо впливає на перше враження від додатку та рівень задоволеності користувача. Добре продуманий інтерфейс формує позитивний користувацький досвід, а також підвищує довіру до програмного продукту. Саме тому під час проєктування було дотримано основних принципів юзабіліті — таких як простота і зрозумілість дизайну, мінімалізм, логічне розміщення елементів керування, правильна ієрархія інформації, приємне колірне оформлення, а також обов’язкова наявність зворотного зв’язку. Це можуть бути повідомлення про успішне виконання дії, попередження про помилки або підтвердження введених даних. Такі елементи підвищують зручність взаємодії з додатком і допомагають уникнути непорозумінь або помилок під час його використання.

Якщо говорити безпосередньо про інтерфейс розробленого додатку, то він умовно поділений на три функціональні частини, кожна з яких виконує свою важливу роль у забезпеченні повноцінної взаємодії користувача із системою.

Перша частина є ознайомлювальною – вона надає загальну інформацію про призначення додатку, описує його основні можливості та дозволяє користувачу швидко зрозуміти, з яким типом програмного забезпечення він має справу. Такий інформаційний блок допомагає сформувати правильне очікування та орієнтує користувача перед початком роботи.

Друга частина інтерфейсу призначена для безпосередньої взаємодії. Саме

тут користувач обирає параметри аналізу, задає директорії, які необхідно перевірити, та ініціює виконання ключових функцій програми. Цей функціональний блок виступає своєрідним робочим простором, де здійснюються основні дії, пов’язані із запуском та керуванням процесом аналізу.

Третя частина інтерфейсу відповідає за відображення результатів – вона показує підсумкову інформацію після завершення аналізу. Зокрема, користувач може побачити детальну статистику щодо вмісту вибраної директорії: загальну кількість файлів, їх загальний розмір, кількість унікальних та дублікатів, а також інші важливі параметри, які дозволяють зробити висновки про стан файлової системи. Усі ці дані подаються у зрозумілому вигляді, що сприяє зручному сприйняттю результатів.

У цілому, графічний інтерфейс розробленого програмного додатку є логічно структурованим, зручним у користуванні та адаптованим під потреби кінцевого користувача. Його вигляд наведено на рисунку 3.7, де детально показано компонування основних елементів інтерфейсу та їх призначення.

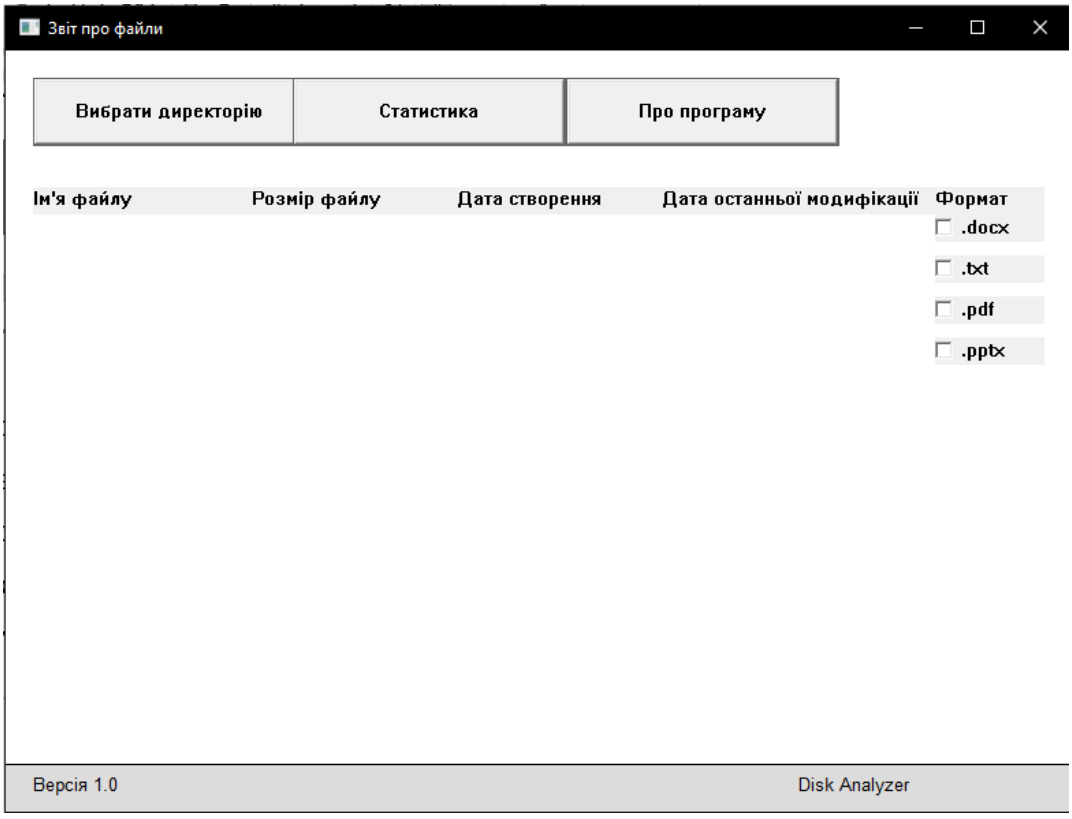


Рисунок 3.7 – Графічний інтерфейс головного вікна

Керування застосунком здійснюється через кнопки на робочій області головного вікна.

Перед початком роботи з додатком слід ознайомитися з програмою та натиснути на кнопку «Про програму» для розуміння функцій програмного додатку (див.рисунок 3.8).

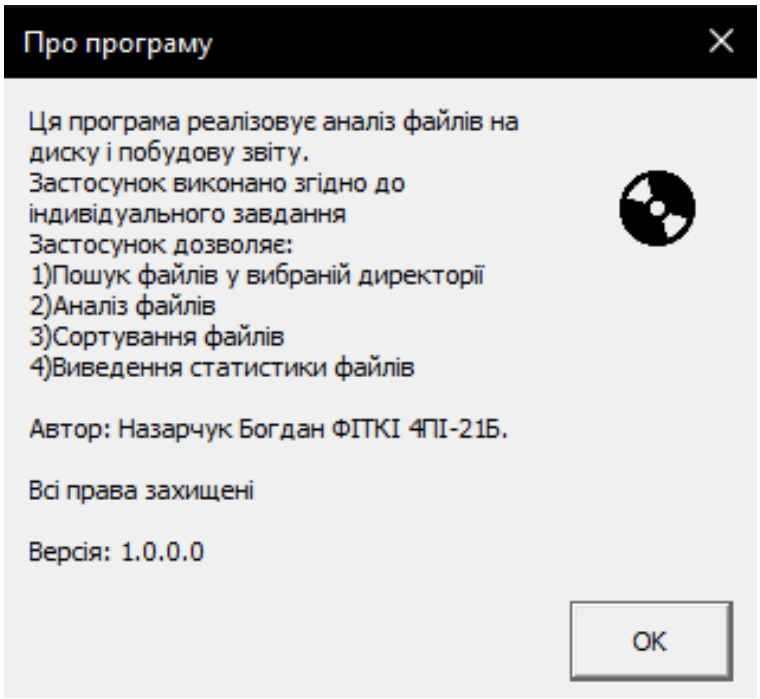


Рисунок 3.8 – Вікно «Про програму»

Вікно містить у собі основну інформацію про програму і розробника.

Переходячи до функціонального складу додатку користувачу потрібно натиснути кнопку «Вибрати директорію» та обрати потрібну папку. Після цього програма миттєво зчитує всі файли в обраній директорії (без вкладених папок) і відображає їх у табличному вигляді. Таблиця містить основну інформацію про кожен файл: назву, розмір, дату створення, дату останньої модифікації та тип (формат). Праворуч видно панель фільтрації за форматами файлів — користувач може обрати, які саме типи файлів він хоче бачити у звіті (.docx, .pdf тощо). Дані оновлюються динамічно відповідно до обраних фільтрів (див.рисунок 3.9).

