

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Застосунок для управління та обміну файлами»

Виконав студент групи ПЗкп-21-1
ОЛІЙНИК Олег Артемович

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
САВІЦЬКИЙ Роман Святославович

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«02» грудня 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ОЛІЙНИК Олег Артемович**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: **«Застосунок для управління та обміну файлами»**, затверджена
Наказом закладу вищої освіти від **«02» грудня 2024 р., №677/с**

Вихідні дані для роботи: **дослідження технологій та методів, які використовуються для організації та управління файлами.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,
що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю.М.	03.12.2024р.	03.12.2024р.
2	Єфремов Ю.М.	03.12.2024р.	03.12.2024р.
3	Єфремов Ю.М.	03.12.2024р.	03.12.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається із застосунку для управління та обміну файлами та пояснювальної записки. Пояснювальна записка до випускної роботи містить 69 сторінок, 33 ілюстрацій та 16 таблиць.

Метою роботи є створення інтуїтивно зрозумілого та функціонального застосунку для управління та обміну файлами з використанням сучасних технологій для управління файлами та папками.

Архітектура застосунку базується на модульній структурі, що забезпечує легкість розширення та підтримки. Основні функції застосунку включають створення, видалення, копіювання, переміщення та перейменування файлів та папок.

Додаток має зручний інтерфейс, який дозволяє користувачеві легко взаємодіяти з файловою системою, відображаючи вміст каталогів на екрані. Також передбачена можливість надсилання файлів.

КЛЮЧОВІ СЛОВА: ОБМІН ФАЙЛАМИ, УПРАВЛІННЯ ФАЙЛАМИ, ОРГАНІЗАЦІЯ ДАНИХ, ФАЙЛОВА СИСТЕМА, ІНТЕРФЕЙС КОРИСТУВАЧА

					ІПЗ.КР.Б-121-24-ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата					
Розроб.	Олійник О.А.				Застосунок для управління та обміну файлами		Літ.	Арк.	Аркушів
Керівник	Єфремов Ю.М.							3	69
							Житомирська політехніка, група ІПЗкп-21-1		
Н. контр.	Тичина О.П.								
Зав. каф.	Вакалюк Т.А.								

ABSTRACT

The bachelor's thesis consists of a application for managing and sharing files over a local network and an explanatory note. The explanatory note to the final thesis contains 69 pages, 33 illustrations and 16 tables.

The goal of the work is to create an intuitive and functional application for managing and sharing files over a local network using modern technologies for managing files and folders.

The architecture of the application is based on a modular structure, which ensures ease of expansion and maintenance. The main functions of the application include creating, deleting, copying, sending, moving and renaming files and folders.

The app has a convenient interface that allows the user to easily interact with the file system, displaying the contents of directories on the screen. It is also possible to send files over a local network.

KEYWORDS: FILE EXCHANGE, FILE MANAGEMENT, DATA ORGANIZATION, FILE SYSTEM, USER INTERFACE

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ЗМІСТ	5
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ.....	8
1.1 Аналіз поставленої задачі	8
1.2 Аналіз існуючого програмного забезпечення за тематикою роботи.....	9
1.3 Аналіз засобів та методів вирішення завдання.....	19
Висновки до першого розділу	21
РОЗДІЛ 2. ПРОЕКТУВАННЯ ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ТА ОБМІНУ ФАЙЛАМИ.....	22
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	22
2.2 Проектування та реалізація алгоритмів роботи системи.....	29
2.3 Реалізація функціоналу застосунку для управління та обміну файлами	30
Висновки до другого розділу.....	47
РОЗДІЛ 3. ТЕСТУВАННЯ ТА ПОРЯДОК РОБОТИ ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ТА ОБМІНУ ФАЙЛАМИ.....	48
3.1 Порядок встановлення та налаштування параметрів.....	48
3.2 Інтерфейс та принцип роботи з файлами	49
3.3 Тестування передачі файлів по локальній мережі.....	61
Висновки до третього розділу	66
ВИСНОВКИ.....	67
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	68
ДОДАТКИ.....	70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

FTP – File Transfer Protocol, мережевий протокол прикладного рівня

IP – Internet Protocol, числовий ідентифікатор мережевого рівня

LAN – локальна комп'ютерна мережа

TCP – протокол управління передачею

ZIP – формат стиснення та архівації даних

ООП – об'єктно-орієнтоване програмування

ПЗ – програмне забезпечення

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми зумовлена необхідністю забезпечення ефективного управління файлами та папками в сучасних комп'ютерних системах. У зв'язку з постійним розвитком інформаційних технологій та збільшенням обсягів даних виникає потреба у створенні зручних і функціональних інструментів для їх організації. Файловий менеджер, як важливий компонент операційних систем, надає користувачам можливість швидкого доступу до файлів, їх впорядкування та обробки.

Метою даної випускної кваліфікаційної роботи є створення інтуїтивно зрозумілого та функціонального застосунок для управління та обміну файлами.

Для досягнення мети дослідження необхідно виконати комплекс завдань, зокрема провести аналіз існуючих файлових менеджерів, їхніх функціональних можливостей, а також визначити оптимальні технології для створення нового програмного рішення. У процесі роботи буде розроблено застосунок, орієнтований на задоволення потреб користувачів у функціональності та зручності інтерфейсу.

Об'єктом дослідження є управління файлами та папками в операційних системах, а предметом — функціонал, архітектура, інтерфейс та технології, які застосовуються для створення файлових менеджерів.

Дослідження включає аналіз сучасних підходів до організації файлових систем, оцінку зручності інтерфейсу користувача, ефективності виконання базових операцій з файлами та можливостей розширення функціоналу через додаткові модулі й інтеграції.

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1 Аналіз поставленої задачі

Метою даної кваліфікаційної роботи бакалавра є створення інтуїтивно зрозумілого та функціонального застосунку для управління та обміну файлами, який дозволить користувачам ефективно керувати файлами та папками, забезпечуючи високу продуктивність та зручність роботи.

Для вирішення поставленої задачі передбачається використання об'єктно-орієнтованого підходу (ООП) — найбільш універсального та поширеного методу програмування. ООП дозволяє організувати програмний код у вигляді класів та об'єктів, що значно спрощує процес розробки, тестування та подальшої підтримки програмного забезпечення.

Файловий менеджер виступає важливим інструментом для роботи з файловою системою комп'ютера, забезпечуючи виконання таких операцій, як створення, видалення, копіювання, переміщення та перейменування файлів і папок. Крім того, передбачається реалізація функцій пошуку, сортування та зручного доступу до даних. Основною метою є створення програмного продукту, який підвищить продуктивність користувачів та зробить управління файлами максимально простим та зручним.

Для досягнення мети необхідно було виконати кілька ключових завдань. Спочатку здійснено аналіз існуючих рішень, що включало огляд функціональних можливостей популярних файлових менеджерів та визначення їхніх сильних і слабких сторін. Це дозволило отримати цінну інформацію для створення покращеної версії програми.

Далі було обрано засоби розробки, включаючи мову програмування, платформу та необхідні інструменти, щоб забезпечити ефективність створення, тестування та впровадження додатку. Особливу увагу приділено проектуванню дизайну інтерфейсу користувача. Розроблено загальну концепцію дизайну та

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

створено інтуїтивно зрозумілі UI-компоненти для забезпечення зручності використання програми.

У процесі реалізації функціоналу додано базові можливості файлового менеджера, такі як створення, видалення, копіювання, переміщення та перейменування файлів і папок. Також інтегровано функції пошуку та сортування за різними критеріями, забезпечивши можливість розширення функціоналу через додаткові модулі чи інтеграції.

Завершальним етапом стало тестування програмного продукту, яке включало перевірку роботи основних функцій та оцінку зручності графічного інтерфейсу, щоб переконатися у відповідності програми очікуванням користувачів.

Результатом виконання роботи стане створення застосунку для управління та обміну файлами, який забезпечить користувачам простий та швидкий доступ до файлів і папок, а також дозволить ефективно управляти ними. Розроблений додаток сприятиме підвищенню продуктивності користувачів під час роботи з локальними пристроями, надаючи їм зручний інструмент для організації даних та виконання щоденних завдань.

1.2 Аналіз існуючого програмного забезпечення за тематикою роботи

Розглянемо існуючі рішення у сфері програмного забезпечення для роботи з файлами та папками. Проаналізуємо їх функціональні можливості, інтерфейс, продуктивність та безпеку, щоб визначити аспекти, які можна вдосконалити, а також ті, які можна застосувати у розробці власного програмного продукту. Такий аналіз дозволяє оцінити поточний стан інструментів для управління файлами та створити ефективні рішення, що відповідають потребам користувачів у різних сценаріях.

Одним із найпоширеніших файлових менеджерів є Windows File Explorer, який є вбудованим компонентом операційних систем Windows. Цей менеджер, розроблений компанією Microsoft, вперше з'явився у ранніх версіях Windows і з часом став одним із ключових інструментів для управління файлами. Завдяки

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

простоті використання, інтеграції з системою та широкому функціоналу, Windows File Explorer є прикладом успішної реалізації файлового менеджера, що забезпечує базові можливості для роботи з файлами та папками.

Аналіз подібних рішень дозволяє оцінити їх сильні та слабкі сторони, врахувати досвід користувачів та створити програмний продукт, що відповідатиме сучасним вимогам зручності, продуктивності й функціональності. Це стане основою для створення файлового менеджера, здатного забезпечити ефективне управління даними та покращити взаємодію користувачів із файловою системою.

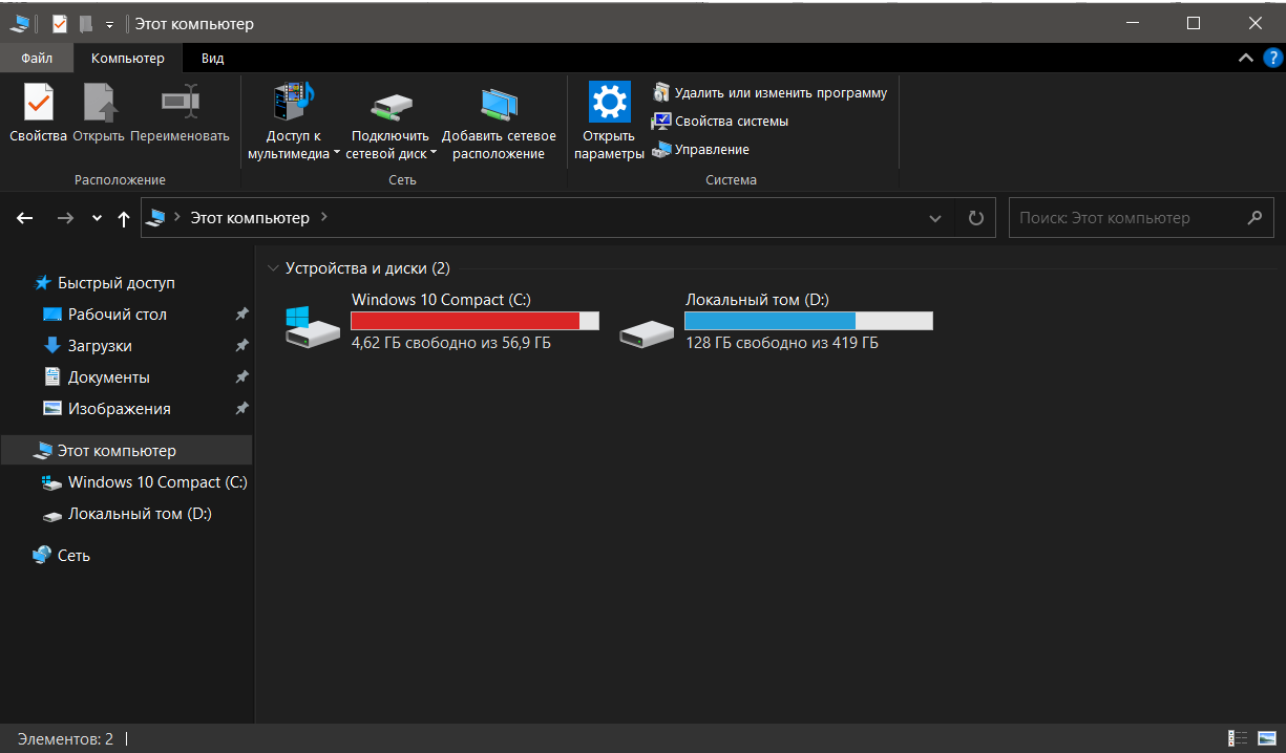


Рис. 1.1 Файловий менеджер Windows

Windows File Explorer має інтуїтивно зрозумілий інтерфейс, що робить його зручним для користувачів будь-якого рівня. Навігація по файловій системі здійснюється швидко та без зайвих складнощів. Усі основні дії, як-от копіювання, переміщення, видалення файлів чи створення папок, виконуються легко завдяки продуманій структурі інтерфейсу. Менеджер надає користувачам набір стандартних функцій для управління файлами: копіювання, переміщення, видалення, перейменування, створення нових файлів і папок. Крім того, є можливість швидкого пошуку, сортування за різними параметрами (ім'я, дата

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

створення, розмір) та попереднього перегляду файлів, що робить роботу із системою більш ефективною. Програма інтегрована з багатьма сервісами Windows. Наприклад, вона підтримує безпосередню роботу з текстовими редакторами, переглядом зображень, хмарними сховищами та системами пошуку. Завдяки цьому виконання багатьох завдань стає простішим, без потреби запускати окремі програми. File Explorer дозволяє користувачам працювати в межах їхнього облікового запису, враховуючи права доступу, встановлені адміністратором. Це забезпечує контроль безпеки файлів і можливість виконання дій тільки з тими файлами, до яких надано відповідні дозволи. У нових версіях менеджера з'являються нові можливості, як-от підтримка зовнішніх пристроїв (наприклад, флеш-накопичувачів і карт пам'яті), перегляд вмісту ZIP-архівів без додаткового програмного забезпечення, а також доступ до хмарних сховищ, що робить його універсальним для багатьох користувачів.

З недоліків можна навести те, що програмне забезпечення має мінімальний набір опцій для кастомізації. Наприклад, користувачі не можуть додавати нові панелі інструментів, змінювати розташування кнопок або створювати додаткові елементи в інтерфейсі, що зменшує його гнучкість для персонального налаштування. Проблемою також є недостатній функціонал для досвідчених користувачів. Користувачам, які потребують розширених функцій, як-от пакетне перейменування файлів, складні дії з архівами чи синхронізація папок, можливостей File Explorer може не вистачати. У таких випадках доводиться використовувати сторонні програми або інші файлові менеджери. Зниження продуктивності теж є помітною проблемою. При роботі з великим обсягом файлів або папок із великою кількістю елементів File Explorer може працювати повільніше, ніж альтернативи. Особливо це помітно при перегляді каталогів із тисячами файлів або відкритті директорій на мережевих ресурсах. Вагомим недоліком також є обмежена підтримка архівів. Менеджер має вбудовану підтримку ZIP-архівів, але не працює з іншими популярними форматами, такими як RAR або 7z. Це означає, що для обробки таких архівів користувачі змушені встановлювати додаткове програмне забезпечення. Також відсутня підтримка

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

декількох вкладок. File Explorer не дозволяє відкривати кілька вкладок (tabs) у межах одного вікна, як це реалізовано в сучасних браузерях або альтернативних файлових менеджерах. Це створює незручності при роботі з кількома папками одночасно, змушуючи користувачів відкривати окремі вікна. Недосконалою є функція пошуку. Хоча у File Explorer є вбудована система пошуку, вона не завжди працює достатньо швидко чи точно. Наприклад, пошук файлів із великими обсягами даних або специфічними атрибутами може бути повільним і неточним у порівнянні зі спеціалізованими програмами для пошуку. File Explorer не має подвійної панелі, яка значно спрощує виконання операцій між двома каталогами, наприклад, копіювання чи переміщення файлів. Це обмеження особливо помітне для користувачів, які звикли до подібного підходу в альтернативних файлових менеджерах, як-от Total Commander. В програмі не найкращий вбудований FTP/SFTP-клієнт. File Explorer підтримує роботу з віддаленими серверами через FTP чи SFTP-протоколи, але вони є недосконалими. Для більш якісного виконання таких завдань користувачі змушені звертатися до сторонніх інструментів, що створює додаткові труднощі та витрати часу.

Total Commander — це потужний і універсальний файловий менеджер для операційних систем Windows, який надає користувачам розширені можливості для керування файлами та папками.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

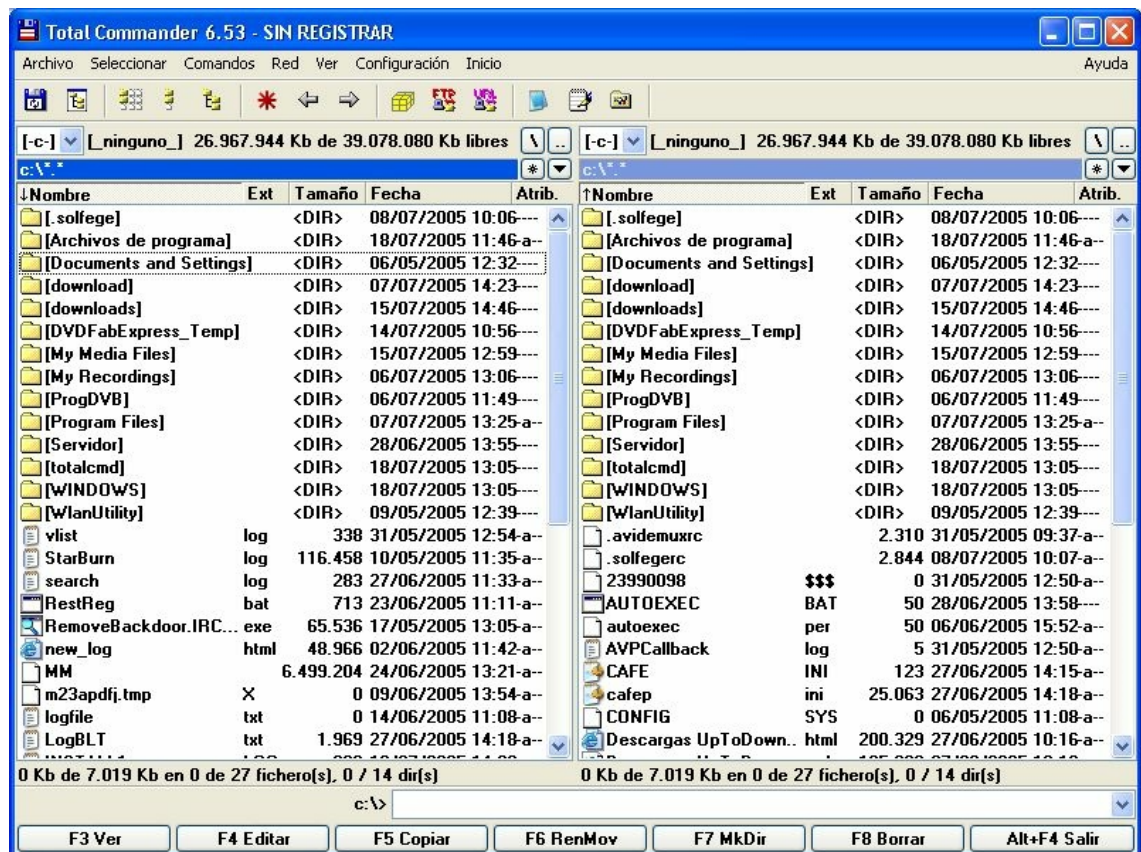


Рис. 1.2 Файловий менеджер Total Commander

Програма забезпечує одночасний доступ до двох каталогів, що значно полегшує копіювання, переміщення, та організацію файлів. Це зручно для користувачів, які часто працюють з великими обсягами даних. Total Commander дозволяє працювати з архівами багатьох форматів (ZIP, RAR, TAR, GZ тощо). Можна переглядати, розпаковувати і створювати архіви без необхідності встановлення додаткових програм. Програма надає можливість підключатися до віддалених серверів для передачі файлів, синхронізації каталогів, або завантаження даних, що особливо корисно для веб-розробників та адміністраторів мереж. Total Commander може працювати з мережевими ресурсами та дисками, спрощуючи доступ до спільних папок у корпоративних мережах. Користувачі можуть призначати комбінації клавіш для швидкого доступу до найчастіше використовуваних функцій, що значно пришвидшує роботу. Total Commander дозволяє налаштовувати вигляд під потреби користувача, включаючи зміну тем, піктограм та панелей інструментів. Інструмент пошуку дозволяє знаходити файли за різноманітними параметрами,

включаючи ім'я, розмір, дату створення, або навіть текстовий вміст файлів. Завдяки плагінам Total Commander можна розширити, додавши підтримку нових форматів файлів, архіваторів чи інших функцій. Програма дозволяє автоматично синхронізувати вміст локальних або віддалених каталогів, що корисно для резервного копіювання та підтримання актуальності даних. Зручний інструмент для пакетного перейменування дозволяє змінювати імена багатьох файлів одночасно за заданими правилами. Total Commander має функцію порівняння файлів за вмістом, що є корисним для виявлення відмінностей між двома версіями документа чи коду. Програма дозволяє відкривати та переглядати різноманітні формати (текстові, графічні, мультимедійні тощо) безпосередньо в середовищі Total Commander.

Інтерфейс програми може здатися складним для користувачів, які звикли до простих файлових менеджерів. Щоб повністю опанувати функціонал, потрібен час і практика. Total Commander не є безкоштовним. Хоча доступна демо-версія, для розблокування повного функціоналу потрібно придбати ліцензію. Інтерфейс виглядає старомодно в порівнянні з сучасними файловими менеджерами, що може не сподобатися користувачам, які віддають перевагу естетичному оформленню. Для доступу до таких сервісів, як Google Drive чи Dropbox, потрібно використовувати додаткові плагіни, що не завжди зручно. Щоб максимально ефективно використовувати Total Commander, потрібно витратити час на його налаштування та встановлення потрібних плагінів, що може бути складним для недосвідчених користувачів. Деякі функції програми можуть бути прихованими або недостатньо очевидними, через що користувачам доводиться звертатися до документації чи навчальних матеріалів. Нові версії Windows або сторонні плагіни інколи можуть викликати збої в роботі Total Commander, особливо без оновлення програми до останньої версії.

Commander One – це багатофункціональний двопанельний файловий менеджер для macOS, створений компанією Eltima Software. Програма надає користувачам інтуїтивний інтерфейс для управління файлами та папками, а також додаткові інструменти для роботи з архівами, серверами та хмарними

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

сервісами. Завдяки розширеним функціям, Commander One підходить як для повсякденного використання, так і для виконання складних завдань.

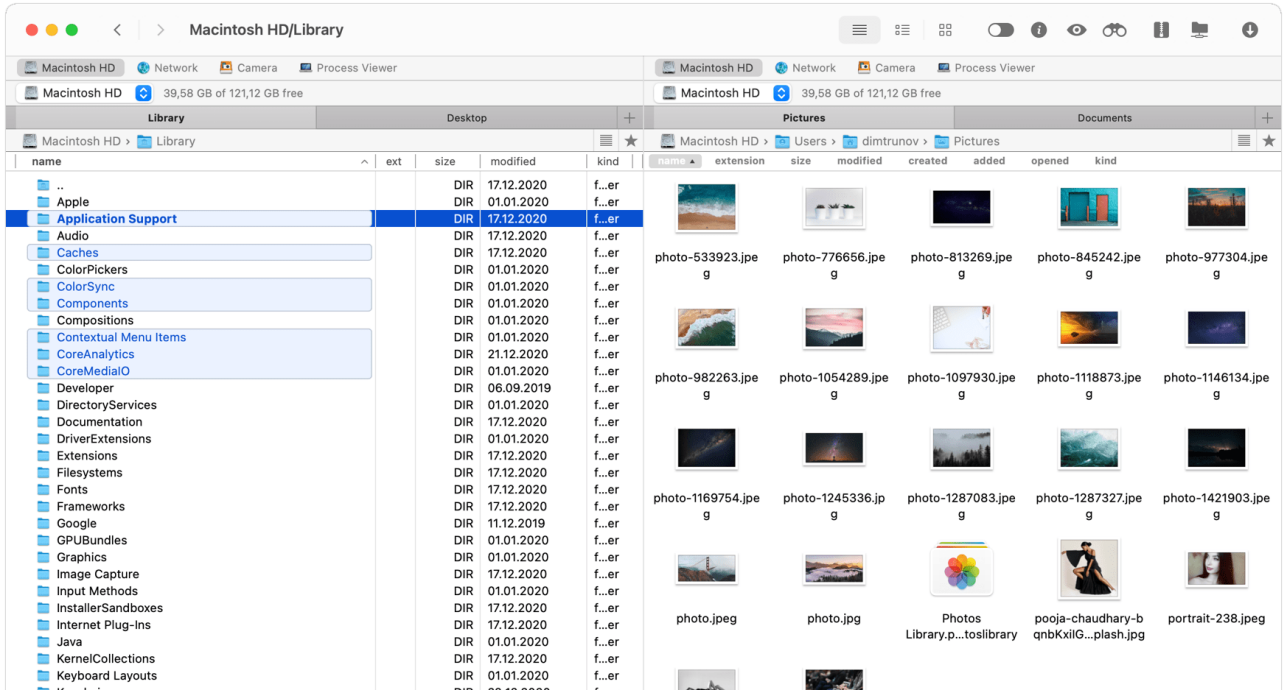


Рис. 1.3 Файловий менеджер Commander One

Завдяки двопанельному дизайну, користувачі можуть одночасно працювати з двома каталогами, що спрощує виконання завдань із копіювання, переміщення та управління файлами. Програма працює з форматами ZIP, RAR, 7z та іншими, дозволяючи як витягати файли з архівів, так і створювати нові, без потреби встановлювати додаткові утиліти. Вбудований клієнт FTP/SFTP забезпечує доступ до віддалених серверів, що дозволяє безпосередньо передавати, редагувати чи видаляти файли. Commander One підтримує такі популярні сервіси, як Google Drive, Dropbox, OneDrive та інші, що дозволяє зручно управляти файлами у хмарному середовищі. Користувачі можуть відкривати кілька директорій у вкладках в межах одного вікна, що значно підвищує ефективність багатозадачної роботи. Потужний інструмент для пошуку дозволяє швидко знаходити файли та папки, враховуючи як локальні дані, так і файли на підключених сервісах. Програма дозволяє налаштовувати інтерфейс, гарячі клавіші та інші параметри, що робить її максимально адаптованою до потреб користувача.

Безкоштовна версія має обмежений функціонал, а доступ до Pro-версії потребує оплати, що може стати перепорою для користувачів, які шукають повністю безкоштовні рішення. Порівняно з Finder, програма може вимагати більше ресурсів, що впливає на продуктивність на старих або слабких пристроях. Деякі специфічні можливості, доступні в інших файлових менеджерах, можуть бути відсутніми в Commander One, що обмежує його функціональність у певних ситуаціях. Хоча підтримка хмарних платформ наявна, вона може поступатися за зручністю спеціалізованим клієнтам цих сервісів. Також програма доступна виключно для macOS, що виключає можливість її використання на Windows чи Linux і звужує аудиторію.

Finder — це основний файловий менеджер операційної системи macOS, створений компанією Apple. Програма надає користувачам інтуїтивно зрозумілий інтерфейс для роботи з файлами та папками, що дозволяє переглядати, організовувати й управляти даними на комп'ютері. Finder є невід'ємною частиною macOS, тісно інтегрованою з іншими функціями операційної системи, включаючи iCloud, Spotlight, та інші сервіси, які забезпечують зручну взаємодію з екосистемою Apple.

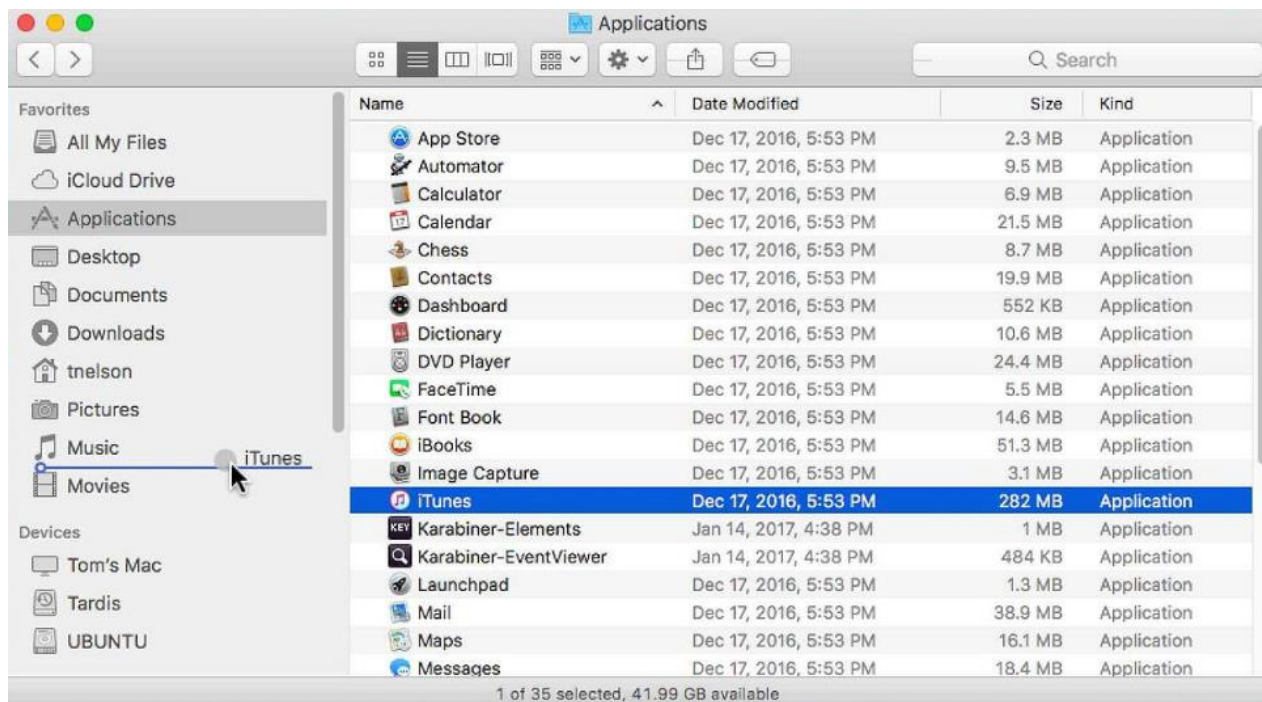


Рис. 1.4 Файловий менеджер Finder

Простий у використанні, що робить його зручним навіть для початківців. Quick Look дає змогу швидко переглядати файли без необхідності відкривати спеціалізовані програми, що заощаджує час. Легко синхронізує файли з iCloud, забезпечуючи доступ до них з будь-якого пристрою Apple. Дає можливість додавати теги до файлів і папок для зручнішої організації та пошуку. Спрощує управління різними директоріями одночасно завдяки можливості працювати з кількома вкладками в одному вікні. Має глибоку інтеграцію з іншими додатками та сервісами macOS, такими як Spotlight і Siri.

Має менше можливостей для налаштування інтерфейсу та функціональності порівняно з деякими сторонніми менеджерами файлів. При роботі з великою кількістю файлів або великими файлами може спостерігатися певне зниження продуктивності. Не має деяких розширених функцій, таких як FTP-клієнти чи мережеві інструменти, доступні в сторонніх програмах. Менш гнучкий у роботі з різними форматами архівів порівняно з деякими сторонніми додатками. Доступний тільки для користувачів macOS, що обмежує його використання на інших платформах.

Dolphin — це файловий менеджер для робочого середовища KDE на Linux, який забезпечує зручне та гнучке управління файлами та папками. Завдяки інтуїтивно зрозумілому інтерфейсу та широкому набору функцій, Dolphin є популярним серед користувачів KDE. Цей менеджер дозволяє легко організовувати, знаходити та редагувати файли, а також підтримує розширену налаштуванням для оптимізації робочого процесу. Dolphin також пропонує підтримку вкладок, перегляду вмісту без відкриття файлів, інтеграцію з різними додатками та підтримку мережевих ресурсів.

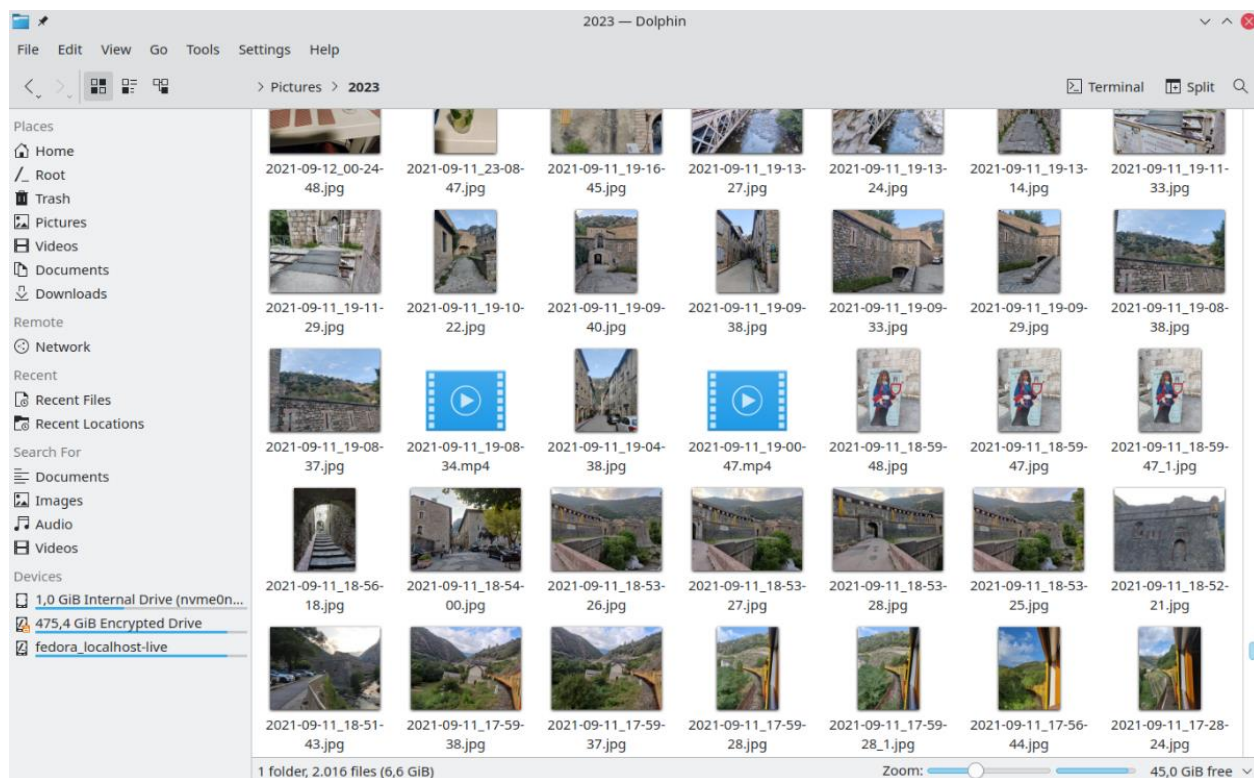


Рис. 1.5 Файловий менеджер Dolphin.

Простий і зрозумілий інтерфейс, який легко освоїти навіть для новачків. Можливість відкривати кілька вкладок в одному вікні, що дозволяє зручно працювати з різними директоріями одночасно. Можливість детальної налаштування інтерфейсу та функцій для адаптації до індивідуальних потреб користувача. Використання бічних панелей для швидкого доступу до важливих місць і директорій. Тісна інтеграція з середовищем KDE, що забезпечує узгодженість інтерфейсу та поведінки додатків. Вбудовані засоби для швидкого пошуку та фільтрації файлів і папок. Робота з різними мережевими протоколами (FTP, SFTP, SMB) для зручного доступу до віддалених ресурсів. Можливість попереднього перегляду вмісту файлів, таких як зображення, відео та документи. Підтримка плагінів для розширення функціональності.

Найкраще працює в середовищі KDE, і в інших середовищах можуть виникати проблеми з інтеграцією та виглядом. Може споживати більше системних ресурсів порівняно з іншими більш легкими файловими менеджерами, такими як Thunar або PCManFM. Велика кількість налаштувань і функцій може бути надмірною для користувачів, яким потрібен простий файловий менеджер. Деякі користувачі можуть стикатися з проблемами

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		18

сумісності або помилками при роботі з певними типами файлів чи мережевими ресурсами.

Таблиця 1.1

Підсумкова таблиця результатів

Назва / Критерій	Dolphin	Total Commander	Finder	Windows Explorer	Control
Простий інтерфейс	-	-	+	+	+
Декілька вкладок	-	+	-	-	+
Швидкість виконання операцій	+/-	+	-	-	+
Компактність	-	-	-	+/-	+
Зміна стилю	+	-	-	+	+
Проста передача файлів по мережі	-	-	-	+	+

1.3 Аналіз засобів та методів вирішення завдання

Програмне забезпечення буде розроблене із застосуванням Windows Forms, що дозволяє створювати як інтерфейс користувача, так і функціональну частину додатка. Це API дає можливість створювати програми, які працюють на різних версіях операційної системи Windows без значних змін або адаптацій. Windows Forms — це API для графічного інтерфейсу користувача в Microsoft .NET Framework, яке спрощує доступ до елементів інтерфейсу Windows, обгортаючи існуючий Win32 API в керованому коді. Це дозволяє розробникам використовувати Windows Forms незалежно від мови програмування, такої як C#, C++, VB.Net, J# та інші. Windows Forms також дозволяє створювати кросплатформенні графічні інтерфейси користувача, використовуючи прямий доступ до Win32-функцій, які не доступні на інших платформах.

Windows Forms надає зручний інтерфейс для створення графічного інтерфейсу, що дозволяє розробникам швидко створювати візуальні елементи та взаємодіяти з ними. Бібліотека включає в себе багато вбудованих елементів

управління, таких як кнопки, тексти, списки, таблиці тощо, що спрощує створення складних інтерфейсів. Windows Forms інтегровано з .NET Framework, що дозволяє використовувати всі можливості цієї платформи, включаючи мови програмування, класи, бібліотеки та інструменти розробки. Цей інтерфейс програмування надає простий механізм для обробки подій, що дозволяє реагувати на дії користувача, такі як клік мишею чи натискання клавіші. Windows Forms дозволяє застосовувати різні стилі та теми оформлення, що дає змогу створювати інтерфейси згідно з вимогами користувачів або дизайном програми. Інтегровані інструменти в Visual Studio надають великий набір функцій для проектування і редагування інтерфейсу користувача, що дозволяє зручно працювати з візуальними елементами.

C# — це сучасна об'єктно-орієнтована мова програмування, розроблена компанією Microsoft. C# охоплює статичну типізацію, сильну типізацію, лексичну область видимості та підтримує імперативне, декларативне, функціональне, загальне об'єктно-орієнтоване та компонентно-орієнтоване програмування.

C# має чистий синтаксис і схожий на C++, Java, що полегшує вивчення для тих, хто вже має досвід роботи з іншими мовами програмування. Ця мова програмування підтримує об'єктно-орієнтоване програмування, що дозволяє створювати логічно структуровані програми з використанням класів, об'єктів і наслідування. Також C# підтримує різноманітні типи даних, включаючи числові, рядкові, масиви, а також вбудовані колекції. Вона також має велику кількість вбудованих бібліотек для різних завдань, що спрощує розробку програм. Завдяки .NET Framework програми, написані на C#, можуть працювати на різних платформах, таких як Windows, macOS та Linux. C# має вбудовані засоби для керування пам'яттю та перевірки на відсутність переповнення буфера, що дозволяє знизити ймовірність виникнення помилок під час розробки. Також .NET Framework дає змогу використовувати C# разом з іншими мовами програмування, такими як Visual Basic .NET та F#, а також інтегруватися з іншими технологіями Microsoft та сторонніми розробниками.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

Висновки до першого розділу

Було проведено аналіз предметної області, що дозволив визначити ключові вимоги до файлового менеджера, спрямованого на організацію роботи з файлами.

Під час дослідження подібних продуктів було проаналізовано і порівняно різні додатки, що використовують схожі принципи роботи, і забезпечують функціональність для створення, видалення, копіювання та переміщення файлів і папок.

На основі визначених вимог і критеріїв було сформовано набір основних технологій і засобів для реалізації додатку.

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЕКТУВАННЯ ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ТА ОБМІНУ ФАЙЛАМИ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

Застосунок забезпечує інтерфейс користувача, який дає змогу керувати файлами та папками, виконуючи різні операції, такі як створення, копіювання, переміщення, видалення та перейменування файлів. Функціональність застосунку залежить від дій користувачів, які можуть виконувати різноманітні операції для організації своїх файлів та обміну ними через локальну мережу (LAN). Для зручності та наочності взаємодії між елементами додатку була розроблена діаграма варіантів використання (рис. 2.1).