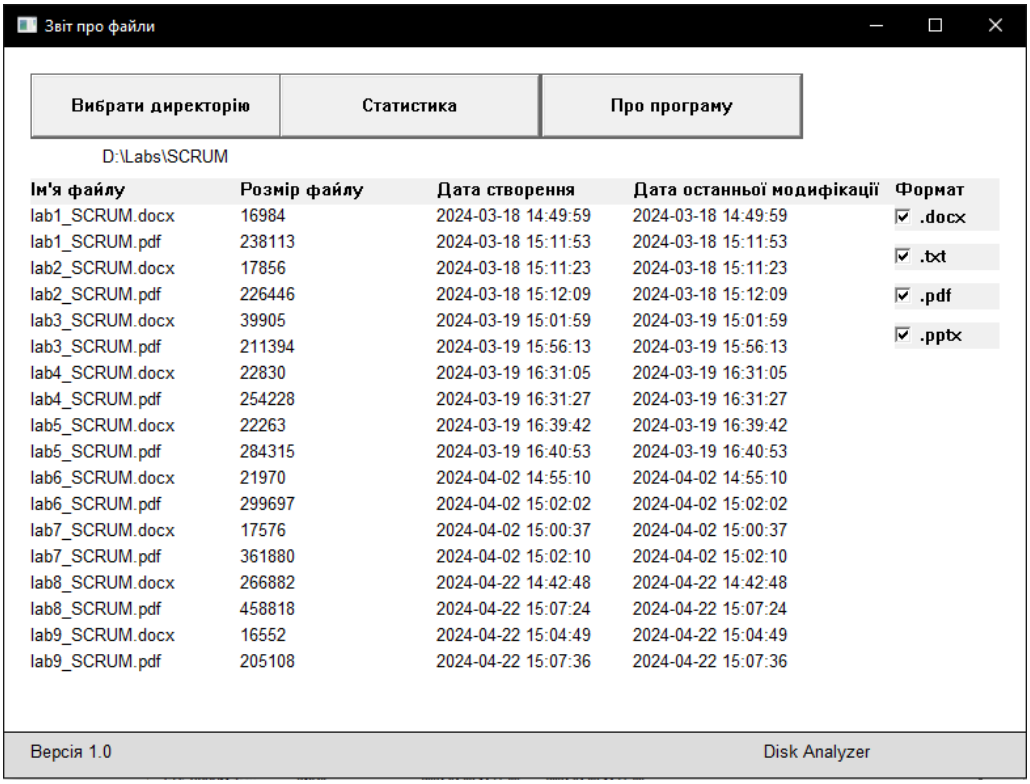


Рисунок 3.9 – Результат аналізу файлів

Розглядаючи елементи додатка окремо, можна виділити наступний складовий графічного інтерфейсу – це нижня частина, в якій вказано назву та версію застосунку. Такий підхід дозволяє користувачеві швидко ідентифікувати програму та переконатися, що він працює з актуальною версією, що особливо важливо у випадках технічної підтримки чи оновлення.

Кнопка «Статистика» виглядає таким чином: вона має простий і зрозумілий дизайн, зазвичай прямокутної форми з написом або піктограмою, що символізує збір даних. Натискання на цю кнопку запускає процес аналізу вибраної директорії, у результаті чого система збирає і обробляє інформацію про файли. Після завершення обробки на екрані з’являється підсумкова інформація — загальна кількість файлів, їхній сумарний розмір, найпоширеніші типи файлів, а також інші статистичні показники, що можуть бути корисними користувачу для оцінки стану даних на диску.

Дизайн кнопки враховує принципи зручності використання: вона має

достатній розмір для комфортного натискання, помітне розміщення на інтерфейсі та змінює вигляд під час наведення або натискання, що забезпечує зворотний зв'язок користувачу. Така реалізація дозволяє інтуїтивно зрозуміти її функціональне призначення навіть без ознайомлення з інструкцією (див.рисунок 3.10).

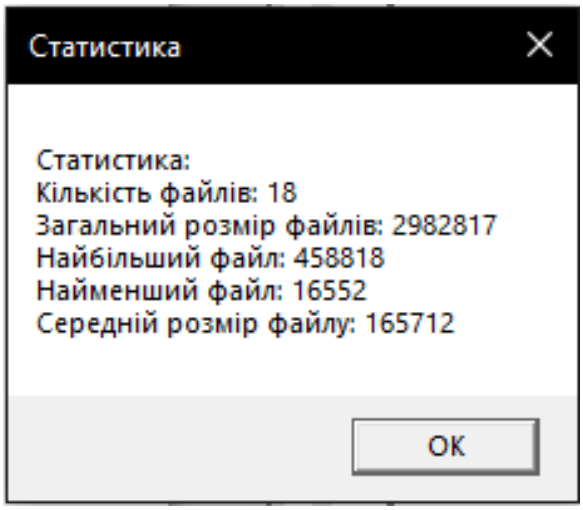


Рисунок 3.10 – Вікно додатку «Статистика»

Вікно зі статистикою містить основну інформацію про набір даних. У вікні зазначено, що кількість файлів становить 18, їхній загальний розмір – 2 982 817 байтів. Найбільший файл має розмір 458 818 байтів, а найменший – 16 552 байти. Середній розмір одного файлу складає 165 712 байтів. Ця інформація є корисною для оцінки обсягу даних, що обробляються у програмі.

Отже, користувач отримує простий і зрозумілий інтерфейс для робити з розробленим програмним додатком, що може допомогти йому в повсякденному житті.

3.4 Висновки

У ході варіантного аналізу засобів програмної реалізації було розглянуто декілька мов програмування та інструментів, з урахуванням вимог до продуктивності, гнучкості та підтримки системного доступу до файлової

структури. На основі отриманих результатів було обґрунтовано вибір мови програмування C++, яка забезпечує високу швидкість виконання, низькорівневий контроль над ресурсами та широкі можливості для розробки продуктивного застосунку.

На етапі розробки програмного додатку було створено окремі функціональні модулі, кожен з яких відповідає за виконання конкретних завдань — обробку даних, аналіз файлів, виведення результатів тощо. Такий модульний підхід підвищив масштабованість і підтримуваність системи, дозволяючи в подальшому легко вносити зміни та розширення.

Особливу увагу приділено розробці користувацького інтерфейсу. Було обґрунтовано доцільність використання графічного інтерфейсу, який забезпечує зручну взаємодію з програмою навіть для користувачів без технічної підготовки. Інтерфейс розроблений відповідно до принципів простоти, наочності та доступності, що значно підвищує загальну ергономіку програмного забезпечення.

У результаті всі розглянуті рішення спрямовані на створення ефективного, зручного та функціонального застосунку, орієнтованого на кінцевого користувача.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз методів тестування системи

Тестування програмного забезпечення є однією з найбільш важливих та відповідальних складових процесу створення програмних систем. Це не просто технічна процедура, яка виконується наприкінці проєкту, а цілісний, багатоетапний процес, що супроводжує розробку програмного забезпечення на всіх її стадіях. Його основна мета полягає в тому, щоб гарантувати якість, функціональність, стабільність, надійність, а також зручність та безпеку продукту перед його випуском у реальне середовище експлуатації. Саме завдяки тестуванню стає можливим виявлення недоліків ще до того, як ними зіштовхнеться кінцевий користувач, що, в свою чергу, дозволяє зекономити час, ресурси і зберегти репутацію розробника.

Процес тестування включає не тільки перевірку того, чи виконує програмний продукт свої функції, але й аналіз його поведінки у нештатних ситуаціях, підвищеному навантаженні, при взаємодії з іншими програмами або системами, а також у випадках введення помилкових або некоректних даних. Такий підхід дозволяє досягти максимальної відповідності продукту не лише технічним вимогам, а й очікуванням користувачів. Крім того, тестування відіграє роль своєрідного зворотного зв’язку для команди розробників: виявлені помилки або нестабільна поведінка можуть свідчити про концептуальні помилки в архітектурі або логіці програми, що своєю чергою вимагає перегляду певних рішень.

У сфері тестування існує кілька класифікацій, однак найбільш поширеним є поділ на функціональне та нефункціональне тестування [27]. Вони охоплюють різні аспекти системи, проте обидва є однаково важливими для комплексної оцінки якості програмного забезпечення. Функціональне тестування зосереджується на перевірці реалізованих функцій, які були зазначені в технічному завданні або специфікації. Головне питання, на яке воно дає відповідь – чи робить програма те, що повинна робити. Наприклад, якщо користувач

натискає певну кнопку, система повинна виконати відповідну дію. Якщо програма повинна аналізувати файли в директорії, то вона має це робити коректно: враховуючи розміри, типи файлів, імена, глибину вкладеності та інші параметри. Функціональне тестування дозволяє визначити, чи спрацьовують всі елементи інтерфейсу, чи працює логіка обробки даних, чи правильно формуються результати аналізу, і чи всі компоненти взаємодіють між собою згідно із заданими правилами. З практичної точки зору, саме цей вид тестування є першим, який реалізується при перевірці нового функціоналу, оскільки без нього неможливо перейти до глибшого аналізу роботи програми.

Натомість нефункціональне тестування звертає увагу на такі характеристики системи, як її стабільність, швидкодія, масштабованість, безпека, сумісність та зручність у користуванні. Воно дає відповіді не на питання “що робить програма”, а на питання “як добре вона це робить”. Наприклад, при обробці великої кількості файлів важливо не лише правильно вивести результат, але й зробити це у прийнятні строки. Якщо програма зависає або суттєво сповільнюється при обробці директорії з великою кількістю вкладень, це свідчить про потребу в оптимізації алгоритмів. Крім того, інтерфейс користувача повинен бути інтуїтивно зрозумілим, зручним навіть для людей, які не мають спеціальної підготовки. Наявність сповіщень про помилки, підказок, візуального зворотного зв’язку – все це також відноситься до нефункціональних аспектів і відіграє важливу роль у формуванні позитивного користувацького досвіду.

Тестування вимагає ретельно підготовленого середовища. Потрібно визначити перелік сценаріїв використання, можливі комбінації вхідних даних, граничні умови та нештатні ситуації. Наприклад, тестування має включати перевірку того, як система поводить себе при аналізі порожньої директорії, або при спробі обробити каталог, який містить десятки тисяч елементів, або навіть при доступі до системних файлів, які заблоковані правами адміністратора. Варто також звернути увагу на перевірку реакції програми на дії, які можуть призвести до помилок, наприклад, відключення носія під час сканування, втрату доступу до

директорії, або обробку пошкоджених файлів. Саме такі тестові сценарії дозволяють виявити слабкі місця системи та підвищити її надійність.

Під час тестування програмного забезпечення, створеного в межах бакалаврської кваліфікаційної роботи, основну увагу було зосереджено саме на функціональному тестуванні. Оскільки в рамках проєкту реалізовувався інструмент для аналізу структури та змісту файлової системи, найважливішим завданням було впевнитися в тому, що основні функції – сканування директорій, сортування файлів, пошук дублікатів, формування статистики – працюють стабільно, коректно та у відповідності до поставлених вимог. Для цього застосовувався поетапний підхід: кожна функція перевірялась окремо, а також у контексті повної взаємодії з іншими компонентами. Було створено тестове середовище з кількома директоріями, які включали файли з різними іменами, типами, розширеннями, дублікати та елементи, що мали нетипову поведінку (наприклад, нульовий розмір або відсутність розширення).

Окрім функціональної перевірки, проводилась також часткова оцінка нефункціональних характеристик. Наприклад, вимірювався час, необхідний для сканування директорії певного обсягу, оцінювалась реакція інтерфейсу на дії користувача, перевірялась стабільність програми при багаторазовому повторенні операцій або при тривалому запуску без перезавантаження. Особлива увага приділялася виведенню статистики, адже саме ця частина є ключовою для користувача – вона повинна бути зрозумілою, візуально структурованою і при цьому не перевантажувати інтерфейс.

Під час тестування також було виявлено ряд незначних недоліків, зокрема пов’язаних із відображенням неправильно закодованих назв файлів або з обробкою файлів без доступу. Ці недоліки були проаналізовані та усунуті шляхом впровадження додаткових перевірок та обробки виключень. Крім того, інтерфейс було адаптовано для відображення повідомлень про помилки, що значно полегшило взаємодію з програмою в умовах, коли користувач не знає причин збою. Впровадження цих змін покращило загальне враження від користування

додатком, а також засвідчило важливість тестування не лише як перевірки правильності коду, а й як інструменту покращення користувацького досвіду.

Підсумовуючи вищезазначене, варто наголосити, що тестування програмного забезпечення є критично необхідним етапом створення будь-якого цифрового продукту. Воно дозволяє не лише виявити помилки, а й підвищити якість продукту, забезпечити його відповідність очікуванням користувачів, а також підготувати програму до масштабного використання. Без ґрунтовного тестування неможливо гарантувати, що програмне забезпечення буде працювати стабільно, ефективно і безпечно. У рамках даної бакалаврської кваліфікаційної роботи тестування дозволило не лише підтвердити правильність реалізації основних функцій програми, а й виявити можливі напрями подальшого вдосконалення системи, що є свідченням ефективності обраного підходу до розробки та якості виконаного проєкту загалом.

4.2 Тестування системи

Для підтвердження результатів роботи програмного застосунку «Disk Analyzer» очікуваним було здійснення тестування основних функцій, доступних користувачу. Тестування є ключовим етапом у процесі розробки програмного забезпечення, оскільки дозволяє виявити помилки, недоліки у функціональності або взаємодії з користувачем ще до моменту впровадження продукту в експлуатацію.

Під час тестування перевірялась коректність роботи елементів графічного інтерфейсу, відповідність поведінки кнопок очікуваним діям, стабільність функцій фільтрації, сортування, та виведення статистичної інформації. Особлива увага приділялась реакції системи на типові дії користувача та її здатності адекватно опрацьовувати різні сценарії взаємодії, зокрема ситуації, коли не було вибрано директорію для аналізу.

Тестування проводилось згідно з заздалегідь підготовленими сценаріями, які охоплювали як базову, так і розширену функціональність додатку. Кожен тестовий випадок мав свій унікальний ідентифікатор, опис дії, методику

проведення та очікуваний результат. Результати тестування дозволяють зробити висновок про стабільність і готовність програмного забезпечення до подальшого використання.

Результати тестування наведено у таблиці 4.1.

Таблиця Б.1 – Тестування основних функцій програмного застосунку

ID	Назва	Методика проведення тестування	Очікуваний результат	Результат
ТВ-1	Перевірка відображення елементів головного вікна	1. Відкрити файл «DiskAnalyzer.exe»	1. Відображається логотип та назва застосунку. 2. Відображається робоча область. 3. Відображається меню і список логів.	Виконано
ТВ-2	Перевірка коректного відкриття вікна «Про програму»	1. Відкрити файл «DiskAnalyzer.exe» 2. Натиснути на кнопку «Про програму» в меню.	1. Відкривається вікно «Про програму». 2. Відкрите вікно відображається поверх головного вікна. 3. Кнопка «Закрити» здійснює вихід з вікна.	Виконано
ТВ-3	Перевірка роботи кнопки «Вихід»	1. Відкрити файл «DiskAnalyzer.exe» 2. Натиснути на кнопку «Вихід»	1. Програма успішно закриється.	Виконано
ТВ-4	Перевірка коректного відкриття вікна «Статистика»	1. Відкрити файл «DiskAnalyzer.exe» 2. Натиснути на кнопку «Статистика».	1. Програма видає помилку, що не вибрана директорія. 2. Програма успішно відкриває вікно Статистика. 3. Кнопка «Закрити» здійснює вихід з вікна.	Виконано
ТВ-5	Сортування за різними категоріями	1. Відкрити файл «DiskAnalyzer.exe» 2. Натиснути на одну з кнопок в панелі.	1. Здійснюється сортування за вибором категорій сортування	Виконано
ТВ-6	Вибірковий формат файлів для виведення з директорії	1. Відкрити файл «DiskAnalyzer.exe» 2. Натиснути на вибірковий чек-бокс	1. При виборі директорії, програма виводить тільки ті файли, які були вибрані в чек-боксі	Виконано

У ході тестування програмного забезпечення DiskAnalyzer також було перевірено коректність відображення сповіщень про помилки, які виникають у випадку неправильної або неповної взаємодії користувача з інтерфейсом застосунку.

Зокрема, реалізовано два типи інформативних вспливаючих вікон, які з'являються в критичних ситуаціях:

Помилка 1: "Не вдалося знайти шлях до директорії" (див.рисунок 4.1)

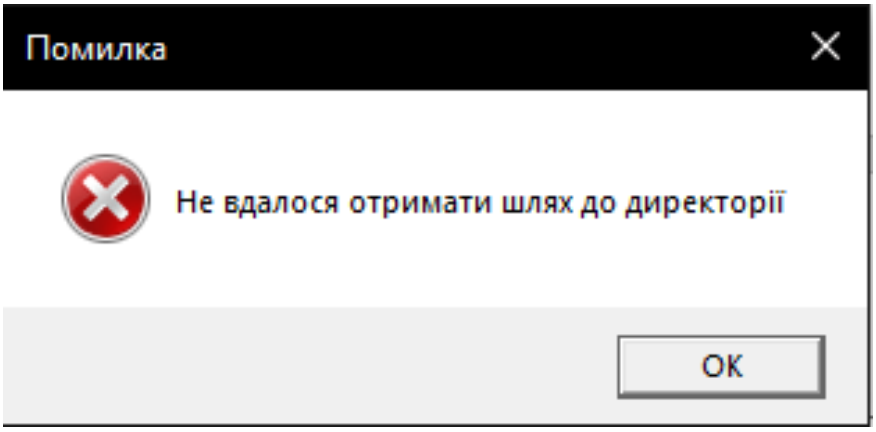


Рисунок 4.1 – Вікно «Помилка»

Виникає у випадку, коли користувач намагається виконати операції, пов’язані з аналізом, не обравши папку. Це дозволяє уникнути фатальних помилок під час доступу до файлової системи.