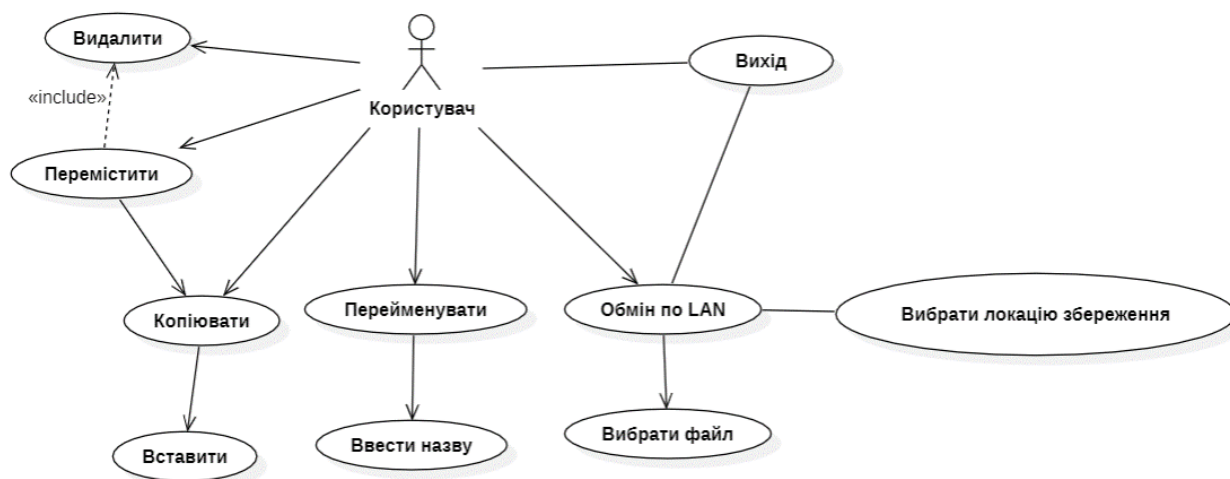


Рис. 2.1 Діаграма варіантів використання



Рис. 2.2 Компоненти та відношення діаграми використання

Опис діаграми варіантів використання наведено в таблицях 2.1 – 2.15.

Таблиця 2.1

Опис прецеденту «Запуск програми»

Ім'я варіанту використання: Запуск програми		Рівень важливості: високий	
Основний актор: Користувач			
Короткий опис:		Відкриття програми та перехід до дерева каталог	

Таблиця 2.2

Опис прецеденту «Відкрити файл»

Ім'я варіанту використання: Відкрити файл		Рівень важливості: високий	
Основний актор: Користувач			
Короткий опис:		При натисненні на файл користувач матиме можливість відкрити його.	

Таблиця 2.3

Опис прецеденту «Відкрити каталог»

Ім'я варіанту використання: Відкрити каталог	Рівень важливості: високий
Основний актор: Користувач	

Короткий опис:	При натисненні на каталог користувач перейде до його вмісту.
----------------	--

Таблиця 2.4

Опис прецеденту «Створити файл»

Ім'я варіанту використання: Створити файл	Рівень важливості: низький
Основний актор: Користувач	
Короткий опис:	Користувач має можливість створити текстовий файл.

Таблиця 2.5

Опис прецеденту «Створити каталог»

Ім'я варіанту використання: Створити каталог	Рівень важливості: низький
Основний актор: Користувач	
Короткий опис:	Користувач має можливість створити каталог

Таблиця 2.6

Опис прецеденту «Копіювання даних»

Ім'я варіанту використання: Копіювання даних	Рівень важливості: низький
Основний актор: Користувач	
Короткий опис:	Користувач має можливість копіювати бажані дані

Таблиця 2.7

Опис прецеденту «Вставка даних»

Ім'я варіанту використання: Вставка даних	Рівень важливості: низький
Основний актор: Користувач	
Короткий опис:	Користувач має можливість вставити раніше скопійовані дані

Таблиця 2.8

Опис прецеденту «Переміщення даних»

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		24

Ім'я варіанту використання: Переміщення даних	Рівень важливості: низький
Основний актор: Користувач	
Короткий опис:	При виборі функції переміщенні дані будуть видалені з локації копіювання та скопійовані в локацію вставки

Таблиця 2.9

Опис прецеденту «Перейменування»

Ім'я варіанту використання: Перейменування	Рівень важливості: низький
Основний актор: Користувач	
Короткий опис:	Користувач може перейменувати будь-який файл або каталог, за умови що введена назва не використовується

Таблиця 2.10

Опис прецеденту «Відправити дані по локальній мережі»

Ім'я варіанту використання: Відправити дані по локальній мережі	Рівень важливості: високий
Основний актор: Користувач	
Короткий опис:	Користувач може надсилати дані після вибору отримувача у вікні або введенні його IP-адреси

Таблиця 2.11

Опис прецеденту «Отримати дані по локальній мережі»

Ім'я варіанту використання: Отримати дані по локальній мережі	Рівень важливості: високий
Основний актор: Користувач	
Короткий опис:	Користувач може отримати дані після того як натисне кнопку запуску сервера

Таблиця 2.12

Опис прецеденту «Зміна локації збереження вхідних даних»

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		25

Ім'я варіанту використання: Зміна локації збереження вхідних даних	Рівень важливості: низький
Основний актор: Користувач	
Короткий опис:	Користувач має можливість змінювати локацію збереження вхідних даних

Таблиця 2.13

Опис прецеденту «Зупинка серверу»

Ім'я варіанту використання: Зупинка серверу	Рівень важливості: високий
Основний актор: Користувач	
Короткий опис:	Користувач може зупинити сервер, щоб безпечно повернутись до головного вікна

Таблиця 2.14

Опис прецеденту «Пошук»

Ім'я варіанту використання: Пошук	Рівень важливості: високий
Основний актор: Користувач	
Короткий опис:	Користувач може знайти файли в конкретному розділі

Таблиця 2.15

Опис прецеденту «Вихід»

Ім'я варіанту використання: Вихід	Рівень важливості: високий
Основний актор: Користувач	
Короткий опис:	Вихід з додатку

Основна ціль розробки застосунку полягає в створенні зручного та ефективного інструменту для організації та керування файлами і папками на комп'ютері. Менеджер має забезпечити користувачам можливість швидко та легко виконувати основні операції з файлами, такі як створення, копіювання, переміщення, перейменування та видалення файлів.

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		26

Основною метою є спрощення доступу до даних, підвищення продуктивності та зручності користувачів при роботі з файлами. Застосунок пропонує інтуїтивно зрозумілий інтерфейс для організації даних, підтримуючи різноманітні формати файлів і забезпечуючи легку навігацію по файловій системі. Завдяки цьому, основною метою є полегшення роботи з файлами та підвищення ефективності виконання повсякденних завдань, пов'язаних з управлінням інформацією на комп'ютері.

Бізнес-вимоги передбачають створення десктопного додатка, основною метою якого є спрощення роботи з файлами та папками на комп'ютері. Проект реалізується як структурований інструмент із функціями для управління файлами. Серед функціональних можливостей додатку передбачено створення, копіювання, переміщення, перейменування та видалення файлів і папок, підтримку обміну файлами по локальній мережі, інтуїтивний інтерфейс для вибору файлів і папок, можливість вибору місця збереження файлів і вихід із додатку.

Нефункціональні вимоги включають простоту сприйняття, зокрема, додаток повинен бути інтуїтивно зрозумілим і легко освоюваним за 5 хвилин. Система повинна бути надійною, працювати без помилок і забезпечувати стабільну продуктивність без зниження ефективності. Оновлення версій повинно виконуватися автоматично, відповідно до налаштувань користувача.

Системні вимоги передбачають мінімальну конфігурацію в 256 МБ пам'яті, процесор із тактовою частотою 600 MHz, і операційну систему Windows 7 або новішу. Усі інтерфейси програми повинні відповідати стандартам користувацького інтерфейсу Windows. Обмеження включають недопустимість використання інформації, що суперечить нормам моралі, законодавству або може зашкодити психіці користувачів. Аналіз функціональних вимог дозволив визначити основні сутності, які є необхідними для програмної реалізації додатку.

На рисунку 2.3 представлено діаграму класів, що використовуються для роботи додатку.

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

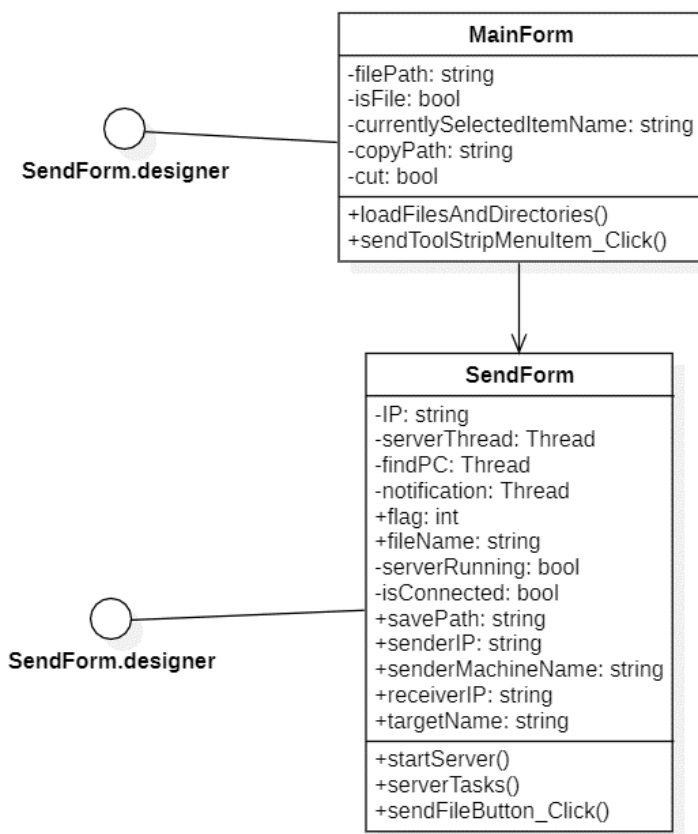


Рис. 2.3 Діаграма класів

Клас MainForm відповідає за головну форму додатка. У ньому визначені властивості, такі як filePath, що зберігає шлях до файлу, isFile, що вказує, чи є вибраний елемент файлом, і currentlySelectedItemName, що містить ім'я поточного вибраного елемента. Також клас включає властивості copyPath для збереження шляху до файлу для копіювання та cut, яка визначає, чи вибраний елемент для вирізання. Основними методами класу є loadFilesAndDirectories для завантаження файлів і директорій, а також sendToolStripMenuItem_Click для обробки події кліку на пункт меню відправки.

Клас SendForm відповідає за функціональність форми для відправки файлів. У ньому визначені властивості для роботи з мережею, такі як IP для збереження IP-адреси, serverThread для запуску потоку сервера, і findPC, що відповідає за пошук інших пристроїв. Додатково, notification використовується для роботи зі сповіщеннями, а flag зберігає стан системи. Клас також включає властивості fileName для імені файлу, serverRunning для перевірки роботи

сервера, і isConnected, що вказує на наявність з'єднання. Змінні savePath, senderIP, senderName, receiverIP і receiverName зберігають відповідно шлях для збереження файлу, IP-адресу та ім'я машини відправника й отримувача. Основні методи включають startServer для запуску сервера, serverTasks для отримання вхідних файлів, і sendFileButton_Click для обробки події кліку на кнопку відправки файлу.

2.2 Проектування та реалізація алгоритмів роботи системи

Додаток є простим у використанні та містить усі основні функції файлового менеджера, що дає користувачам змогу легко взаємодіяти з файлами. Алгоритм роботи з застосунком представлений на рисунку 2.4, де детально описано кожен етап його роботи.

У головному вікні користувач бачить список усіх доступних файлів і папок. Взаємодія з файлами передбачає зміну їхніх назв, створення нових файлів або папок, видалення вибраних елементів, а також копіювання й вставку. У разі введення вже існуючої назви під час перейменування, процес припиняється. При переміщенні файлу або папки оригінал видаляється.

Користувач також має доступ до функції надсилання файлів. Для цього запускається сервер, який починає пошук доступних ПК для обміну. Якщо отримувач доступний, файли передаються між пристроями. У разі недоступності отримувача сервер припиняє свою роботу. Завершення процесу обміну файлів здійснюється через зупинку сервера.

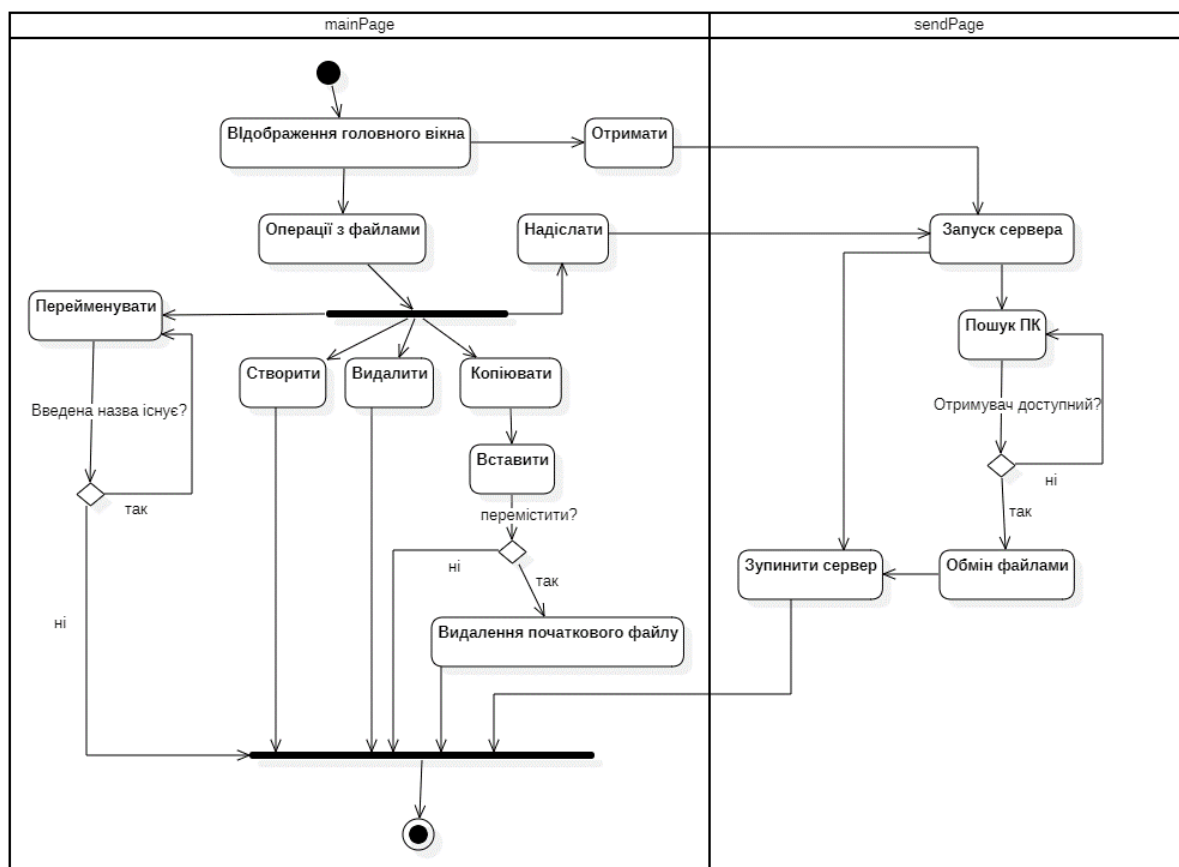


Рис. 2.4 Діаграма активності

2.3 Реалізація функціоналу застосунку для управління та обміну файлами

Розглянемо процес завантаження змісту та виконання операцій над ними.

Метод `loadListView1` у застосунку відповідає за завантаження та відображення файлів і папок у користувацькому інтерфейсі. На початку створюються змінні для зберігання інформації про папки, тимчасового шляху до файлу та атрибутів файлу. Якщо вибраний елемент є файлом, формується його повний шлях, створюється об'єкт для отримання інформації про файл, а текстові поля інтерфейсу оновлюються для відображення імені файлу та його розширення. Далі зчитуються атрибути файлу, і файл відкривається за допомогою відповідного методу. Якщо вибраний елемент є папкою, зчитуються атрибути для зазначеного шляху, отримується список файлів і папок у поточному каталозі, після чого відбувається перехід до їхнього вмісту. Для кожного файлу

з каталогу перевіряється, чи є його розширення в списку зображень, пов'язаних із розширеннями. Якщо немає, до списку додається значок, асоційована з файлом. Потім отримується дата останньої зміни, розраховується розмір файлу в кілобайтах, визначається його тип, і створюється елемент списку з відповідними параметрами. Цей елемент також додається до listView1.

Лістинг методу loadListView1:

```
public void loadListView1()
{
    DirectoryInfo fileList;
    string tempFilePath = "";
    FileAttributes fileAttr;
    try
    {
        if (isFile1)
        {
            if (File.Exists(Path.GetFullPath(selectedItems1[0])))
                tempFilePath = Path.GetFullPath(selectedItems1[0]);
            else
                tempFilePath = addBackSlash() + selectedItems1[0];
            FileInfo fileDetails = new FileInfo(tempFilePath);
            fileAttr = File.GetAttributes(tempFilePath);
            Process.Start(tempFilePath);
        }
        else
        {
            fileAttr = File.GetAttributes(filePath1);
        }

        if ((fileAttr & FileAttributes.Directory) == FileAttributes.Directory)
        {
            string subOutput, date;
            ListViewItem item;
            Icon fileIcon = SystemIcons.WinLogo;
            fileList = new DirectoryInfo(filePath1);
            FileInfo[] files = fileList.GetFiles();
            DirectoryInfo[] dirs = fileList.GetDirectories();
            listView1.Items.Clear();
            listView1.BeginUpdate();

            foreach (DirectoryInfo dir in dirs)
            {
                date = dir.LastWriteTime.ToString("dd.MM.yyyy HH:mm");
                subOutput = " ";
                item = new ListViewItem(dir.Name);
                item.ImageIndex = 0;
                item.SubItems.Add(date);
                item.SubItems.Add("directory");
                item.SubItems.Add(subOutput);
                listView1.Items.Add(item);
            }

            foreach (FileInfo file in files)
            {
                if (!imageList1.Images.ContainsKey(file.Extension))
                {
                    fileIcon =
                        System.Drawing.Icon.ExtractAssociatedIcon(file.FullName);
                    imageList1.Images.Add(file.Extension, fileIcon);
                }
            }
        }
    }
    catch { }
}
```

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						31
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
    date = file.LastWriteTime.ToString("dd.MM.yyyy HH:mm");
    subOutput = $"{file.Length / 1000:n0} kb";
    item = new ListViewItem(file.Name);
    item.ImageKey = file.Extension;
    item.SubItems.Add(date);
    item.SubItems.Add(MimeTypes.GetContentType(file.FullName));
    item.SubItems.Add(subOutput);
    listView1.Items.Add(item);
}
listView1.EndUpdate();
}
}
catch (Exception e)
{
    Console.WriteLine("loadListView1" + Path.GetFullPath(selectedItems1[0]));
    Console.WriteLine(e.Message);
}
}

```

Метод goBack1 забезпечує навігацію у файловій системі, дозволяючи перейти до попереднього каталогу. Основна його функція — змінити поточний шлях на рівень вище. Для цього метод отримує поточний шлях із текстового поля, знаходить останній символ "\" і зберігає частину рядка, яка передує цьому символу. Потім оновлюється властивість, що позначає поточний шлях як каталог, а текстове поле змінюється для відображення нового шляху.

Лістинг методу goBack1:

```

public void goBack1()
{
    try
    {
        string path = filePathTextBox1.Text;
        path = path.Substring(0, path.LastIndexOf("\\")+1);
        this.isFile1 = false;
        filePathTextBox1.Text = path;
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}

```

Метод listView1_ItemSelectionChanged обробляє подію зміни вибору елементів у компоненті інтерфейсу. Він виконується щоразу, коли користувач змінює вибрані елементи списку. Основна мета методу — оновити список вибраних елементів, визначити координати вибраного елемента та перевірити, чи є вибраний елемент файлом або каталогом. На початку метод створює новий список для зберігання імен вибраних елементів. Усі вибрані елементи перераховуються в циклі, і їхні імена додаються до списку. Далі перевіряється,

чи існує файл за вказаним шляхом. Шлях формується на основі імені вибраного елемента та доповнюється слешем у разі необхідності. Залежно від атрибутів, визначається, чи є вибраний елемент каталогом або файлом.

Лістинг методу listView1_ItemSelectionChanged:

```
private void listView1_ItemSelectionChanged(object sender,
ListViewItemSelectionChangedEventArgs e)
{
    try
    {
        selectedItems1 = new List<string>();
        foreach (ListViewItem item in listView1.SelectedItems)
        {
            selectedItems1.Add(item.Text);
            Console.Write(item.Text + "\t");
        }
        x = e.Item.Position.X;
        y = e.Item.Position.Y;

        FileAttributes fileAttr;
        if (File.Exists(Path.GetFullPath(selectedItems1[0])))
            fileAttr = File.GetAttributes(selectedItems1[0]);
        else
            fileAttr = File.GetAttributes(addBackSlash() + selectedItems1[0]);

        if ((fileAttr & FileAttributes.Directory) == FileAttributes.Directory)
        {
            isFile1 = false;
        }
        else
        {
            isFile1 = true;
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

Метод InitializeUI відповідає за налаштування користувацького інтерфейсу відповідно до заданого параметра конфігурації. Він приймає рядковий аргумент, який використовується для отримання поточного налаштування з файлу конфігурації. Якщо параметр має значення "light", метод змінює текст кнопки для вибору кольорового режиму та налаштовує кольори інтерфейсу для світлого режиму. Це включає зміну кольору тексту та фону форми, списку файлів і контекстного меню. Після цього значення конфігураційного параметра оновлюється на "dark". Якщо параметр має значення "dark", метод налаштовує інтерфейс для темного режиму. Текст кнопки оновлюється, а кольори тексту та

фону адаптуються для темного режиму. Після завершення значення параметра конфігурації змінюється на "light".

Лістинг методу InitializeUI:

```
private void InitializeUI(string key)
{
    try
    {
        var uiMode = ConfigurationManager.AppSettings[key];
        if (uiMode == "light")
        {
            btnChangeMode.Text = "Dark Mode";
            this.ForeColor = Color.FromArgb(47, 54, 64);
            this.BackColor = Color.FromArgb(255, 255, 255);
            listView1.ForeColor = Color.FromArgb(47, 54, 64);
            listView1.BackColor = Color.FromArgb(255, 255, 255);
            listView2.ForeColor = Color.FromArgb(47, 54, 64);
            listView2.BackColor = Color.FromArgb(255, 255, 255);
            treeView1.ForeColor = Color.FromArgb(47, 54, 64);
            treeView1.BackColor = Color.FromArgb(255, 255, 255);
            filePathTextBox1.ForeColor = Color.FromArgb(47, 54, 64);
            filePathTextBox1.BackColor = Color.FromArgb(255, 255, 255);
            filePathTextBox2.ForeColor = Color.FromArgb(47, 54, 64);
            filePathTextBox2.BackColor = Color.FromArgb(255, 255, 255);
            searchTextBox.ForeColor = Color.FromArgb(47, 54, 64);
            searchTextBox.BackColor = Color.FromArgb(255, 255, 255);
            comboBox1.ForeColor = Color.FromArgb(47, 54, 64);
            comboBox1.BackColor = Color.FromArgb(255, 255, 255);
            comboBox2.ForeColor = Color.FromArgb(47, 54, 64);
            comboBox2.BackColor = Color.FromArgb(255, 255, 255);
            contextMenuStrip1.ForeColor = Color.FromArgb(47, 54, 64);
            contextMenuStrip1.BackColor = Color.FromArgb(255, 255, 255);
            ConfigurationManager.AppSettings[key] = "dark";
        }
        else
        {
            btnChangeMode.Text = "Light Mode";
            this.ForeColor = Color.FromArgb(255, 255, 255);
            this.BackColor = Color.FromArgb(47, 54, 64);
            listView1.ForeColor = Color.FromArgb(255, 255, 255);
            listView1.BackColor = Color.FromArgb(47, 54, 64);
            listView2.ForeColor = Color.FromArgb(255, 255, 255);
            listView2.BackColor = Color.FromArgb(47, 54, 64);
            treeView1.ForeColor = Color.FromArgb(255, 255, 255);
            treeView1.BackColor = Color.FromArgb(47, 54, 64);
            filePathTextBox1.ForeColor = Color.FromArgb(255, 255, 255);
            filePathTextBox1.BackColor = Color.FromArgb(47, 54, 64);
            filePathTextBox2.ForeColor = Color.FromArgb(255, 255, 255);
            filePathTextBox2.BackColor = Color.FromArgb(47, 54, 64);
            searchTextBox.ForeColor = Color.FromArgb(255, 255, 255);
            searchTextBox.BackColor = Color.FromArgb(47, 54, 64);
            comboBox1.ForeColor = Color.FromArgb(255, 255, 255);
            comboBox1.BackColor = Color.FromArgb(47, 54, 64);
            comboBox2.ForeColor = Color.FromArgb(255, 255, 255);
            comboBox2.BackColor = Color.FromArgb(47, 54, 64);
            contextMenuStrip1.ForeColor = Color.FromArgb(255, 255, 255);
            contextMenuStrip1.BackColor = Color.FromArgb(47, 54, 64);
            ConfigurationManager.AppSettings[key] = "light";
        }
    }
    catch (Exception ex)
    {
        throw;
    }
}
```

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						34
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
}

```

Метод `renameMethod` відповідає за перейменування файлів або каталогів, вибраних у застосунку. Він перевіряє, який із двох елементів інтерфейсу перебуває у фокусі, і надає можливість редагування імені для вибраного елемента. Якщо в активному списку є вибраний елемент, увімкнено режим редагування, і виконується редагування імені.

Лістинг методу `renameMethod`:

```

private void renameMethod()
{
    if (firstInFocus)
    {
        if (listView1.SelectedItems.Count > 0)
        {
            ListViewItem selectedItem = listView1.SelectedItems[0];
            listView1.LabelEdit = true; // Дозволити редагування
            selectedItem.BeginEdit(); // Розпочати редагування
        }
    }
    else
    {
        if (listView2.SelectedItems.Count > 0)
        {
            ListViewItem selectedItem = listView2.SelectedItems[0];
            listView2.LabelEdit = true; // Дозволити редагування
            selectedItem.BeginEdit(); // Розпочати редагування
        }
    }
}

```

Метод `listView1_AfterLabelEdit` обробляє завершення перейменування у першому елементі інтерфейсу. Після редагування нове ім'я перевіряється на відсутність конфліктів. Якщо вибраний елемент є файлом або каталогом, перевіряється його існування, а також те, чи нове ім'я не порожнє. Далі визначається, чи є елемент файлом або директорією, і виконується відповідне. Якщо перейменування успішне, ім'я елемента в списку оновлюється. У разі помилки користувач отримує повідомлення з описом проблеми.

Лістинг методу `listView1_AfterLabelEdit`:

```

private void listView1_AfterLabelEdit(object sender, LabelEditEventArgs e)
{
    if (e.Label != null)
    {
        string oldName = listView1.Items[e.Item].Text;
        string newName = e.Label;
        string oldFilePath, newFilePath;

        if (File.Exists(Path.GetFullPath(oldName)))
        {
            oldFilePath = Path.GetFullPath(selectedItems1[0]);

```

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						35
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        newFilePath = newName;

        Console.WriteLine("rename" + newFilePath);
    }
    else
    {
        oldFilePath = addBackSlash() + oldName;
        newFilePath = addBackSlash() + newName;
    }
    try
    {
        FileAttributes fileAttr = File.GetAttributes(oldFilePath);
        if ((fileAttr & FileAttributes.Directory) == FileAttributes.Directory)
        {
            if (newName != "" && !Directory.Exists(newFilePath))
            {
                Directory.Move(oldFilePath, newFilePath);
            }
        }
        else
        {
            if (newName != "" && !File.Exists(newFilePath))
            {
                File.Move(oldFilePath, newFilePath);
            }
        }

        listView1.Items[e.Item].Text = newName; // Оновлюємо текст у ListView1
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Error renaming file: {ex.Message}");
    }
}
}

```

Метод deleteMethod відповідає за видалення файлів і каталогів, обраних користувачем. Спочатку визначається список обраних елементів залежно від того, яка панель активна. Користувачу пропонується підтвердити видалення через діалогове вікно. Якщо обрано один елемент, у повідомленні вказується його ім'я. Якщо елементів декілька, зазначається їхня кількість. Якщо користувач підтверджує видалення, процес виконується асинхронно в окремому потоці. Для кожного елемента зі списку визначається його повний шлях. Якщо елемент існує, перевіряється, чи є він файлом або каталогом. Після завершення всіх операцій інтерфейс оновлюється, щоб відобразити актуальний стан файлової системи. У разі виникнення помилки інформація про неї виводиться в консоль.

Лістинг методу deleteMethod:

```

private void deleteMethod()
{
    try
    {
        List<string> selectedItems = firstInFocus ? selectedItems1 : selectedItems2;
    }
}

```

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		36

```

        DialogResult dialogResult;
        if (selectedItems.Count <= 1)
            dialogResult = MessageBox.Show("Delete file: " + selectedItems[0] + "?",
"Warning", MessageBoxButtons.YesNo);
        else
            dialogResult = MessageBox.Show("Delete files " + selectedItems.Count +
"?", "Warning", MessageBoxButtons.YesNo);
        if (dialogResult == DialogResult.Yes)
        {
            ThreadPool.QueueUserWorkItem((o) =>
            {
                string delFilePath;

                foreach (string path in selectedItems)
                {
                    if (File.Exists(Path.GetFullPath(path)))
                        delFilePath = Path.GetFullPath(path);
                    else
                        delFilePath = addBackSlash() + path;

                    FileAttributes fileAttr = File.GetAttributes(delFilePath);
                    if ((fileAttr & FileAttributes.Directory) ==
FileAttributes.Directory)
                    {
                        Directory.Delete(delFilePath, true);
                    }
                    else
                    {
                        File.Delete(delFilePath);
                        loadAction1();
                        loadAction2();
                    }
                }
                loadAction1();
                loadAction2();
            }, null);
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

```

Метод pasteMethod реалізує вставку файлів і каталогів у вказане місце файлової системи, враховуючи можливість уникнення конфліктів імен шляхом перейменування. На початку визначається обраний для вставки елемент, залежно від того, який список (перший чи другий) перебуває у фокусі. Потім визначаються шляхи копіювання та вставки, причому шлях для вставки формується із поточного шляху та імені вибраного елемента. Метод запускає асинхронну операцію для обробки кожного елемента зі списку файлів або каталогів для копіювання. Якщо обраний елемент є файлом, перевіряються його атрибути, формується шлях для вставки, і за необхідності додається суфікс " — сору", поки не буде знайдено унікальне ім'я. Після цього викликається метод

CopyFileAsync для копіювання файлу. Якщо виконується операція вирізання, оригінальний файл видаляється. Якщо елемент є каталогом, аналогічно перевіряється його наявність у цільовому місці. У разі збігу імен до назви каталогу додається суфікс " — copy". Для копіювання каталогу використовується метод CopyFoldereAsync, а якщо операція передбачає переміщення, оригінальний каталог видаляється після копіювання. Після завершення копіювання оновлюються обидва елементи користувацького інтерфейсу, щоб відобразити зміни. У випадку помилки виконання виводиться повідомлення, а прогрес-форма закривається. Прапорець, що вказує на виконання операції вирізання, скидається.

Лістинг методу pasteMethod:

```
private async Task pasteMethod(ProgressForm pf)
{
    try
    {
        var tokenSource = new CancellationTokenSource();
        string selectedItem = (selectedItems1.Count == 0 && selectedItems2.Count == 0)
? "" :
        selectedItem = firstInFocus ? selectedItems1[0] : selectedItems2[0];

        Console.Out.WriteLine(selectedItem);

        pf.copyPath = copyItems[0].Substring(0, copyItems[0].LastIndexOf("\\"));
        pf.pastePath = addBackSlash() + selectedItem;
        selectedItem += "\\";

        await Task.Run(async () =>
        {
            foreach (string copyPath in copyItems)
            {
                if (File.Exists(copyPath) || Directory.Exists(copyPath))
                {
                    tokenSources.Add(tokenSource);
                    Console.WriteLine("Розмір списку після додавання токена: " +
tokenSources.Count);

                    FileInfo fileDetails = new FileInfo(copyPath);
                    string name = fileDetails.Name;
                    string ext = fileDetails.Extension;
                    string pastePath;

                    if ((File.GetAttributes(addBackSlash() + selectedItem) &
FileAttributes.Directory) == FileAttributes.Directory)
                        pastePath = addBackSlash() + selectedItem + name;
                    else
                        pastePath = addBackSlash() + name;

                    FileAttributes fileAttr = File.GetAttributes(copyPath);

                    if ((fileAttr & FileAttributes.Directory) ==
FileAttributes.Directory)
```

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						38
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        {
            Console.WriteLine(copyPath + " " + pastePath);
            while (Directory.Exists(pastePath))
            {
                pastePath += " - copy";
            }

            await CopyFoldereAsync(pf, copyPath, pastePath,
tokenSource.Token);

            if (cut == true)
            {
                Directory.Delete(copyPath, true);
            }
        }
        else
        {
            while (File.Exists(pastePath))
            {
                pastePath = pastePath.Substring(0,
pastePath.LastIndexOf(".")) + " - copy" + ext;
            }
            await CopyFileAsync(pf, copyPath, pastePath,
tokenSource.Token);
            if (cut == true)
            {
                File.Delete(copyPath);
            }
        }
    }
    }
    Invoke((Action)(() => loadListView1()));
    Invoke((Action)(() => loadListView2()));
});
Invoke((Action)(() => pf?.Close()));
cut = false;
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
    Invoke((Action)(() => pf?.Close()));
}
}

```

Метод CopyFolderAsync забезпечує асинхронне копіювання вмісту каталогу, включаючи всі підкаталоги та файли, у нове місце, використовуючи буфер заданого розміру та підтримуючи можливість скасування операції через токен. Спочатку створюється цільовий каталог, якщо він не існує. Далі отримується список усіх підкаталогів у вихідному каталозі, і для кожного з них створюється відповідний каталог у цільовому місці. Після цього формується список файлів у вихідному каталозі, включаючи файли з усіх підкаталогів, і для кожного з них викликається метод CopyFileAsync.

Лістинг методу CopyFolderAsync:

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

```
private static async Task CopyFoldereAsync(ProgressForm pf, string copyPath, string
pastePath, CancellationToken token, int bufferSize = 81920)
{
    Directory.CreateDirectory(pastePath);

    foreach (string dirPath in Directory.GetDirectories(copyPath, "*",
SearchOption.AllDirectories))
    {
        Directory.CreateDirectory(dirPath.Replace(copyPath, pastePath));
    }

    foreach (string newPath in Directory.GetFiles(copyPath, "*.*",
SearchOption.AllDirectories))
    {
        await CopyFileAsync(pf, newPath, newPath.Replace(copyPath, pastePath), token);
    }
}
```

Метод CopyFileAsync відповідає за асинхронне копіювання окремого файлу. Він відкриває потоки для читання вихідного файлу та запису в цільовий файл. У процесі копіювання створюється буфер, через який дані порціями читаються з вихідного файлу та записуються в цільовий. Під час кожної ітерації метод оновлює прогрес у формі, використовуючи делегат, щоб відобразити обсяг переданих даних. Також виконується перевірка токена на скасування операції. Якщо копіювання завершується успішно, файл у новому місці повністю відповідає оригіналу за вмістом.

Лістинг методу CopyFileAsync:

```
private static async Task CopyFileAsync(ProgressForm pf, string sourceFile, string
destinationFile, CancellationToken token, int bufferSize = 81920)
{
    using (FileStream sourceStream = new FileStream(sourceFile, FileMode.Open,
FileAccess.Read))
    using (FileStream destinationStream = new FileStream(destinationFile,
FileMode.Create, FileAccess.Write))
    {
        byte[] buffer = new byte[bufferSize];
        int bytesRead;

        while ((bytesRead = await sourceStream.ReadAsync(buffer, 0, buffer.Length,
token)) > 0)
        {
            try
            {
                pf.Invoke((Action)(() => pf.UpdateProgressBar(bufferSize/1000)));
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            await destinationStream.WriteAsync(buffer, 0, bytesRead, token);
            token.ThrowIfCancellationRequested();
        }
    }
}
```

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

Метод `storageMemoryCount` обчислює загальний обсяг пам'яті, зайнятої вибраними файлами та каталогами. Спочатку визначається, який набір елементів обрано для обчислення: це може бути один з двох списків вибраних елементів, залежно від стану змінної `firstInFocus`. Далі для кожного елемента в списку обчислюється його розмір. Якщо елемент є каталогом, то для кожного файлу в цьому каталозі (та підкаталогах) визначається його розмір і додається до загальної суми, яка підсумовує всі розміри файлів. Для кожного каталогу виводиться його обсяг пам'яті у форматі, що показує кількість кілобайтів. Якщо ж елемент є файлом, то його розмір просто додається до загальної суми пам'яті. Після обчислення всіх елементів оновлюється текстове поле, що показує загальний обсяг пам'яті, зайнятої вибраними файлами. У разі виникнення помилки при доступі до каталогу виводиться повідомлення "no access". Після завершення обчислення для всіх елементів результат відображається на інтерфейсі користувача.

Лістинг методу `storageMemoryCount`:

```
private void storageMemoryCount()
{
    try
    {
        long selectedSpace = 0;
        long eachSpace = 0;
        string subOutput;
        int amount = 0;
        List<string> selectedItems = firstInFocus ? selectedItems1 : selectedItems2;

        foreach (string item in selectedItems)
        {
            FileInfo fileDetails = new FileInfo(addBackSlash() + item);
            if ((fileDetails.Attributes & FileAttributes.Directory) ==
                FileAttributes.Directory)
            {
                DirectoryInfo dirDetails = new DirectoryInfo(addBackSlash() + item);
                try
                {
                    foreach (FileInfo f in dirDetails.GetFiles("*.\"",
                        SearchOption.AllDirectories))
                    {
                        selectedSpace += f.Length;
                        eachSpace += f.Length;
                        amount++;
                    }
                    subOutput = $"{eachSpace / 1000:n0} kb";
                    eachSpace = 0;
                }
                catch (Exception ex)
                {
                    subOutput = "no access";
                }
            }
        }
    }
}
```

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						41
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        Invoke((MethodInvoker)delegate
        {
            if(firstInFocus)
                listView1.FindItemWithText(dirDetails.Name).SubItems[3].Text =
subOutput;
            else
                listView2.FindItemWithText(dirDetails.Name).SubItems[3].Text =
subOutput;
        });
    }
    else
    {
        selectedSpace += fileDetails.Length;
        amount++;
    }
    Invoke((MethodInvoker)delegate
    {
        selectedItemsLabel.Text = $"{selectedSpace / 1000:n0} kb in {amount}
files";
    });
}
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
}

```

Розглянемо систему обміну файлами по локальній мережі.

Спочатку перевіряється доступність мережі. Якщо мережа підключена, метод отримує IP-адреси поточного комп'ютера і знаходить відповідну адресу. Знайдена IP-адреса виводиться в консоль і відображається в призначеному для цього елементі інтерфейсу. Далі IP-адреса розбивається на частини для визначення діапазону адрес у локальній мережі. Для кожної адреси в діапазоні від 10 до 254 створюється об'єкт Ping, і виконується асинхронний запит через `ping.SendAsync()`, за винятком IP-адреси поточного комп'ютера. Обробка результатів здійснюється в обробнику події `pingCompletedEvent`. Якщо мережа доступна, змінюється колір тексту та оновлюється текст мітки, щоб показати, що комп'ютер підключений до мережі.

Лістинг методу `searchPC`:

```

void searchPC()
{
    bool isNetworkUp = NetworkInterface.GetIsNetworkAvailable();
    if (isNetworkUp)
    {
        var host = Dns.GetHostEntry(Dns.GetHostName());
        foreach (var ip in host.AddressList)
        {
            if (ip.AddressFamily == AddressFamily.InterNetwork)
            {
                this.IP = ip.ToString();
            }
        }
    }
}

```

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		42

```

        Console.WriteLine(this.IP);
    }
}
Invoke((MethodInvoker)delegate
{
    infoLabel.Text = "This Computer: " + this.IP;
});
string[] ipRange = IP.Split('.');
for (int i = 100; i < 255; i++)
{
    Ping ping = new Ping();
    string testIP = ipRange[0] + '.' + ipRange[1] + '.' + ipRange[2] + '.' +
i.ToString();
    if (testIP != this.IP)
    {
        ping.PingCompleted += new
PingCompletedEventHandler(pingCompletedEvent);
        ping.SendAsync(testIP, 100, testIP);
    }
}

Invoke((MethodInvoker)delegate
{
    notificationLabel.ForeColor = Color.Green;
    notificationLabel.Text = "Online";
});

if (!serverRunning)
    startServer();
}
else
{
    Invoke((MethodInvoker)delegate
    {
        notificationLabel.ForeColor = Color.Red;
        notificationLabel.Text = "Offline";
    });
    MessageBox.Show("No connection to LAN");
}
}

```

Метод serverTasks відповідає за обробку вхідних даних, що надходять через TCP-з'єднання, для прийому файлів. Він виконується в нескінченному циклі, постійно перевіряючи наявність нових файлів і обробляючи з'єднання з клієнтами. На початку кожної ітерації перевіряється, чи був отриманий файл. Якщо файл надійшов, відображається запит на його збереження. Якщо користувач відмовляється від збереження, файл видаляється, а прапорець скидається. Далі метод приймає нове TCP-з'єднання і змінює текст мітки, щоб вказати, що файл надходить від конкретного відправника. Потім відкривається потік для читання даних від клієнта.