Помилка 2: "Статистика недоступна, оскільки файли ще не завантажені" (див.рисунок 4.2)

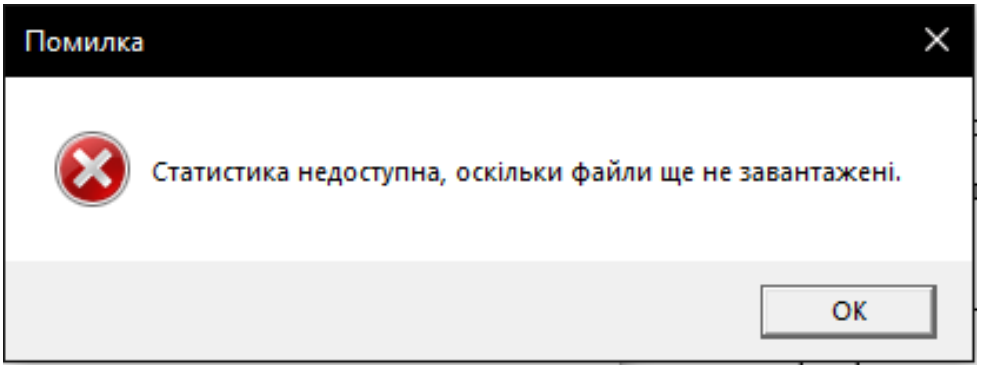


Рисунок 4.2 – Вікно «Помилка»

З’являється при натисканні кнопки «Статистика» без попереднього завантаження файлів. Повідомлення блокує подальший доступ до вікна зі статистичними даними, оскільки такі дані відсутні.

Ці повідомлення мають інформативний характер, зрозуміло формулюють причину помилки та дають змогу користувачу швидко зорієнтуватися щодо необхідних дій. Відображення таких вікон перевірено під час виконання тестових сценаріїв ТВ-4 та додаткових ручних тестів на виняткові ситуації.

Отже це підтверджує відповідність функціональності заявленим вимогам до зручності та надійності інтерфейсу.

4.3 Висновки

Отже, у результаті виконання бакалаврської кваліфікаційної роботи було проведене тестування програмного застосунку «Disk Analyzer, яке підтвердило працездатність основних функцій, передбачених технічним завданням. Усі ключові елементи графічного інтерфейсу – вікна, кнопки, меню – функціонують відповідно до очікуваного сценарію. Особливу увагу було приділено перевірці коректного відображення вікон («Про програму», «Статистика»), роботи кнопки «Вихід», а також функціоналу сортування та фільтрації даних за заданими критеріями.

Усі тестові сценарії було виконано успішно, що засвідчує стабільність програмного продукту під час типових користувацьких дій. Зафіксовані результати демонструють, що додаток не лише виконує передбачені функції, а й надає інтуїтивно зрозумілий інтерфейс для взаємодії з користувачем. Це свідчить про достатній рівень зручності та якості реалізації інтерфейсу.

Таким чином, результати тестування підтверджують, що застосунок «Disk Analyzer» є готовим до використання та здатен ефективно виконувати завдання з аналізу файлової структури директорій, формування статистичних звітів і покращення організації інформації на комп’ютері користувача.

ВИСНОВКИ

У бакалаврської кваліфікаційній роботі було всебічно досліджено, спроєктовано та реалізовано програмний застосунок для аналізу файлів на диску з подальшим формуванням звітності. Реалізація бакалаврської кваліфікаційної роботи відповідає методичним вказівкам [28].

У межах дослідження проведено детальний аналіз існуючих програмних рішень у цій сфері, зокрема засобів класифікації, візуалізації та роботи з великими обсягами файлової інформації. Було виявлено основні переваги й недоліки аналогів, що дозволило сформувати актуальні вимоги до власної розробки.

У першому розділі визначено проблему накопичення та неструктурованості файлів, проаналізовано сучасні програмні аналоги та сформульовано задачу дослідження. Було обґрунтовано необхідність створення нового інструменту, здатного ефективно обробляти файлову структуру директорій, фільтрувати й візуалізувати дані за заданими параметрами.

У другому розділі розроблено метод детермінованого аналізу файлів, побудовано структурні схеми системи та запропоновано алгоритми її роботи. Методика забезпечує передбачувану та надійну обробку даних, що стало основою для реалізації функціоналу системи.

У третьому розділі виконано варіантний аналіз та обґрунтовано вибір мови програмування та засобів реалізації (обрано C++), що відповідають вимогам до швидкодії та кросплатформеності. Реалізовано основні модулі системи, включаючи модуль збору інформації про файли, модуль фільтрації та статистичної обробки. Також було приділено увагу інтерфейсу користувача, який спроєктовано відповідно до принципів зручності та інтуїтивності.

У четвертому розділі проведено тестування програмного забезпечення за основними функціональними сценаріями, а також перевірку обробки виняткових ситуацій. Отримані результати засвідчили стійкість та функціональну повноту розробленого застосунку. Програма коректно реагує на дії користувача, надає

сповіщення про помилки, виконує сортування та генерацію статистики.

Таким чином, у результаті виконаної роботи: реалізовано ефективний інструмент аналізу файлової структури директорій, забезпечено інформативне виведення статистичних даних, забезпечено стабільну роботу програмного забезпечення в стандартних та виняткових ситуаціях, виконано повний цикл від постановки задачі до тестування та аналізу результатів.

Розроблений застосунок може бути використаний як окремий інструмент для оптимізації роботи з файлами на цифрових носіях або як складова більших систем адміністрування та аудиту даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Назарчук Б.Ю., Черноволик Г.О. Використання програмного забезпечення системи аналізу файлів на диску/ Матеріали Молодь в науці: дослідження, проблеми, перспективи Вінницького національного технічного університету (МН ВНТУ–2025) : збірник доповідей [Електронний ресурс] — Режим доступу до ресурсу https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25650

2. Дослідження методів збору та обробки даних в росподілених інформаційних системах [Електронний ресурс] – Режим доступу до ресурсу https://science.lpnu.ua/sites/default/files/journal-paper/2021/dec/25668/stattya3czhuravelsdumichoshpur.pdf?

3. Рефлексія в програмуванні: просте і зрозуміле пояснення [Електронний ресурс] – https://foxminded.ua/refleksiia-v-prohramuvanni/

4. Виявлення конфіденційних даних [Електронний ресурс] – Режим доступу до ресурсу https://www.solix.com/uk/kb/sensitive-data-discovery/

5. Політика про обробку та захист персональних даних [Електронний ресурс] – Режим доступу до ресурсу https://servier.ua/polityka-pro-obrobku-ta-zakhyst-personalnykh-danykh/

6. Огляд програмних засобів статистичного аналізу даних. [Електронний ресурс] – Режим доступу до ресурсу http://www.Economy.nayka.com.ua/?o p=1&z=5676.

7. WinDirStat [Електронний ресурс] – Режим доступу до ресурсу https://windirstat.net/

8. WizTree [Електронний ресурс] – Режим доступу до ресурсу https://diskanalyzer.com .

9. Disk Inventory X [Електронний ресурс] – Режим доступу до ресурсу https://www.derlien.com/ .

10. TreeSize Free [Електронний ресурс] – Режим доступу до ресурсу https://www.jam-software.com/treesize_free.

11. Рекурсія в програмуванні та як її застосовувати [Електронний ресурс] –

Режим доступу до ресурсу https://foxminded.ua/rekursiia-v-prohramuvanni/.

12. Порівняльний метод [Електронний ресурс] – Режим доступу до ресурсу https://metodologiasapiens.com/uk/metodos/metodo-comparativo/Візуалізація [Електронний ресурс] – Режим доступу до ресурсу https://socialdata.org.ua/manual/manual5/

13. Візуалізація [Електронний ресурс] – Режим доступу до ресурсу https://socialdata.org.ua/manual/manual5/

14. Формати даних [Електронний ресурс] – Режим доступу до ресурсу https://socialdata.org.ua/manual/manual2/

15. Iglberger Klaus. C++ Software Design. Bavaria, O'Reilly Media, 1st edition, 2022. 435p.

16. Eckel B. Thinking in Java. – 4th ed. – Prentice Hall, 2006. – 1150 p.

17. Albahari J., Albahari B. C# 10 in a Nutshell: The Definitive Reference. – O’Reilly Media, 2022. – 1080 p.

18. Microsoft. Visual Studio Documentation. [Електронний ресурс] – Режим доступу до ресурсу https://docs.microsoft.com/en-us/visualstudio/ .

19. Qt Company. Qt Creator Manual. [Електронний ресурс] – Режим доступу до ресурсу https://doc.qt.io/qtcreator/ .

20. Code::Blocks Team. Code::Blocks User Manual. [Електронний ресурс] – Режим доступу до ресурсу http://wiki.codeblocks.org/index.php/Main_Page .