Лістинг методу serverTasks:

```

void serverTasks()
{
    try

```

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

```

{
    while (true)
    {
        if (fileReceived == 1)
        {
            if (MessageBox.Show("Save File?", "File received",
                MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.No)
            {
                File.Delete(savePath);
                fileReceived = 0;
            }
            else
            {
                fileReceived = 0;
            }
        }

        tcpclient = tcplistener.AcceptTcpClient();
        Invoke((MethodInvoker)delegate
        {
            fileNotificationLabel.Text = "File Coming from " + senderIP + " " +
senderName;
        });
        isConnected = true;
        NetworkStream stream = tcpclient.GetStream();
        if (flag == 1 && isConnected)
        {
            savePath = savePathLabel.Text + "\\\" + fileName;
            using (var output = File.Create(savePath))
            {
                var buffer = new byte[1024];
                int bytesRead;
                while ((bytesRead = stream.Read(buffer, 0, buffer.Length)) > 0)
                {
                    output.Write(buffer, 0, bytesRead);
                }
                flag = 0;
                tcpclient.Close();
                isConnected = false;
                fileName = "";
                Invoke((MethodInvoker)delegate
                {
                    fileNotificationLabel.Text = "";
                });
                fileReceived = 1;
            }
        }
        else if (flag == 0 && isConnected)
        {
            Byte[] bytes = new Byte[256];
            String data = null;
            int i;
            while ((i = stream.Read(bytes, 0, bytes.Length)) != 0)
            {
                data = System.Text.Encoding.ASCII.GetString(bytes, 0, i);
            }
            string[] msg = data.Split('@');
            fileName = msg[0];
            senderIP = msg[1];
            senderName = msg[2];
            tcpclient.Close();
            isConnected = false;
            flag = 1;
        }
    }
}

```

```

    }
    catch (Exception)
    {
        flag = 0;
        isConnected = false;
        if (tcpclient != null)
            tcpclient.Close();
    }
}

```

Обробка події натискання кнопки `sendFileButton_Click` відповідає за відправку файлу на інший комп'ютер у локальній мережі. Процес починається, якщо вибраний комп'ютер зі списку комп'ютерів в мережі або введена IP-адреса в полі `ipBox`, сервер запущений і вибрано файл для відправки. У разі введення IP-адреси в поле `ipBox`, вона присвоюється змінній `receiverIP`, а змінна `receiverName` залишається порожньою. Якщо ж обраний комп'ютер зі списку, то `receiverIP` отримує IP-адресу цього комп'ютера, а `receiverName` — його ім'я. Метод пінгує цільовий комп'ютер. Якщо пінг не вдається, виводиться повідомлення про недоступність комп'ютера. Якщо пінг пройшов успішно, запускається новий потік для відображення повідомлення про початок відправки файлу. Після цього сервер зупиняється, поточний серверний потік завершується, і мітка `serverRunning` змінюється на `false`. Створюється новий сокет, і встановлюється з'єднання з цільовим комп'ютером на порті 11000. Спочатку передаються дані про ім'я файлу, IP-адресу відправника та ім'я машини. Після цього сокет закривається, і створюється новий для передачі самого файлу. По завершенні відправки виводиться повідомлення про успішну передачу файлу.

Лістинг методу `sendFileButton_Click`:

```

private void sendFileButton_Click(object sender, EventArgs e)
{
    receiverIP = null;
    receiverName = null;
    if ((onlinePCList.SelectedIndices.Count > 0 || ipBox.Text != "") && serverRunning
    && fileNameLabel.Text != ".")
    {
        if (ipBox.Text != "")
        {
            receiverIP = ipBox.Text;
            receiverName = "";
        }
        else
        {
            receiverIP = onlinePCList.SelectedItem[0].Text;
            receiverName = onlinePCList.SelectedItem[0].SubItems[1].Text;
        }
        try
        {

```

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						45
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        Ping p = new Ping();
        PingReply r;
        r = p.Send(receiverIP);
        if (!(r.Status == IPStatus.Success))
        {
            MessageBox.Show("Target computer is not available.", "",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        else
        {
            notification = new Thread(new ThreadStart(showNotification));
            notification.Start();
            fileNotificationLabel.Text = "Please don't do other tasks. File
sending to " + receiverIP + " " + receiverName + "...";
            tcpListener.Stop();
            serverThread.Abort();
            serverThread.Join();
            serverRunning = false;
            clientSocket = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
            clientSocket.Connect(new IPEndPoint(IPAddress.Parse(receiverIP),
                11000));

            string fileName = fileNameLabel.Tag.ToString();
            byte[] fileNameData = Encoding.Default.GetBytes(fileName + "@" +
this.IP + "@" + Environment.MachineName);
            clientSocket.Send(fileNameData);
            clientSocket.Shutdown(SocketShutdown.Both);
            clientSocket.Close();
            clientSocket = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
            clientSocket.Connect(new IPEndPoint(IPAddress.Parse(receiverIP),
                11000));

            clientSocket.SendFile(fileNameLabel.Text);
            clientSocket.Shutdown(SocketShutdown.Both);
            clientSocket.Close();
            Invoke((MethodInvoker)delegate
            {

            });
            MessageBox.Show("File sent to " + receiverIP + " " + receiverName,
                "Confirmation", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        if (clientSocket != null)
        {
            clientSocket.Shutdown(SocketShutdown.Both);
            clientSocket.Close();
        }
    }
    finally
    {
        for (int i = 0; i < onlinePCList.SelectedIndices.Count; i++)
        {
            onlinePCList.Items[this.onlinePCList.SelectedIndices[i]].Selected =
false;
        }
        fileNotificationLabel.Text = ".";
        startServer();
    }
}
}
}

```

Висновки до другого розділу

В другому розділі було проведено детальний аналіз усіх можливих варіантів використання програмного продукту відповідно до вимог до програмного забезпечення. На основі зібраних даних була спроектована структура застосунку, враховуючи модель компонентів та їх взаємодії.

Було розроблено алгоритм роботи з додатком та описано послідовність його використання. Основний функціонал включає можливості навігації по файловій системі, перегляду та керування файлами і каталогами, а також операції, такі як перейменування, видалення, копіювання і вставка файлів і папок.

Окремо була розглянута взаємодія користувача з інтерфейсом додатку, включаючи обробку подій при взаємодії з елементами інтерфейсу, зокрема вибір файлів чи папок, натискання кнопок та робота з контекстним меню.

В результаті проведеного аналізу та реалізації функціоналу був створений зручний і інтуїтивно зрозумілий застосунок для управління та обміну файлами, що відповідає всім вимогам і забезпечує ефективну роботу з файловою системою.

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ТЕСТУВАННЯ ТА ПОРЯДОК РОБОТИ ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ТА ОБМІНУ ФАЙЛАМИ

3.1 Порядок встановлення та налаштування параметрів

Встановлення та налаштування платформи вимагає розгортання всіх його компонентів та налаштування як кожного елементу окремо, так і загальної системи в цілому.

Принцип роботи взаємодії компонентів програми було описано в діаграмі розгортання (рис. 3.1).

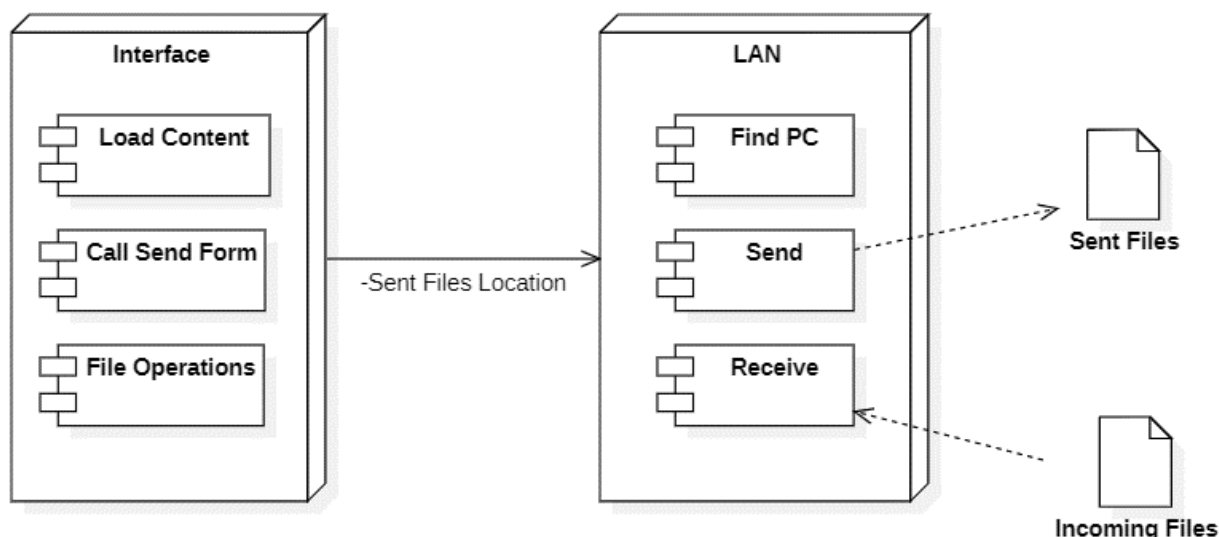


Рис. 3.1 Діаграма розгортання

Діаграма розгортання показує взаємодію між інтерфейсом користувача та локальною мережею (LAN) для передачі файлів. Вона складається з двох основних компонентів: Інтерфейс та LAN, які спільно здійснюють обробку файлів.

Компонент Interface забезпечує завантаження вмісту, включаючи список файлів і папок, доступних користувачеві. Він також дозволяє ініціювати форму для відправки файлів, що надає можливість обирати файли та передавати їх через мережу. Крім того, цей компонент виконує операції з файлами, такі як

копіювання, переміщення, перейменування та видалення на комп'ютері користувача.

Компонент LAN відповідає за пошук доступних комп'ютерів у локальній мережі для встановлення з'єднання з метою обміну файлами. Він забезпечує надсилання файлів до вибраного комп'ютера в мережі, а також отримання файлів, що надходять від інших пристроїв у локальній мережі.

Інтерфейс надсилає інформацію про місцезнаходження файлів, які потрібно відправити, до модуля LAN. Це дозволяє модулю LAN обробляти надсилання файлів через локальну мережу.

3.2 Інтерфейс та принцип роботи з файлами

Інтерфейс застосунку складається з кількох функціональних областей, що дозволяють користувачу ефективно працювати з файлами та папками. У верхній частині розташована панель інструментів із кнопками для виконання основних операцій, таких як відкриття, перейменування, видалення, копіювання, вирізання, вставка файлів, а також відправка файлів через локальну мережу та копіювання шляху до обраного об'єкта. Під панеллю інструментів розташовані текстові поля, що відображають поточний шлях для кожної з панелей, а також кнопки для переходу вперед, назад або до кореневої директорії.

Зліва знаходиться панель для навігації між дисками, яка дозволяє швидко перемикатися між доступними розділами файлової системи, наприклад. Центральну частину інтерфейсу займають дві області для перегляду вмісту каталогів, що дає змогу одночасно працювати з двома різними шляхами. У цих областях відображаються імена файлів і папок, дати їх останніх змін, типи та розміри. Над областями розташована інформація про використання пам'яті на поточному диску, включно із зайнятим і вільним місцем, а також загальним обсягом.

Справа зверху є поле пошуку, яке дозволяє користувачу швидко знаходити файли та папки в обраній директорії. У правому нижньому куті є кнопка перемикавання теми оформлення, наприклад, на темний режим.

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						49
Змн.	Арк.	№ докум.	Підпис	Дата		

Інтерфейс орієнтований на забезпечення зручності та інтуїтивної зрозумілості, спрощуючи виконання таких операцій, як копіювання, переміщення, перейменування, видалення та навігація по файловій системі.

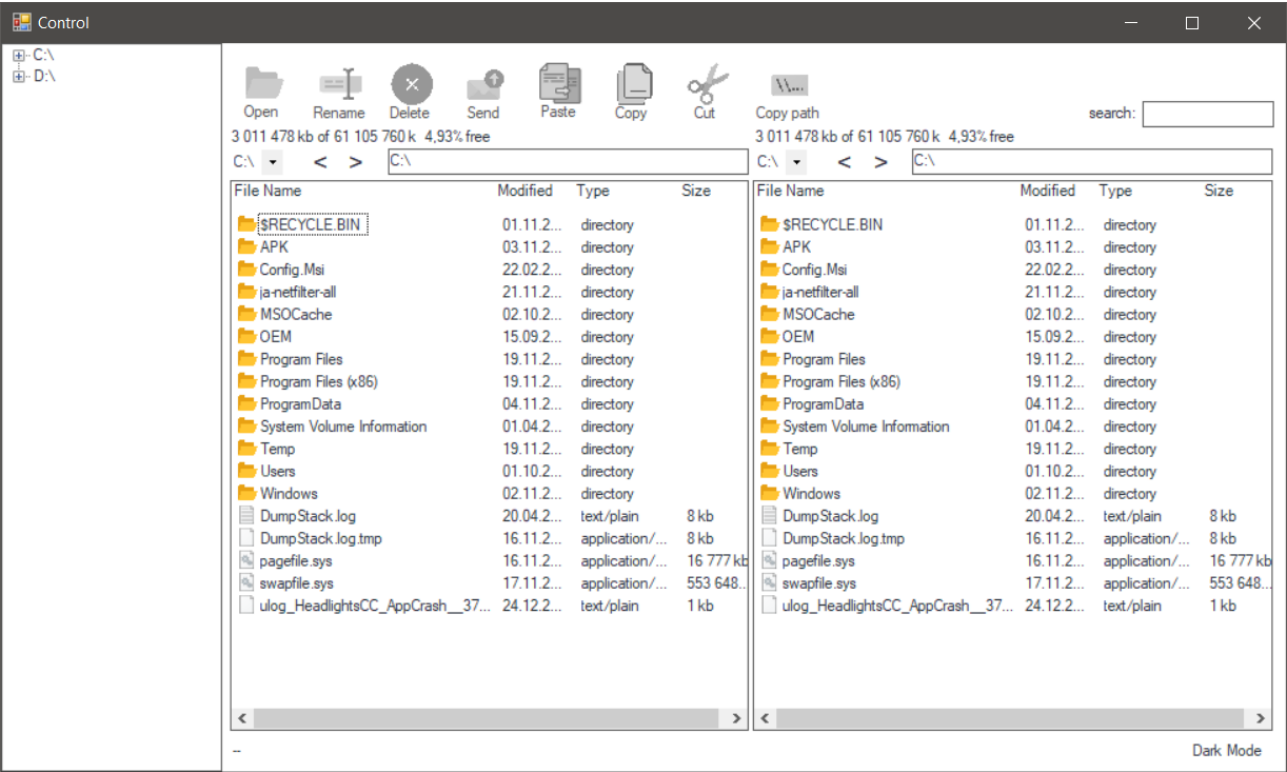


Рис. 3.2 Головне вікно

Коли користувач вводить шлях у текстовий рядок адреси, програма розпочинає обробку введених даних. Після натискання відповідної кнопки, система перевіряє існування вказаного шляху в файловій системі. Якщо шлях правильний, програма завантажує вміст відповідної директорії, відображаючи список файлів і папок у головному вікні із зазначенням їхніх параметрів, таких як ім'я, дата модифікації, тип і розмір. У разі некоректного шляху користувач отримує повідомлення про помилку, а текстовий рядок залишається доступним для внесення виправлень.

Після успішного переходу текстовий рядок оновлюється, щоб відображати поточний шлях. Одночасно синхронізується панель файлового дерева, і активна директорія виділяється для полегшення орієнтації. Цей процес забезпечує зручність і швидкість переходу між папками та файлами в системі, спрощуючи управління даними. (Рис. 3.3).

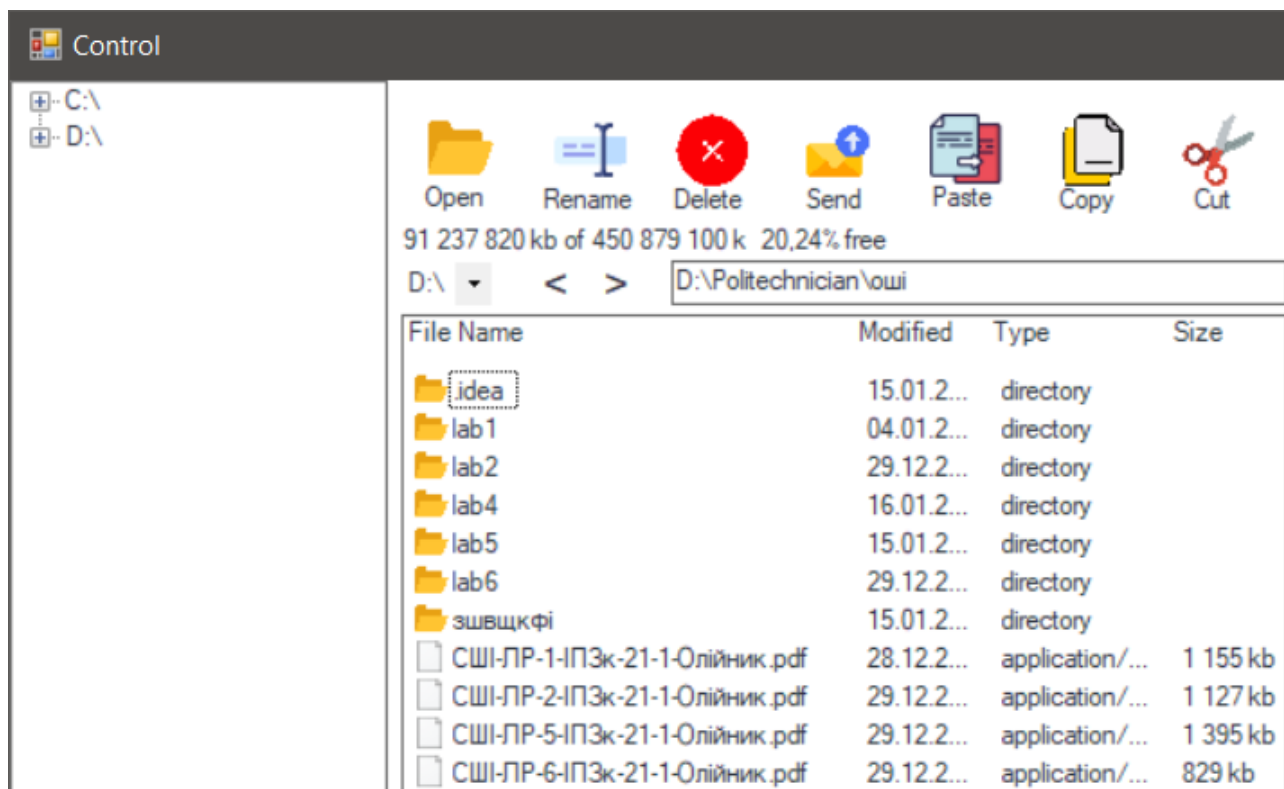


Рис. 3.3 Головне вікно

Коли користувач виконує подвійне натискання лівою кнопкою миші на елемент застосунку, починається взаємодія між програмою та операційною системою. Система реєструє два послідовні кліки, що відповідають встановленому інтервалу подвійного клацання. Після цього файлова програма визначає, який саме елемент було обрано: файл чи папку.

Якщо це папка, програмне забезпечення спочатку змінює шлях у полі адреси, відображаючи нову директорію, наприклад, з "D:\" на "D:\Util". Далі файловий застосунок запитує інформацію про вміст цієї папки у файлової системи, отримуючи список файлів і папок, які в ній зберігаються. Цей список завантажується в робочу область вікна та відображається у вигляді значків, таблиць чи списків залежно від вибраного режиму перегляду.