21. Create a report using an aggregation [Електронний ресурс] – Режим доступу до ресурсу https://www.servicenow.com/docs/ bundle /yokohama – security - management /page/product/vulnerability-response/task/generate-report.html

22. How to sort files by part of the filename? [Електронний ресурс] – Режим доступу до ресурсу https://unix.stackexchange.com/questions/248350/how-to-sort-files-by-part-of-the-filename

23. Calculating function on files from directory [Електронний ресурс] – Режим доступу до ресурсу https://stackoverflow.com/questions/44630323/calculating-function-on-files-from-directory

24. Dialog box functions [Електронний ресурс] – Режим доступу до ресурсу

https://doc.windev.com/en-US/?3021014

25. How to set footer to be on the bottom of the screen [Електронний ресурс] – Режим доступу до ресурсу https://stackoverflow.com/questions/35020698/how-to-set-footer-to-be-on-the-bottom-of-the-screen

26. What is GUI? Graphical User Interfaces, Explained: вебсайт. [Електронний ресурс] – Режим доступу до ресурсу https://blog.hubspot.com/website/what-is-gui# .

27. Види тестування. [Електронний ресурс] – Режим доступу до ресурсу https://dan-it.com.ua/uk/blog/vidy-funkcionalnogo-i nefunkcionalnogo-testirovanija/ .

28. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О.Н. Романюк, Г.О. Черноволик – Вінниця: ВНТУ, 2022. – 52с.

ДОДАТКИ

57

Додаток А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«25» 03 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка програмного забезпечення системи аналізу файлів на диску і побудови звіту»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц., каф. ПЗ Черноволик Г.О.
«24» 03 2025 року.

Виконав:
студент гр. 4ПП-21б Назарчук Б.Ю.
«24» 03 2025 року.

м. Вінниця – 2025 рік

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка програмного забезпечення системи аналізу файлів на диску і побудови звіту».

Галузь застосування – програмне забезпечення.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол №18 від 10 лютого 2025 р.

3. Мета та призначення розробки.

Метою роботи є створення засобу для автоматизованого аналізу файлів на диску. Призначення — виявлення, класифікація файлів та формування звітів для оптимізації зберігання даних.

4. Вихідні дані для проведення НДР

Джерелом інформації є технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

5. Технічні вимоги

Вхідні дані — шлях до файлів на дискові з комп’ютера.

Вихідні дані — аналіз файлів на диску і побудований звіт.

6. Конструктивні вимоги.

Інтерфейс програми повинен відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

–пояснювальна записка до бакалаврської кваліфікаційної роботи;

–технічне завдання;

–лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз розвитку програмного застосунку для аналізу файлів та побудови звіту	25.03.25 – 03.04.25
2	Розробка моделі та алгоритм програмного застосунку	04.04.25 – 15.04.25
3	Варіантний аналіз та обгрунтування вибору засобів програмної реалізації	16.04.25 – 17.04.25
4	Розробка програмних модулів та методів реалізації програмного додатку для аналізу файлів	18.04.25 – 27.04.25
5	Тестування програмного забезпечення	28.04.25 – 11.05.25
6	Оформлення матеріалів до захисту БКР	12.05.25 – 30.05.25

10. Порядок контролю та прийняття.

Усі етапи бакалаврської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

60

Додаток Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Розробка програмного забезпечення системи аналізу файлів на диску і побудови звіту»

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна роботя)

Підрозділ: кафедра програмного забезпечення, факультет ІТКІ, 4ПІ-21Б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 7.2%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту.
- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ к.т.н., доц., каф. ПЗ Черноволик Г.О.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Назарчук Б. Ю.
(підпис) (прізвище, ініціали)

Додаток В

Лістинг програми

```
#include <Windows.h>
#include <iostream>
#include <iomanip>
#include <filesystem>
#include <vector>
#include <algorithm>
#include <chrono>
#include <ctime>
#include <shlobj.h> // Додано для використання діалогового вікна
#include "Resource.h"

using namespace std;
namespace fs = std::filesystem;

struct FileData {
    wstring name;
    long long size = 0;
    time_t creationTime = 0;
    time_t lastModifiedTime = 0;
};

vector<FileData> files; // Глобальна змінна, доступна у всіх функціях
wstring directoryPath; // Змінна для зберігання шляху обраної директорії


wstring GetDirectoryPathFromUser(HWND hwnd) {
    wchar_t szDir[MAX_PATH];
    BROWSEINFO bInfo;
    ZeroMemory(&bInfo, sizeof(bInfo));
    bInfo.hwndOwner = hwnd;
    bInfo.pidlRoot = NULL;
    bInfo.pszDisplayName = szDir;
    bInfo.lpszTitle = L"Виберіть директорію";
    bInfo.ulFlags = BIF_RETURNONLYFSDIRS;

    LPITEMIDLIST pidl = SHBrowseForFolder(&bInfo);
    if (pidl != NULL) {
        if (SHGetPathFromIDList(pidl, szDir)) {
            CoTaskMemFree(pidl);
            directoryPath = szDir;
            return szDir;
        }
    }
    return L"";
}

bool CompareName(const FileData& a, const FileData& b) {
    return a.name < b.name;
}

bool CompareSize(const FileData& a, const FileData& b) {
    return a.size < b.size;
}
```

```
bool CompareCreationTime(const FileData& a, const FileData& b) {
    return a.creationTime < b.creationTime;
}

bool CompareLastModifiedTime(const FileData& a, const FileData& b) {
    return a.lastModifiedTime < b.lastModifiedTime;
}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

class Report {
public:
    void generateReport(HWND hwnd) {
        // Отримуємо контекст виведення для роботи з вікном
        HDC hdc = GetDC(hwnd);
        if (hdc == NULL) {
            MessageBox(hwnd, L"Помилка отримання контексту виведення", L"Помилка", MB_OK | MB_ICONERROR);
            return;
        }

        // Налаштування шрифту для виведення тексту
        HFONT hFont = CreateFont(16, 0, 0, 0, FW_NORMAL, FALSE, FALSE, FALSE, ANSI_CHARSET, OUT_DEFAULT_PRECIS,
CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH | FF_DONTCARE, L"Arial");
        SelectObject(hdc, hFont);

        int headerY = 100; // Початкова у-координата заголовка
        int statisticY = headerY + 20; // Початкова у-координата статистики

        // Виведення заголовків+
        static bool headerCreated = false;
        if (!headerCreated) {
            CreateWindow(L"STATIC", L"Ім'я файлу", WS_VISIBLE | WS_CHILD, 20, headerY, 180, 20, hwnd, (HMENU)1, NULL, NULL);
            CreateWindow(L"STATIC", L"Розмір файлу", WS_VISIBLE | WS_CHILD, 180, headerY, 150, 20, hwnd, (HMENU)2, NULL, NULL);
            CreateWindow(L"STATIC", L"Дата створення", WS_VISIBLE | WS_CHILD, 330, headerY, 150, 20, hwnd, (HMENU)3, NULL, NULL);
            CreateWindow(L"STATIC", L"Дата останньої модифікації", WS_VISIBLE | WS_CHILD, 480, headerY, 200, 20, hwnd, (HMENU)4, NULL, NULL);
            CreateWindow(L"STATIC", L"Формат", WS_VISIBLE | WS_CHILD, 680, headerY, 80, 20, hwnd, (HMENU)5, NULL, NULL);
            headerCreated = true;
        }

        // Додамо виведення шляху обраної директорії
        TextOut(hdc, 75, 75, directoryPath.c_str(), directoryPath.length());

        // Виведення даних про файли
        for (const auto& file : files) {
            // Виведення даних на екран
            TextOut(hdc, 20, statisticY, file.name.c_str(), file.name.length());
            wstring wsize = to_wstring(file.size);
            TextOut(hdc, 180, statisticY, wsize.c_str(), wsize.length());
            wchar_t creationTimeString[26];
            struct tm creationTimeStruct;
            localtime_s(&creationTimeStruct, &file.creationTime);
            wcsftime(creationTimeString, sizeof(creationTimeString), L"%Y-%m-%d %H:%M:%S", &creationTimeStruct);
            TextOut(hdc, 330, statisticY, creationTimeString, wcslen(creationTimeString));
            wchar_t lastModifiedTimeString[26];
            struct tm lastModifiedTimeStruct;
            localtime_s(&lastModifiedTimeStruct, &file.lastModifiedTime);
            wcsftime(lastModifiedTimeString, sizeof(lastModifiedTimeString), L"%Y-%m-%d %H:%M:%S", &lastModifiedTimeStruct);
            TextOut(hdc, 480, statisticY, lastModifiedTimeString, wcslen(lastModifiedTimeString));
```

```
        statisticY += 20; // Збільшення у-координати для наступного запису
    }

    // Вивільнення контексту виведення та шрифту
    ReleaseDC(hwnd, hdc);
    DeleteObject(hFont);
}
void SortFiles(int criteria, bool reverse) {
    switch (criteria) {
    case 1:
        if (reverse) {
            sort(files.begin(), files.end(), [](const FileData& a, const FileData& b) { return a.name > b.name; });
        }
        else {
            sort(files.begin(), files.end(), [](const FileData& a, const FileData& b) { return a.name < b.name; });
        }
        break;
    case 2:
        if (reverse) {
            sort(files.begin(), files.end(), [](const FileData& a, const FileData& b) { return a.size > b.size; });
        }
        else {
            sort(files.begin(), files.end(), [](const FileData& a, const FileData& b) { return a.size < b.size; });
        }
        break;
    case 3:
        if (reverse) {
            sort(files.begin(), files.end(), [](const FileData& a, const FileData& b) { return a.creationTime > b.creationTime; });
        }
        else {
            sort(files.begin(), files.end(), [](const FileData& a, const FileData& b) { return a.creationTime < b.creationTime; });
        }
        break;
    case 4:
        if (reverse) {
            sort(files.begin(), files.end(), [](const FileData& a, const FileData& b) { return a.lastModifiedTime > b.lastModifiedTime; });
        }
        else {
            sort(files.begin(), files.end(), [](const FileData& a, const FileData& b) { return a.lastModifiedTime < b.lastModifiedTime; });
        }
        break;
    }
}
// Додаємо функцію для виведення статистики
void showStatistics(HWND hwnd) {
    if (files.empty()) {
        MessageBox(hwnd, L"Статистика недоступна, оскільки файли ще не завантажені.", L"Помилка", MB_OK | MB_ICONERROR);
        return;
    }

    long long totalSize = 0;
    long long maxSize = 0;
    long long minSize = LLONG_MAX;

    for (const auto& file : files) {
        totalSize += file.size;
        maxSize = max(maxSize, file.size);
        minSize = min(minSize, file.size);
```

```
        }

        long long averageSize = files.empty() ? 0 : totalSize / files.size();

        wstring stats = L"Статистика:\nКількість файлів: " + to_wstring(files.size()) +
            L"\nЗагальний розмір файлів: " + to_wstring(totalSize) +
            L"\nНайбільший файл: " + to_wstring(maxSize) +
            L"\nНайменший файл: " + to_wstring(minSize) +
            L"\nСередній розмір файлу: " + to_wstring(averageSize);

        MessageBox(hwnd, stats.c_str(), L"Статистика", MB_OK);
    }
    void drawFooter(HWND hwnd) {
        HDC hdc = GetDC(hwnd);
        if (hdc == NULL) {
            MessageBox(hwnd, L"Помилка отримання контексту виведення", L"Помилка", MB_OK | MB_ICONERROR);
            return;
        }

        HFONT hFont = CreateFont(16, 0, 0, 0, FW_NORMAL, FALSE, FALSE, FALSE, ANSI_CHARSET, OUT_DEFAULT_PRECIS,
CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH | FF_DONTCARE, L"Arial");
        SelectObject(hdc, hFont);

        RECT rc;
        GetClientRect(hwnd, &rc);

        int footerHeight = 30;
        int footerY = rc.bottom - footerHeight - 5;

        // Зафарбовуємо фон footer'а
        HBRUSH hBrush = CreateSolidBrush(RGB(220, 220, 220)); // Сірий колір
        RECT footerRect = { 0, footerY, rc.right, rc.bottom };
        FillRect(hdc, &footerRect, hBrush);
        DeleteObject(hBrush);

        // Малюємо рамочку
        HPEN hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 0)); // Чорна рамка
        SelectObject(hdc, hPen);
        MoveToEx(hdc, 0, footerY, NULL);
        LineTo(hdc, rc.right, footerY);
        DeleteObject(hPen);

        int textLeftMargin = 20;
        int textRightMargin = 200;
        int textHeight = 20;
        int textY = footerY + (footerHeight - textHeight) / 2;
        SetBkMode(hdc, TRANSPARENT);
        SetTextColor(hdc, RGB(0, 0, 0));

        TextOut(hdc, 20, footerY + 7, L"Версія 1.0", wcslen(L"Версія 1.0"));

        // Виведення назви програми
        TextOut(hdc, rc.right - 200, footerY + 7, L"Disk Analyzer", wcslen(L"Disk Analyzer"));

        ReleaseDC(hwnd, hdc);
        DeleteObject(hFont);
    }
};
class About {
```

```
public:
    void ShowAboutDialog(HWND hwnd) {
        DialogBoxParam(GetModuleHandle(NULL), MAKEINTRESOURCE(IDD_ABOUTBOX), hwnd, AboutDlgProc, 0);
    }

private:
    static INT_PTR CALLBACK AboutDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {
        switch (message) {
        case WM_INITDIALOG:
            SetDlgItemText(hDlg, IDC_ABOUTTEXT, L"Ця програма реалізовує аналіз файлів на диску і побудову звіту.\nЗастосунок виконано згідно до індивідуального завдання\nЗастосунок дозволяє:\n1)Пошук файлів у вибраній директорії\n2)Аналіз файлів\n3)Сортування файлів\n4)Виведення статистики файлів\n\nАвтор: Назарчук Богдан ФІТКІ 4ПІ-21Б.\n\nВсі права захищені\n\nВерсія: 1.0.0.0");
            return (INT_PTR)TRUE;
        case WM_COMMAND:
            if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL) {
                EndDialog(hDlg, LOWORD(wParam));
                return (INT_PTR)TRUE;
            }
            break;
        case WM_DROPFILES: {
            HDROP hDrop = (HDROP)wParam;
            WCHAR szFileName[MAX_PATH];
            DragQueryFile(hDrop, 0, szFileName, MAX_PATH);
            DragFinish(hDrop);
                    break;
        }
        }
        return (INT_PTR)FALSE;
    }
};

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {
    // Створення вікна
    HWND hwnd;
    WNDCLASSEX wc;

    wc.cbSize = sizeof(WNDCLASSEX);
    wc.style = 0;
    wc.lpfnWndProc = WindowProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wc.lpszMenuName = NULL;
    wc.lpszClassName = L"Report";
    wc.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

    if (!RegisterClassEx(&wc)) {
        MessageBox(NULL, L"Помилка реєстрації класу вікна", L"Помилка", MB_OK | MB_ICONERROR);
        return 0;
    }

    hwnd = CreateWindowEx(WS_EX_CLIENTEDGE, L"Report", L"Звіт про файли", WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, 800, 600, NULL, NULL, hInstance, NULL);

    if (hwnd == NULL) {
        MessageBox(NULL, L"Помилка створення вікна", L"Помилка", MB_OK | MB_ICONERROR);
```

```
        return 0;
    }

    // Створення кнопки для вибору директорії
    HWND button = CreateWindow(L"BUTTON", L"Вибрати директорію", WS_TABSTOP | WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON, 20, 20,
200, 50, hwnd, (HMENU)1001, hInstance, NULL);
    if (button == NULL) {
        MessageBox(hwnd, L"Помилка створення кнопки", L"Помилка", MB_OK | MB_ICONERROR);
        return 0;
    }
    HWND statButton = CreateWindow(L"BUTTON", L"Статистика", WS_TABSTOP | WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON, 210, 20, 200,
50, hwnd, (HMENU)1006, hInstance, NULL);
    if (statButton == NULL) {
        MessageBox(hwnd, L"Помилка створення кнопки", L"Помилка", MB_OK | MB_ICONERROR);
        return 0;
    }
    HWND aboutButton = CreateWindow(L"BUTTON", L"Про програму", WS_TABSTOP | WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON, 410, 20,
200, 50, hwnd, (HMENU)1007, hInstance, NULL);
    if (aboutButton == NULL) {
        MessageBox(hwnd, L"Помилка створення кнопки", L"Помилка", MB_OK | MB_ICONERROR);
        return 0;
    }

    // Створення чекбоксів для вибору типів файлів
    HWND docxCheckbox = CreateWindow(L"BUTTON", L".docx", WS_VISIBLE | WS_CHILD | BS_AUTOCHECKBOX, 680, 120, 80, 20, hwnd,
(HMENU)1002, hInstance, NULL);
    HWND txtCheckbox = CreateWindow(L"BUTTON", L".txt", WS_VISIBLE | WS_CHILD | BS_AUTOCHECKBOX, 680, 150, 80, 20, hwnd,
(HMENU)1003, hInstance, NULL);
    HWND pdfCheckbox = CreateWindow(L"BUTTON", L".pdf", WS_VISIBLE | WS_CHILD | BS_AUTOCHECKBOX, 680, 180, 80, 20, hwnd,
(HMENU)1004, hInstance, NULL);
    HWND pptxCheckbox = CreateWindow(L"BUTTON", L".pptx", WS_VISIBLE | WS_CHILD | BS_AUTOCHECKBOX, 680, 210, 80, 20, hwnd,
(HMENU)1005, hInstance, NULL);

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return static_cast<int>(msg.wParam);
}


LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {
    static int sortCriteria = 0; // Поточний критерій сортування
    static bool reverseSort = false; // Показує, чи потрібно сортувати у зворотньому порядку
    static bool showDocxFiles = false; // Показує, чи потрібно показувати .docx файли
    static bool showTxtFiles = false; // Показує, чи потрібно показувати .txt файли
    static bool showPdfFiles = false; // Показує, чи потрібно показувати .pdf файли
    static bool showPptxFiles = false;
    About abou;
    Report rep;

    switch (uMsg) {
    case WM_COMMAND:
        if (HIWORD(wParam) == BN_CLICKED) { // Перевіряємо, чи клікнуто на кнопку або чекбокс
```

```
                if (LOWORD(wParam) == 1001) { // Перевіряємо, чи клікнуто на кнопку "Вибрати директорію"
                    wstring directoryPath = GetDirectoryPathFromUser(hwnd);
                    if (directoryPath.empty()) {
                        MessageBox(hwnd, L"Не вдалося отримати шлях до директорії", L"Помилка", MB_OK | MB_ICONERROR);
                        return 0;
                    }

                    if (!fs::exists(directoryPath) || !fs::is_directory(directoryPath)) {
                        MessageBox(hwnd, L"Директорія не існує або не є директорією.", L"Помилка", MB_OK | MB_ICONERROR);
                        return 0;
                    }

                    files.clear(); // Очищаємо вектор файлів перед генерацією нового звіту

                    for (const auto& entry : fs::directory_iterator(directoryPath)) {
                        if (fs::is_regular_file(entry)) {
                            if (!showDocxFiles && entry.path().extension() == L".docx")
                                continue;
                            if (!showTxtFiles && entry.path().extension() == L".txt")
                                continue;
                            if (!showPdfFiles && entry.path().extension() == L".pdf")
                                continue;
                            if (!showPptxFiles && entry.path().extension() == L".pptx")
                                continue;

                            FileData fileData;
                            fileData.name = entry.path().filename().wstring();
                            fileData.size = fs::file_size(entry.path());
                            auto lastWriteTime = fs::last_write_time(entry.path());

                            auto time = chrono::time_point_cast<chrono::system_clock::duration>(lastWriteTime - fs::file_time_type::clock::now() + chrono::system_clock::now());
                            fileData.creationTime = chrono::system_clock::to_time_t(time);
                            fileData.lastModifiedTime = chrono::system_clock::to_time_t(time);

                            files.push_back(fileData);
                        }
                    }

                    // Перемальовуємо вікно для відображення нового звіту
                    RedrawWindow(hwnd, NULL, NULL, RDW_ERASE | RDW_INVALIDATE | RDW_FRAME | RDW_ALLCHILDREN);
                }
                else if (LOWORD(wParam) == 1002) { // Перевіряємо, чи клікнуто на чекбокс ".docx"
                    showDocxFiles = !showDocxFiles;
                }
                else if (LOWORD(wParam) == 1003) { // Перевіряємо, чи клікнуто на чекбокс ".txt"
                    showTxtFiles = !showTxtFiles;
                }
                else if (LOWORD(wParam) == 1004) { // Перевіряємо, чи клікнуто на чекбокс ".pdf"
                    showPdfFiles = !showPdfFiles;
                }
                else if (LOWORD(wParam) == 1005) { // Перевіряємо, чи клікнуто на чекбокс ".pptx"
                    showPptxFiles = !showPptxFiles;
                }
                else if (LOWORD(wParam) == 1007) { // Перевіряємо, чи клікнуто на кнопку "Про програму"
                    abou.ShowAboutDialog(hwnd); // Показ діалогового вікна "Про програму"
                }
            }
            if (HIWORD(wParam) == BN_CLICKED) {
```

```
            if (LOWORD(wParam) == 1006) { // Перевіряємо, чи клікнуто на кнопку "Статистика"
                rep.showStatistics(hwnd);
            }
        }
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        return 0;
    case WM_PAINT: {
        PAINTSTRUCT ps;
        HDC hdc = BeginPaint(hwnd, &ps);
        EndPaint(hwnd, &ps);

        rep.generateReport(hwnd); // Виклик функції для генерації звіту
        rep.drawFooter(hwnd);
    }
            break;
    case WM_SIZE:
        // Redraw the window to update the footer position
        RedrawWindow(hwnd, NULL, NULL, RDW_ERASE | RDW_INVALIDATE | RDW_FRAME | RDW_ALLCHILDREN);
        break;
    case WM_LBUTTONDOWN: {
        // Обробка натискання на заголовок стовпця
        int x = LOWORD(lParam);
        int y = HIWORD(lParam);

        int headerY = 100; // Початкова y-координата заголовка
        int headerHeight = 20; // Висота заголовка

        if (y >= headerY && y <= headerY + headerHeight) { // Перевіряємо, чи натиснуто на заголовок
            int headerWidths[] = { 150, 150, 150, 200 };
            int headerX = 20;
            int columnIndex = -1;
            for (int i = 0; i < 4; ++i) {
                if (x >= headerX && x <= headerX + headerWidths[i]) {
                    columnIndex = i + 1;
                    break;
                }
                headerX += headerWidths[i];
            }
            if (columnIndex != -1) {
                if (sortCriteria == columnIndex) {
                    reverseSort = !reverseSort;
                }
                else {
                    sortCriteria = columnIndex;
                    reverseSort = false;
                }
                rep.SortFiles(sortCriteria, reverseSort);
                RedrawWindow(hwnd, NULL, NULL, RDW_ERASE | RDW_INVALIDATE | RDW_FRAME | RDW_ALLCHILDREN);
            }
        }
        break;
    }
    default:
        return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}
```