Якщо ж натиснуто на файл, операційна система аналізує його тип на основі розширення, наприклад, ".exe" чи ".txt". Вона звертається до асоційованої програми, яка може відкривати цей тип файлів. Потім система запускає відповідну програму, передаючи їй шлях до файлу, щоб забезпечити доступ до його вмісту. Наприклад, якщо це текстовий файл, відкривається текстовий

редактор, а якщо це виконуваний файл, він запускається як окремий процес. Візуально для користувача зміни відбуваються плавно: інтерфейс оновлюється, а вміст нової папки або файлу стає доступним для взаємодії. (Рис. 3.4 – 3.5).

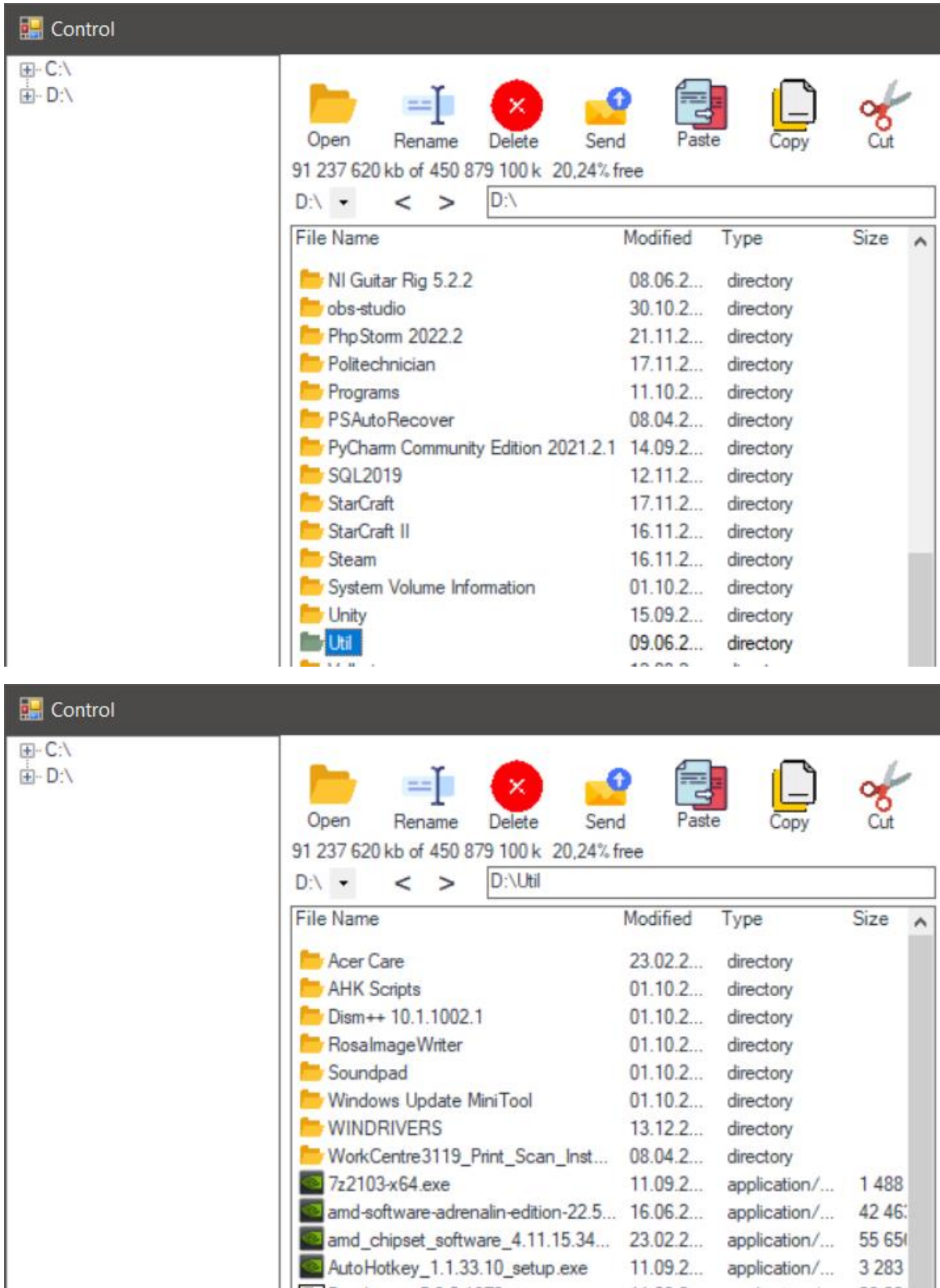


Рис. 3.4 – 3.5 Вміст каталогу

Коли користувач натискає правою кнопкою миші на елемент у застосунку, операційна система відповідно до місця натискання, програма ініціює виклик контекстного меню. Система визначає, на який саме об'єкт вказував курсор під час натискання — це може бути файл, папка або порожній простір вікна.

Контекстне меню має набір доступних команд для цього об'єкта, таких як відкриття, перейменування, копіювання, видалення тощо. Залежно від контексту, набір команд може змінюватися: наприклад, для файлу буде доступна команда перейменування, а для порожнього простору — команди для створення нових елементів.

Після цього відбувається візуальне відображення меню у вигляді списку опцій, яке розташовується біля місця натискання. Програма також активує механізм взаємодії, дозволяючи користувачеві вибрати одну з команд у меню за допомогою курсора. Як тільки користувач наводить курсор на команду, вона підсвічується, сигналізуючи про готовність до виконання. (Рис. 3.6).

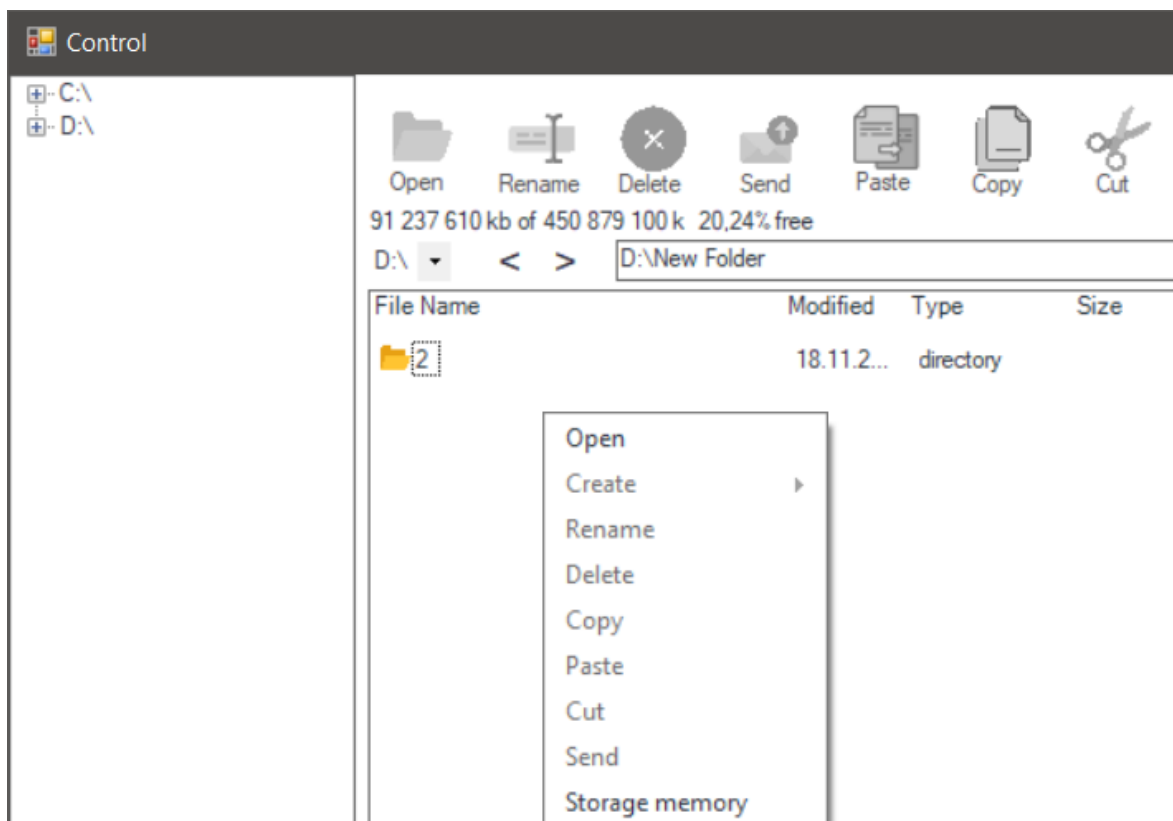


Рис. 3.6 Контекстне меню

Процес створення файлу або папки за допомогою контекстного меню починається з того, що користувач викликає меню, натискаючи правою кнопкою

миші на порожньому просторі вікна застосунку або в обраній директорії. Після цього операційна система реєструє натискання і визначає область вікна, на якій відбулося клацання. Застосунок містить набір доступних дій для цього контексту, таких як створення нового елемента.

Коли контекстне меню з'являється на екрані, користувач наводить курсор миші на опцію створення. При наведенні відкривається вкладене підменю, яке пропонує вибір типу елемента для створення: "Папка" або "Файл". Користувач вибирає потрібний варіант, наприклад, "Папка".

Після вибору команди "Папка" застосунок звертається до файлової системи з командою створити новий елемент у поточній директорії. Одразу після цього у робочій області з'являється папка з типовою назвою "New Folder". (Рис. 3.7 – 3.8).

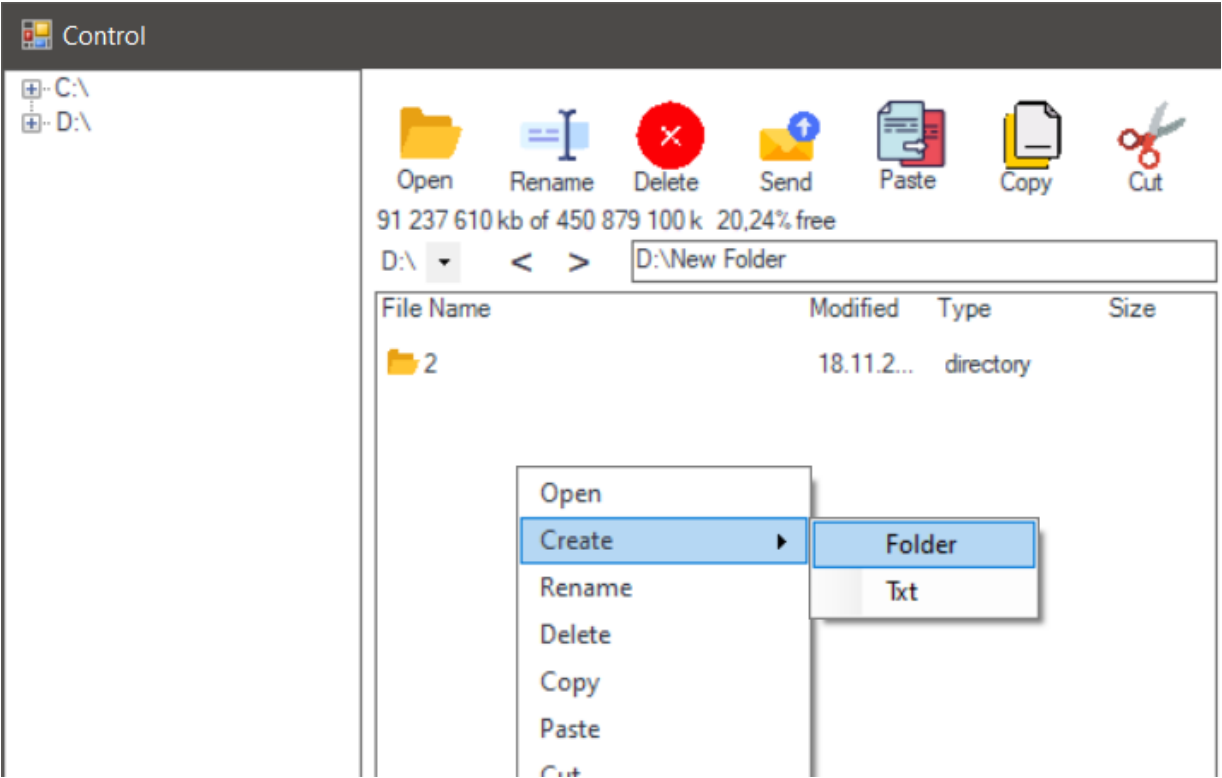


Рис. 3.7 Контекстне меню

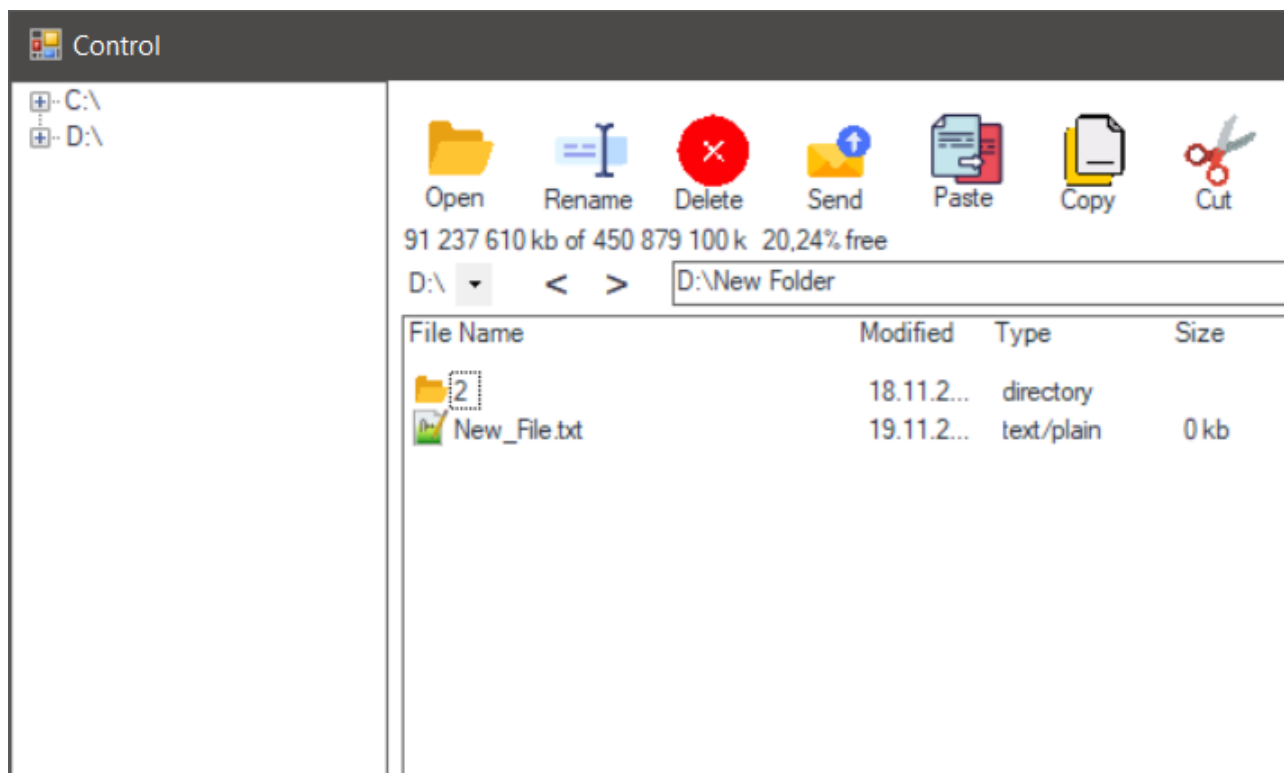


Рис. 3.8 Створення об'єктів.

Процес переміщення та копіювання файлу у застосунку включає декілька етапів і взаємодій між інтерфейсом користувача, файловою системою та програмним забезпеченням. Користувач починає із виділення потрібного файлу або папки у вікні застосунку. Після цього необхідно натиснути команду "Копіювати" або "Вирізати" за допомогою верхніх іконок або контекстного меню. Якщо натиснуто "Копіювати", система готує файл для копіювання, якщо "Вирізати", файл позначається для переміщення. (Рис. 3.9 – 3.10).

Потім користувач переходить до цільової директорії, використовуючи текстове поле шляху або візуальний список директорій. У цільовій директорії виконується команда вставки, після чого застосунок ініціює операцію копіювання або переміщення.

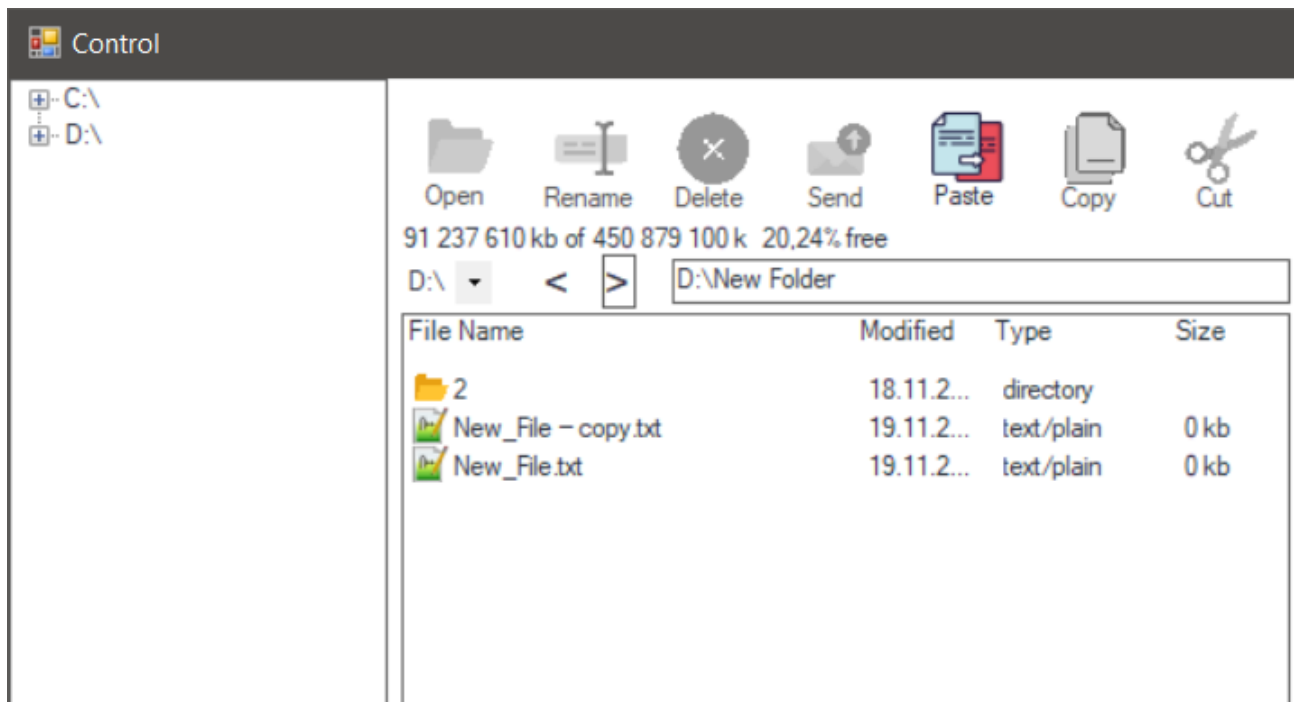


Рис. 3.9 Копіювання файлу.

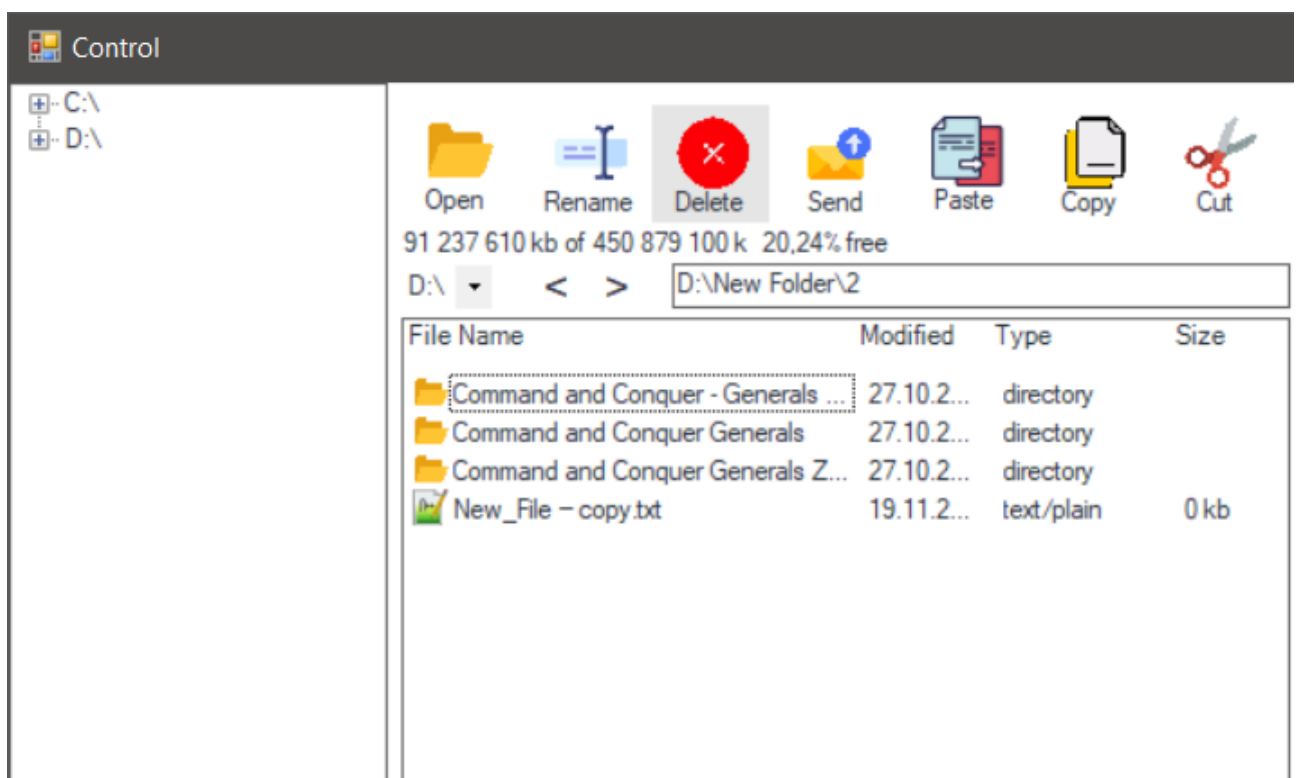


Рис. 3.10 Переміщення файлу.

На екрані з'являється прогрес-бар, який візуально відображає перебіг операції. У цьому ж вікні присутня кнопка зупинки, яка дозволяє користувачеві припинити операцію на будь-якому етапі. (Рис. 3.11).

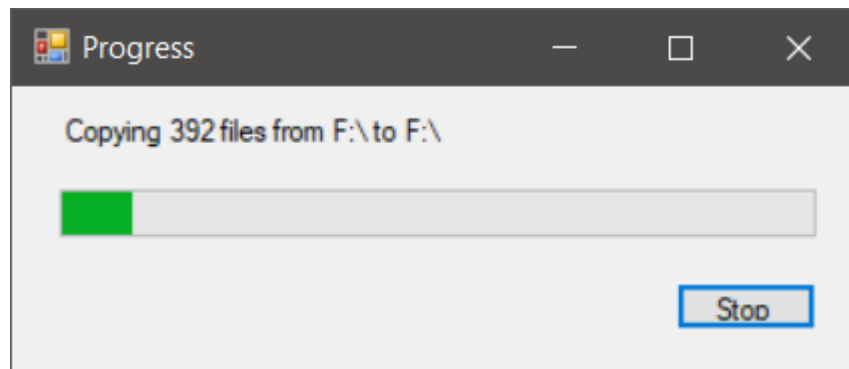


Рис. 3.11 Вікно прогресу.

Процес видалення файлу розпочинається з того, що користувач виділяє потрібний файл або папку у робочій області застосунку. Після цього він натискає на кнопку видалення у верхній панелі інструментів або вибирає опцію видалення у контекстному меню, яке відкривається при натисканні правою кнопкою миші на об'єкті.

Застосунок відображає діалогове вікно підтвердження, щоб уникнути випадкового видалення. У цьому вікні вказується ім'я файлу чи папки та пропонується підтвердити дію або скасувати дію. Якщо користувач підтверджує видалення, операція продовжується. (Рис. 3.12).

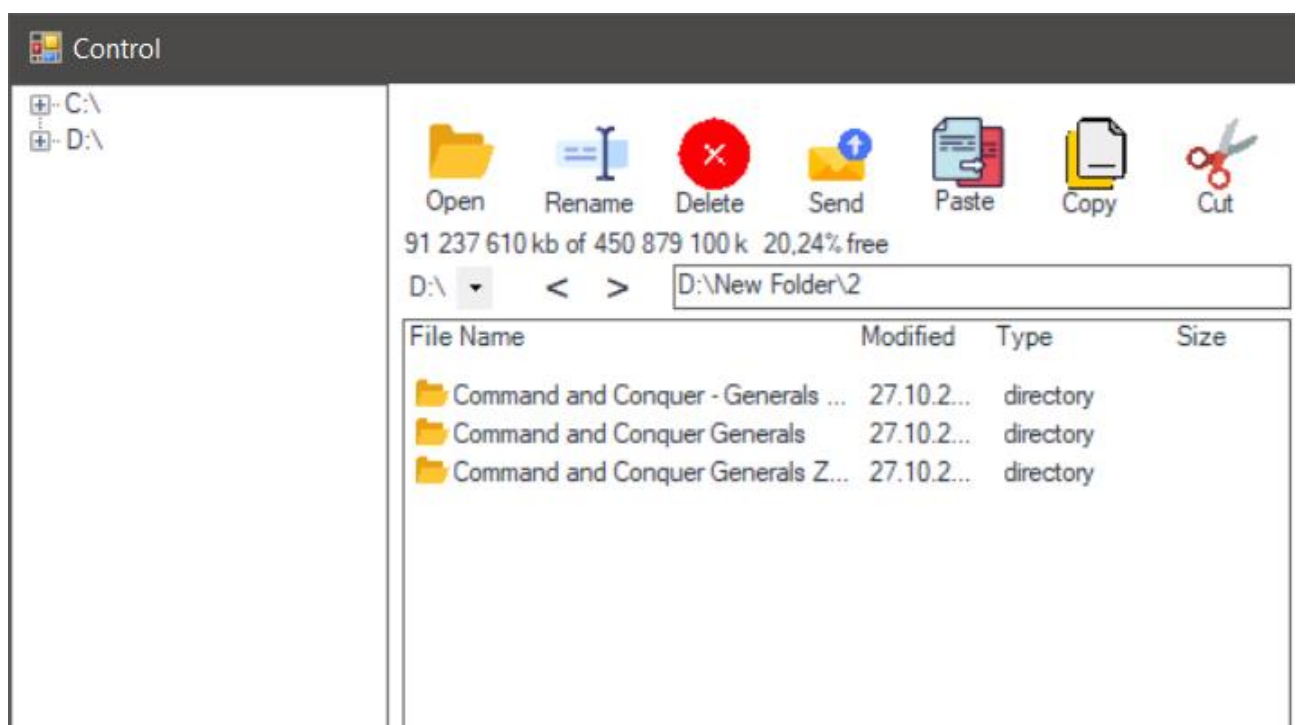


Рис. 3.12 Видалення файлу.

Процес перейменування файлу починається з того, що користувач виділяє потрібний файл або папку у робочій області застосунку. Далі є два способи

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						57
Змн.	Арк.	№ докум.	Підпис	Дата		

ініціювати перейменування: або натиснувши кнопку перейменування на верхній панелі інструментів, або вибравши опцію перейменування у контекстному меню, яке відкривається при натисканні правою кнопкою миші на об'єкті.

Після вибору команди назва файлу або папки стає активною для редагування. Користувач може безпосередньо редагувати текст, наприклад, змінити назву, додати символи або прибрати частину тексту. При цьому розширення файлу, якщо воно відображається, також може бути змінено, що може вплинути на тип файлу. (Рис. 3.12).

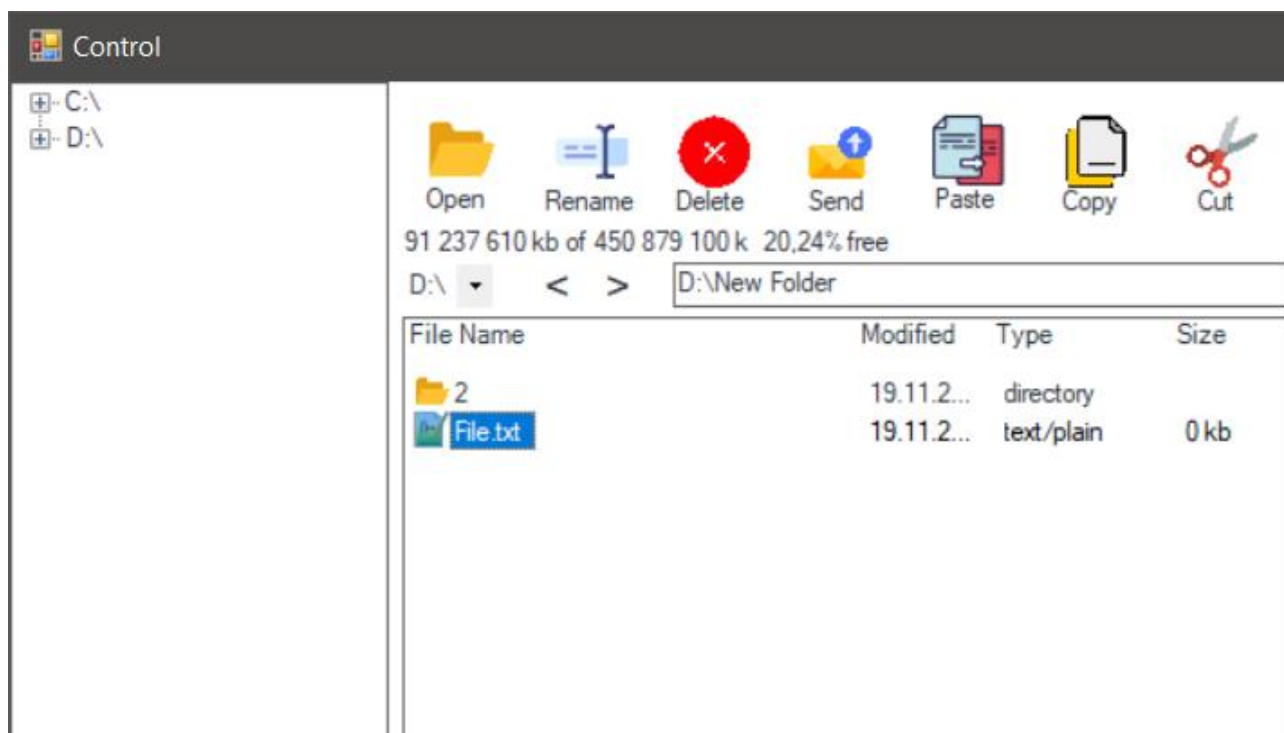


Рис. 3.13 Перейменування файлу.

Процес пошуку файлу або папки розпочинається з того, що користувач вводить ключове слово до текстового поля пошуку, яке розташоване у верхній частині вікна застосунку. Користувач вводить ключове слово, повну або часткову назву файлу чи папки, яку він хоче знайти. Після введення тексту пошук почнеться після натискання клавіші Enter на клавіатурі, у вибраному у випадіючому списку розділі.

Результати пошуку відображаються у вигляді списку в активній на даний момент області застосунку. Користувач може взаємодіяти з результатами, як зі звичайними файлами, — відкривати, копіювати, переміщувати, видаляти тощо.

Процес завершується, коли користувач виконує взаємодію яке призведе до відображення вмісту конкретного розділу або папки. (Рис. 3.14 – 3.15).

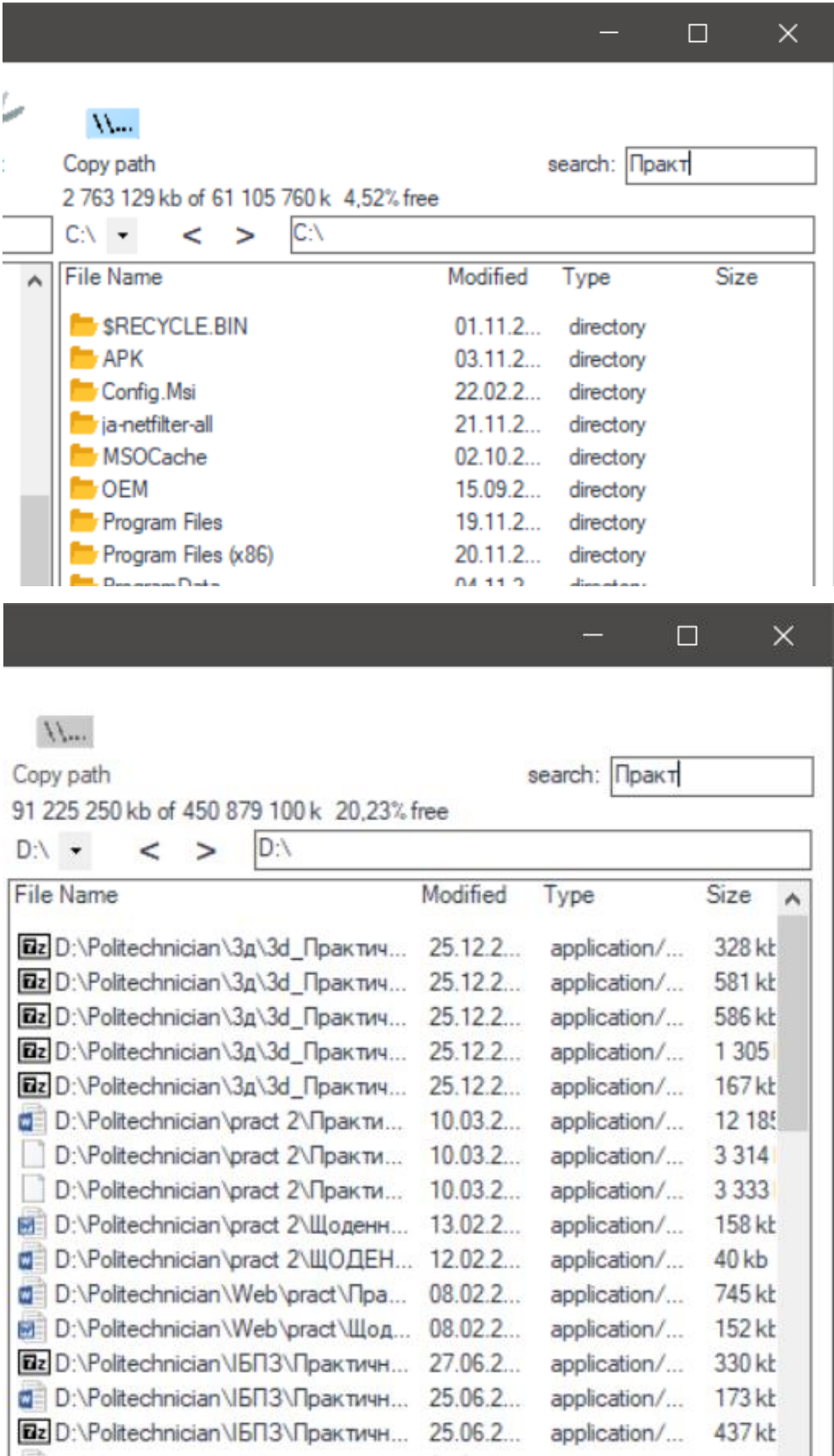
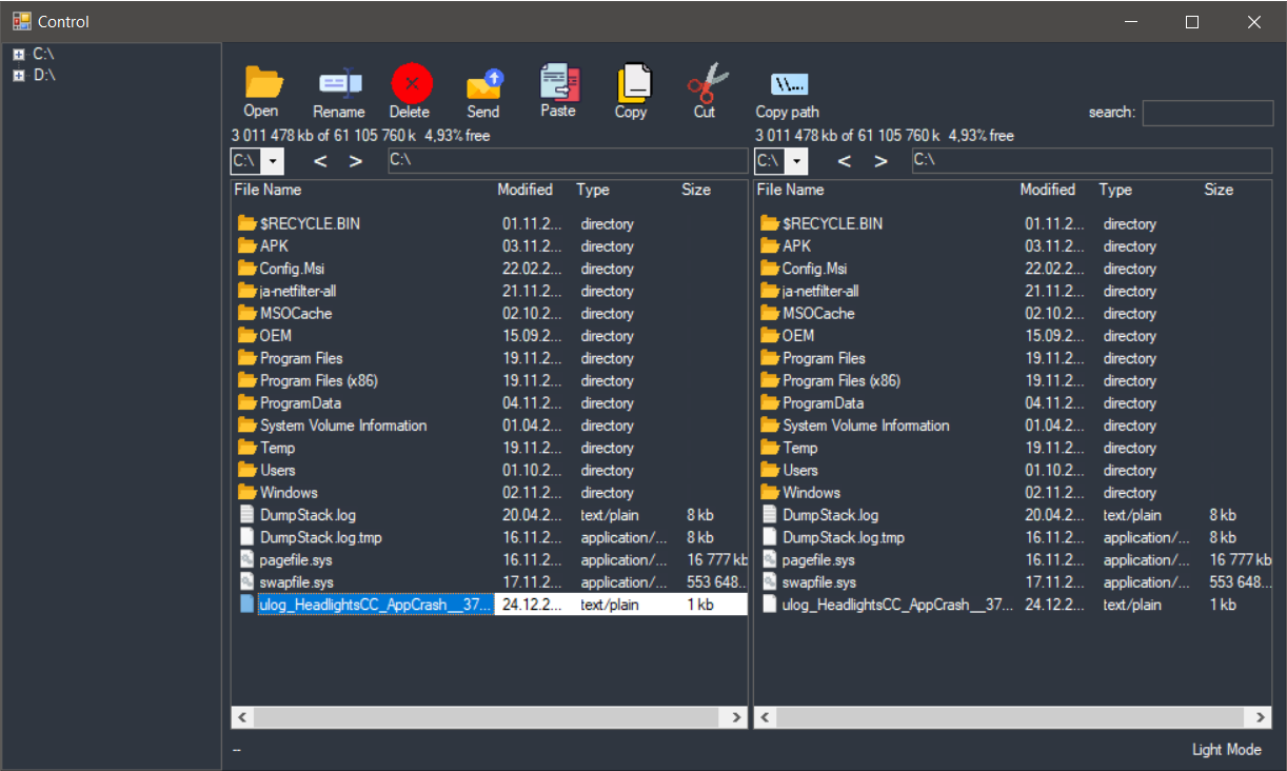


Рис. 3.14 - 3.15 Пошук файлів.