Додаток Г

Ілюстративна частина

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

“Розробка програмного забезпечення системи аналізу файлів на диску і побудови звіту”

Студент: ст. гр. 4ПІ-21б
Назарчук Б.Ю.
Керівник: к.т.н., доцент
Черноволик Г. О.

Вінниця 2025

Рисунок Г.1 – слайд 1

Актуальність задачі

Зі зростанням обсягів цифрових даних аналіз файлової системи стає критично важливим для ефективного управління ресурсами. Розробка такого програмного забезпечення дозволяє автоматизувати контроль за зберіганням файлів і формуванням звітів, що є актуальним як для користувачів, так і для організацій.

Рисунок Г.2 – слайд 2

- **Метою роботи** є покращення організації та ефективності роботи з файловою системою шляхом створення інструменту для швидкого аналізу вмісту директорій, виявлення зайвих або великих файлів, а також побудови інформативного звіту на основі зібраних даних.
- **Об’єкт дослідження** – процес розробки програмного забезпечення для аналізу файлової системи та генерації звітів.
- **Предмет дослідження** – методи та інструменти розробки програмного забезпечення, зокрема принципи об’єктно-орієнтованого програмування мовою C++, а також засоби пошуку й аналізу файлів на диску, що використовуються у створенні застосунку.

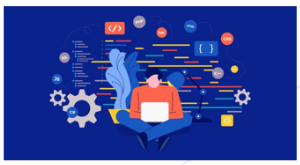


Рисунок Г.3 – слайд 3

Задачі

- Розробити архітектуру програмного застосунку для аналізу файлів;
- Розробити структуру додатку;
- Розробити дизайн додатку;
- Розробити методи, що відповідають за графічний інтерфейс та логіку програми;
- Розробити алгоритм роботи системи;
- Розробити функціонал системи;
- Провести тестування розробленої системи.

Рисунок Г.4 – слайд 4

Новизна отриманих результатів

- Подальшого розвитку набув метод детермінованого аналізу файлових об’єктів, у якому всі етапи виконуються послідовно та передбачувано. Новизна полягає у використанні ітеративного методу сканування замість рекурсивного, що дозволяє стабільно працювати навіть при великій кількості вкладених папок. Також був застосований простий спосіб пошуку дублікатів за іменем і розміром файлів, що дає змогу швидко отримувати результати без великих витрат ресурсів. Особливістю є відображення результатів у вигляді простої таблиці без дерева папок, що робить інтерфейс зрозумілішим. Звіт формується прямо в програмі без PDF або CSV, що спрощує реалізацію та використання. Усе це разом створює ефективний та зручний підхід до аналізу файлів на диску.

Рисунок Г.5 – слайд 5

Аналоги

- WinDirStat — безкоштовна програма для візуального аналізу використання дискового простору з детальними діаграмами та статистикою.
- WizTree — надшвидкий інструмент для сканування жорсткого диску і виявлення файлів, що займають найбільше місця.
- Disk Inventory X — візуалізатор файлової системи для macOS, що дозволяє зручно переглядати розподіл простору на диску.
- TreeSize Free — проста утиліта для Windows, яка показує, які папки займають найбільше місця на диску у вигляді дерева.

Рисунок Г.6 – слайд 6

Порівняльний аналіз аналогів

Критерії оцінювання	Disk Analyzer (Власна розробка)	WizTree	WinDirStat	TreeSize Free	Disk Inventor y X
Сканування та аналіз файлів та папок	1	1	1	1	1
Створення детальних звітів	1	0	1	1	1
Фільтрація та сортування результатів	1	1	0	0	1
Пошук дублікатів файлів	0	1	0	0	0
Збереження звіту у файл	0	0	0	1	0
Включення додаткової інформації про файли	1	0	0	0	0
Сумарний коефіцієнт	4	3	2	3	3

Рисунок Г.7 – слайд 7

Метод детермінованого аналізу файлів на диску

У розробленому підході застосовано чітко структурований алгоритм, який забезпечує стабільність і передбачуваність результатів. Метод базується на:

Ітеративному скануванні файлової системи — дозволяє ефективно обходити директорії великої глибини без ризику переповнення стеку.

Обробці зібраних даних — включає сортування файлів за розміром, групування за типами та виявлення дублікатів через порівняння імен та розмірів.

Формуванні звіту — результати виводяться в адаптивному табличному вигляді прямо у вікні програми, без потреби експорту в зовнішні формати.

Рисунок Г.8 – слайд 8

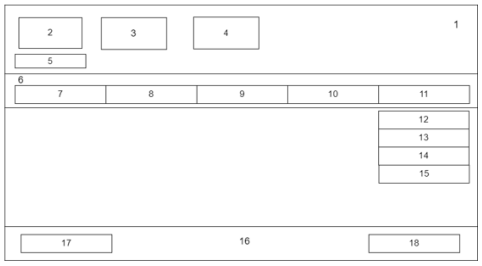
Рисунок Г.9 – слайд 9

Структурна схема головного вікна програмного застосунку

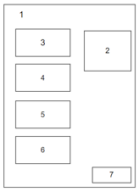
Структурна схема вікна про застосунок

Рисунок Г.10 – слайд 10

Графічні схеми



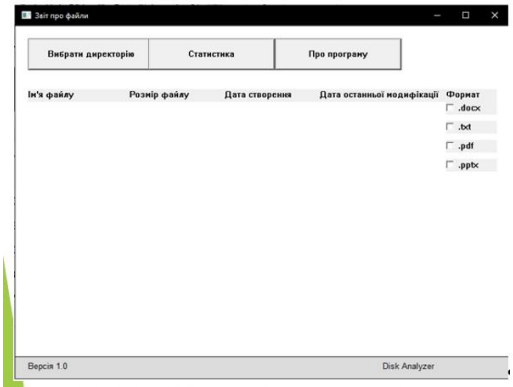
Графічна схема головного вікна застосунку



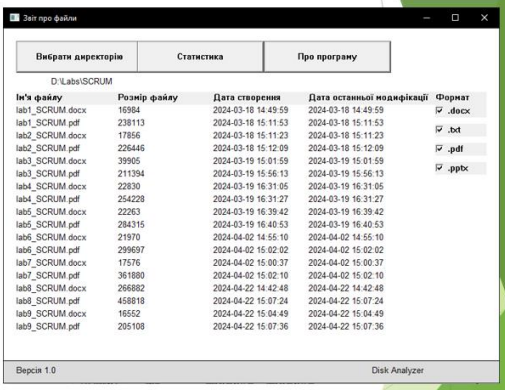
Графічна схема вікна про застосунок

Рисунок Г.11 – слайд 11

Головне вікно програмного застосунку



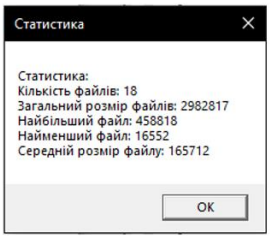
Графічний інтерфейс головного вікна



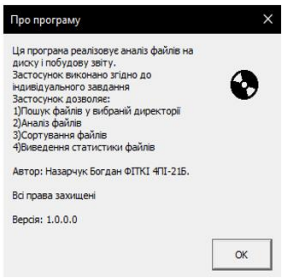
Результат аналізу файлів

Рисунок Г.12 – слайд 12

Додаткові вікна



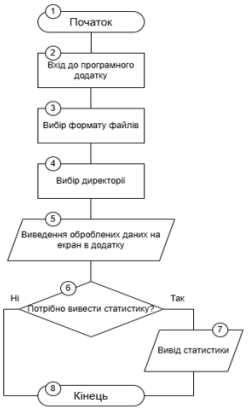
Вікно «Статистика»



Вікно «Про програму»

Рисунок Г.13 – слайд 13

Алгоритм системи



Діаграма діяльності розробленого додатка

Рисунок Г.14 – слайд 14

Блок-схема алгоритму роботи застосунку

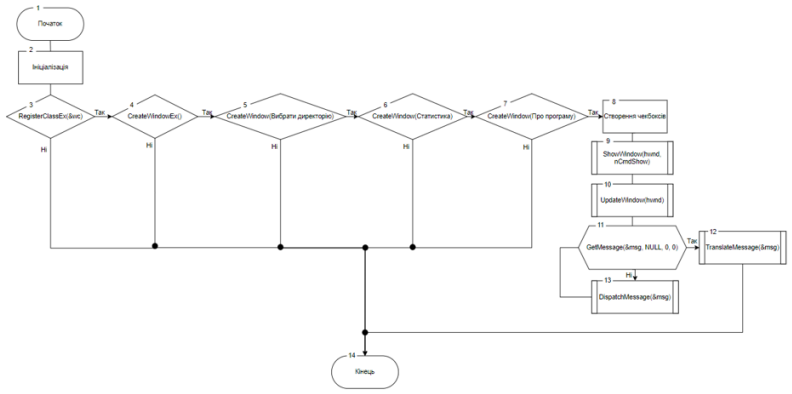


Рисунок Г.15 – слайд 15

Середовище розробки програмного додатку

Для розробки було обрано середовище Microsoft Visual Studio через його зручність та найбільшу кількість переваг у порівнянні з іншими доступними середовищами. Для повноцінного функціоналу застосунку було реалізовано ряд методів на мові C++,

Рисунок Г.16 – слайд 16

Розробка функцій програмного додатку

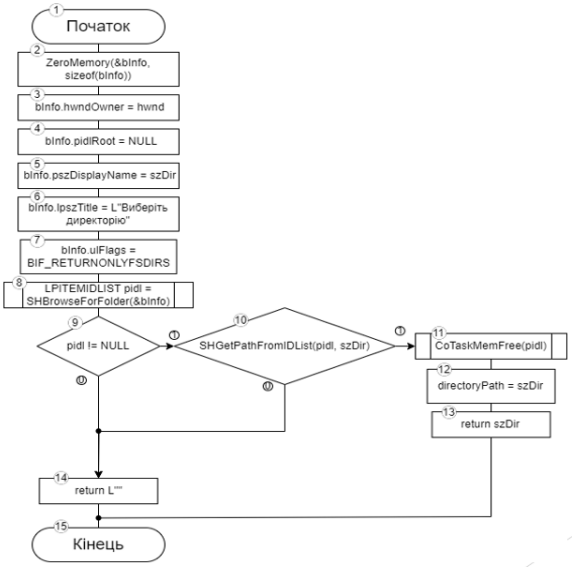


Рисунок Г.17 – слайд 17

Публікації й апробації

- **Апробація матеріалів бакалаврської кваліфікаційної роботи.** Основні положення БКР доповідалися та обговорювалися на Міжнародній конференції «Молодь в науці: дослідження, проблеми, перспективи Вінницького національного технічного університету (МН ВНТУ–2025).

Рисунок Г.18 – слайд 18

Висновки

У дипломній кваліфікаційній роботі було всебічно досліджено, спроєктовано та реалізовано програмний застосунок для аналізу файлів на диску з подальшим формуванням звітності.

Таким чином, у результаті виконаної роботи: реалізовано ефективний інструмент аналізу файлової структури директорій, забезпечено інформативне виведення статистичних даних, забезпечено стабільну роботу програмного забезпечення в стандартних та виняткових ситуаціях, виконано повний цикл від постановки задачі до тестування та аналізу результатів.

Розроблений застосунок може бути використаний як окремий інструмент для оптимізації роботи з файлами на цифрових носіях або як складова більших систем адміністрування та аудиту даних.

Рисунок Г.19 – слайд 19

Дякую за увагу!

Рисунок Г.20 – слайд 20