Програма пропонує кілька варіантів кольорових схем, таких як світла або темна. Користувач вибирає потрібну тему або налаштування кольорів, і зміни застосовуються в реальному часі. Інтерфейс оновлюється, відображаючи вибрану кольорову гаму. Після цього користувач закриває меню налаштувань і продовжує працювати в новій кольоровій гамі. У випадку, якщо зміни не задовольняють користувача, він може вибрати інший варіант. (Рис. 3.16 – 3.17).



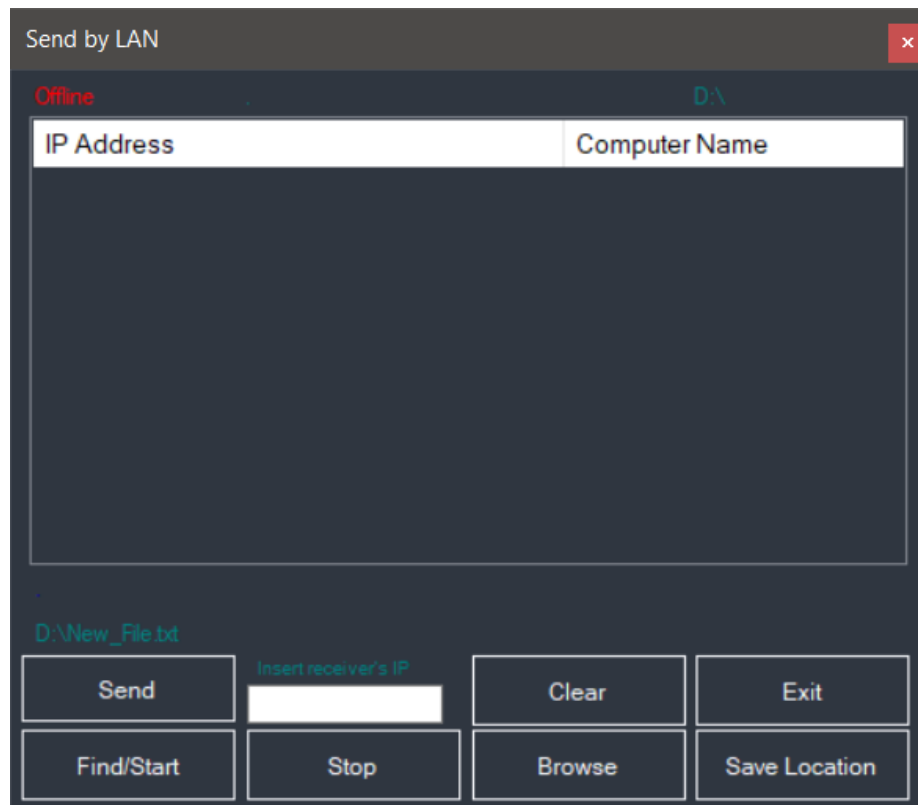


Рис. 3.16 – 3.17 Темна кольорова тема

3.3 Тестування передачі файлів по локальній мережі

Тестування передачі файлів по локальній мережі є важливим етапом для забезпечення надійності та ефективності системи обміну даними. Цей процес дозволяє виявити та усунути потенційні проблеми, що можуть виникнути під час передачі файлів між різними пристроями в локальній мережі. Тестування включає перевірку стабільності з'єднання, швидкості передачі даних, цілісності файлів та сумісності з різними операційними системами та мережевими конфігураціями. Основною метою є забезпечення безперебійного та безпечного обміну файлами для користувачів, що використовують локальну мережу для роботи та обміну інформацією.

Натиснувши кнопку пошуку можемо спостерігати наявні в мережі пристрої. Як видно по назві ПК являється перший з них.

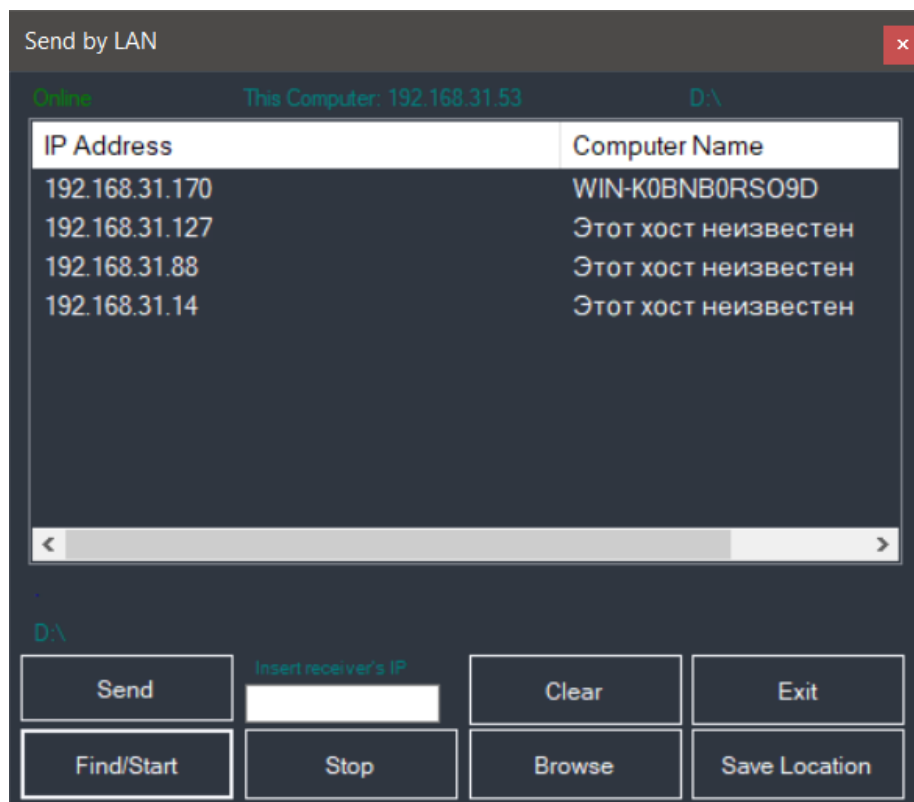


Рис. 3.18 Вікно обміну по локальній мережі

Процес відправки файлу починається з відкриття вікна відправки. У цьому вікні користувач бачить кнопки та інструменти, які допомагають налаштувати та виконати відправку. Спочатку користувач вводить адресу отримувача у рядок адреси або вибирає її зі списку у вікні адрес. Кнопка очищення дозволяє видалити неактуальну адресу.

Далі натискається кнопка пошуку для доступу до файлів, які потрібно відправити. Відкривається вікно, у якому користувач може вибрати один або кілька файлів. Обрані файли додаються до списку у вікні інтерфейсу, після чого користувач може перевірити їх перед відправкою. Якщо потрібний файл не відображається або був вибраний помилково, користувач може повторно скористатися кнопкою вибору файлів, щоб виправити це. (3.19 – 3.20).

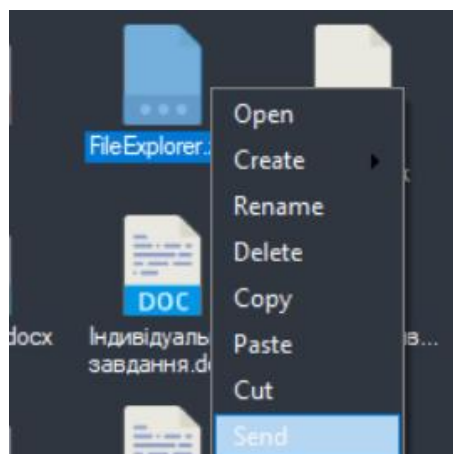


Рис. 3.19

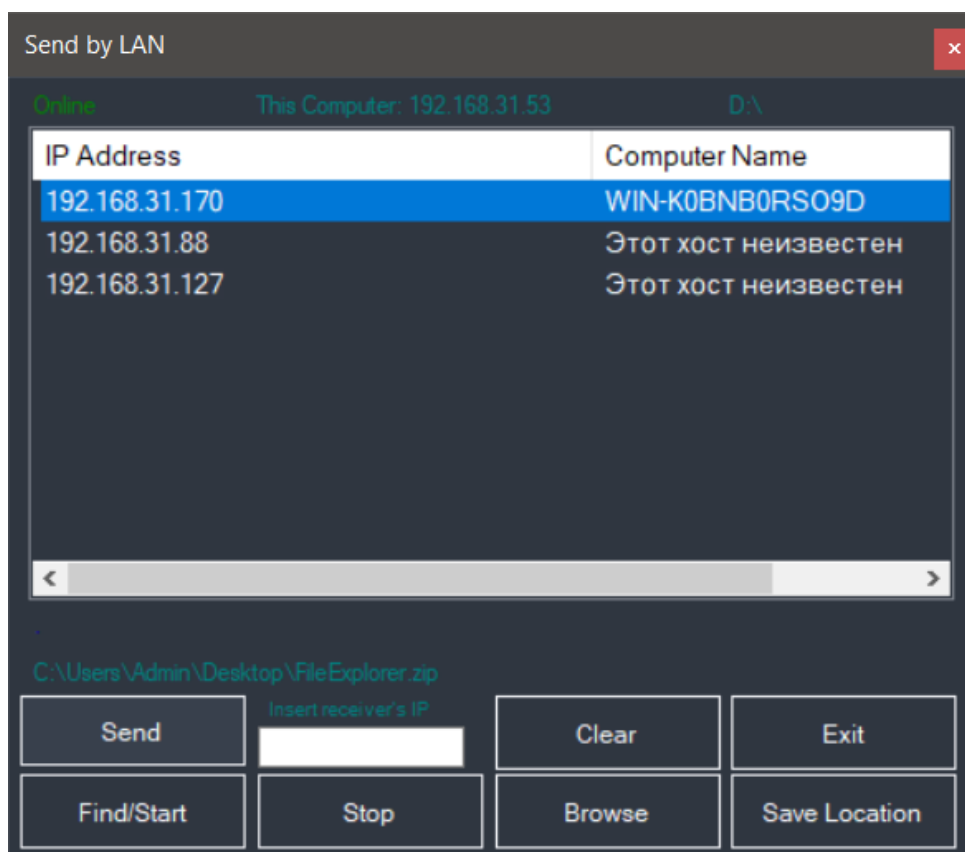


Рис. 3.20 Вікно відправника

Коли все готово, натискається кнопка відправки. Програма починає передачу файлів, і процес відображається у вигляді надпису про відправку на вибрану адресу. Якщо файл успішно відправлений, інтерфейс повідомляє про завершення, а користувач може закрити вікно, натиснувши кнопку виходу.

Бачимо повідомлення про надсилання архіву та про отримання на іншому ПК, а також сам архів та повідомлення з пропозицією зберегти або видалити файл (3.21 – 3.22).

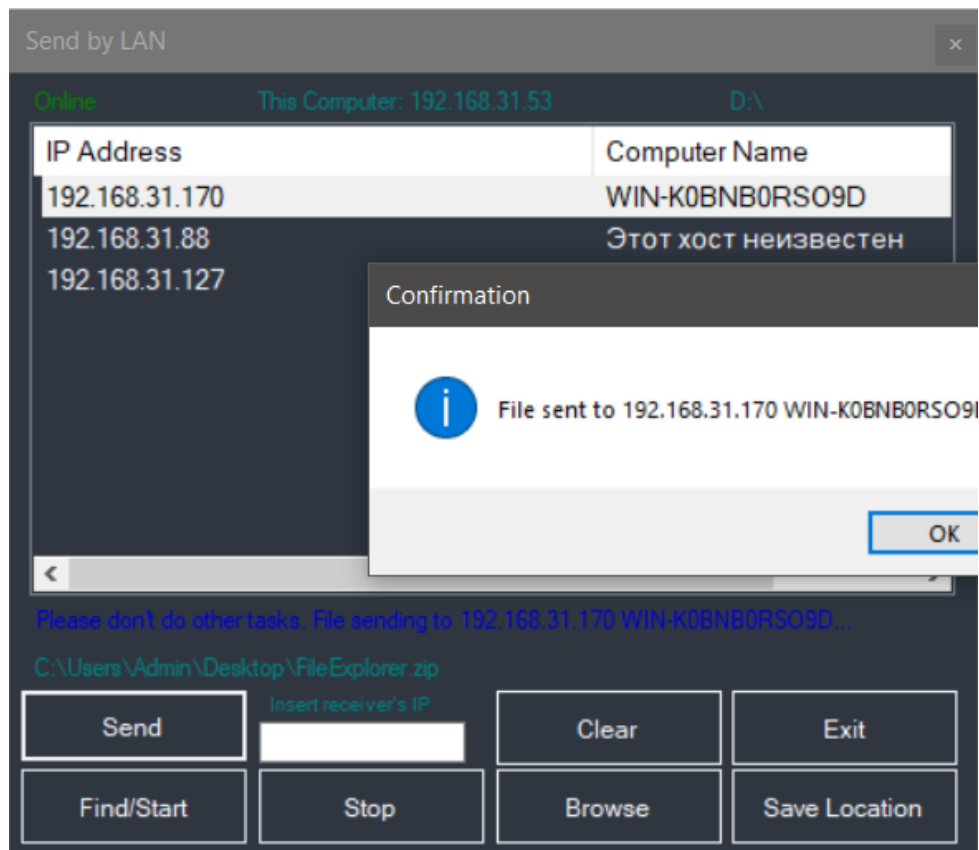


Рис. 3.21 Вікно відправника

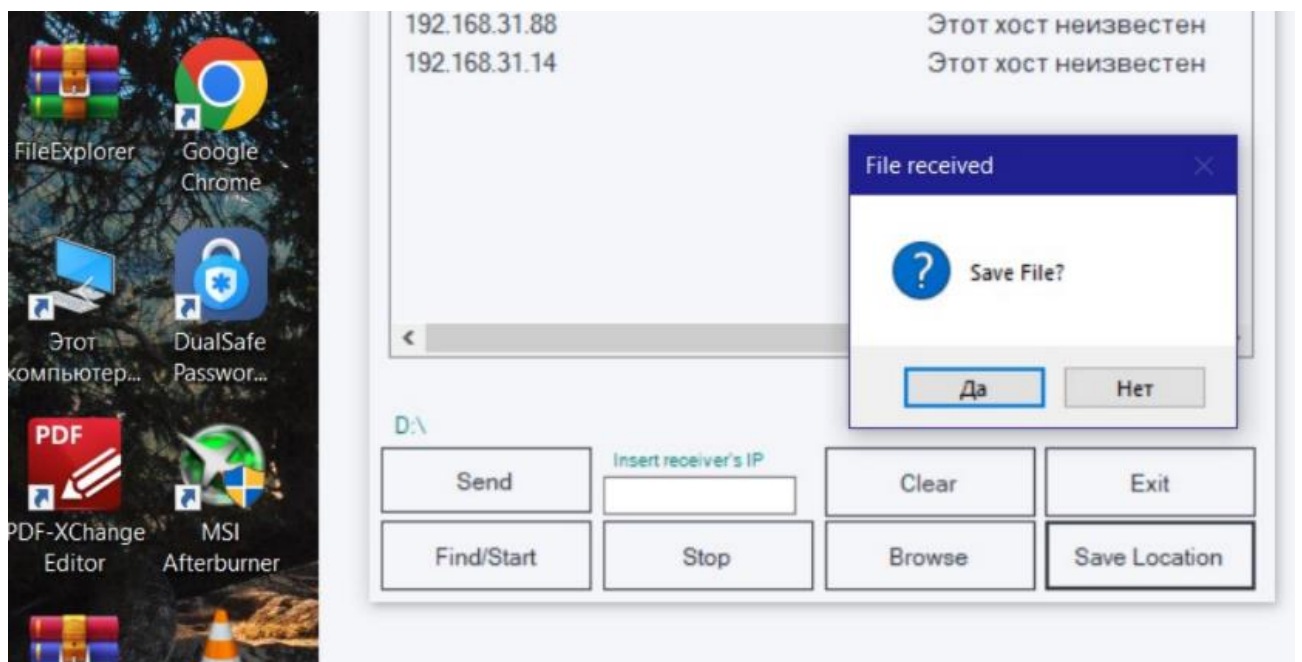


Рис. 3.22 Вікно отримувача

Тепер виконаємо у зворотньому порядку. Надсилаємо файл з іншого ПК на наш. Бачимо отриманий файл та діалогове вікно (3.23 – 3.24).

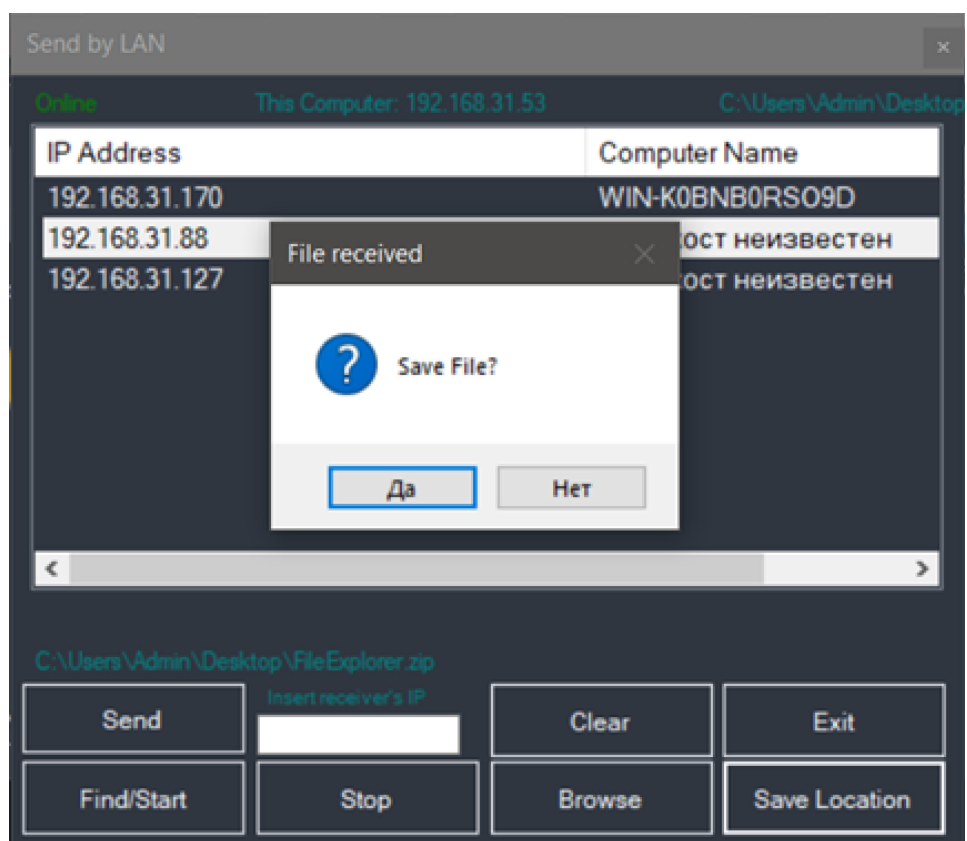


Рис. 3.23 Вікно отримувача

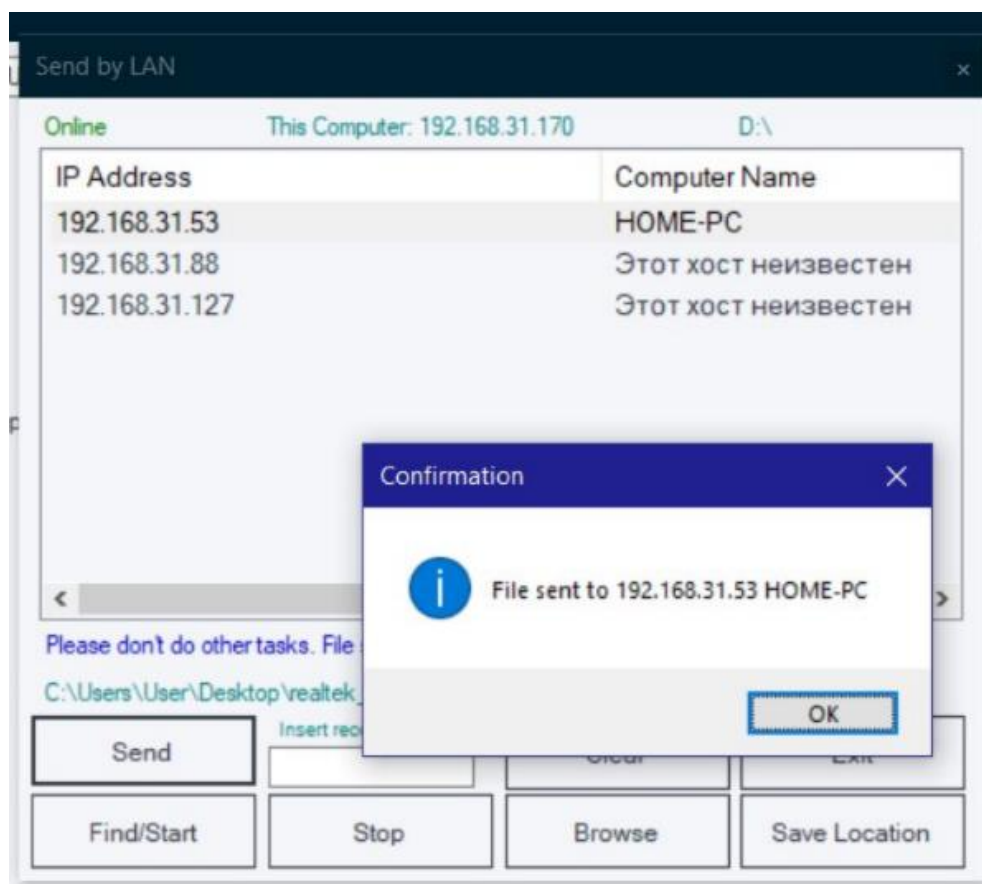


Рис. 3.24 Вікно відправника

Висновки до третього розділу

В третьому розділі було докладно описано інтерфейс застосунку для управління та обміну файлами та його основні функціональні можливості. Особлива увага приділялася реалізації ключових операцій, які доступні користувачам: копіювання, переміщення, видалення та перейменування файлів і папок. Ці функції були спроектовані з урахуванням зручності та інтуїтивності, щоб забезпечити максимально ефективну взаємодію з файловою системою.

Крім цього, було проведено аналіз роботи програми з мережею, зокрема можливостей передачі файлів між пристроями в локальній мережі. Розглянуто, як налаштування мережевих з'єднань і їх тестування впливають на швидкість та стабільність передачі даних. Це дозволило створити рішення, яке забезпечує користувачам безпечний та швидкий обмін файлами у локальній мережі, враховуючи сучасні вимоги до надійності та продуктивності.

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		66

ВИСНОВКИ

Під час розробки були проаналізовані файлові менеджери від інших виробників і для різних операційних систем, що дало змогу краще зрозуміти різноманітність функціональних можливостей і дизайнів інтерфейсів для управління файлами. Це дослідження дозволило сформулювати чітке уявлення про те, яким має бути створюваний додаток. Отримані теоретичні знання та практичний досвід стали основою для подальшого написання програмного коду за допомогою мови програмування C# і використання Windows Forms для створення інтерфейсу.

У ході розробки було спроектовано загальний алгоритм роботи програми, що включає послідовність дій і сценарії використання. Було продумано структуру додатку, визначено ключові елементи та можливі дії користувача. Інтерфейс програми створювався з урахуванням усіх необхідних складових для зручного та ефективного управління файлами. Окрему увагу приділено реалізації можливості змінювати світлову тему програми, щоб забезпечити комфортну роботу для користувачів у різних умовах.

Основною метою проектування та реалізації застосунку для управління та обміну файлами було забезпечення простого та ефективного інструменту для управління файлами, орієнтованого на користувачів, які працюють у локальних мережах для спільного використання та обміну даними. На основі сформованих вимог було розроблено функціональний алгоритм програми, який відповідає задачам і забезпечує реалізацію основного функціоналу.

В цілому, завершений додаток став результатом виконання всіх етапів розробки: від аналізу вимог і проектування до написання коду, тестування та впровадження. Він відповідає сучасним стандартам розробки програмного забезпечення та забезпечує зручність і ефективність для користувачів, які працюють із файлами у своїй повсякденній діяльності.

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						67
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Windows File Explorer [Електронний ресурс] – Режим доступу до ресурсу:
<https://support.microsoft.com/uk-ua/windows/find-and-open-file-explorer-ef370130-1cca-9dc5-e0df-2f7416fe1cb1>
2. Dolphin [Електронний ресурс] – Режим доступу до ресурсу:
<https://apps.kde.org/dolphin>
3. Commander One [Електронний ресурс] – Режим доступу до ресурсу:
<https://commander-one.com>
4. Total Commander [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.ghisler.com>
5. Finder [Електронний ресурс] – Режим доступу до ресурсу:
<https://support.apple.com/uk-ua/guide/mac-help/mchlp2605/mac>
6. Windows Forms .NET [Електронний ресурс] – Режим доступу до ресурсу:
<https://learn.microsoft.com/uk-ua/dotnet/desktop/winforms/overview/?view=netdesktop-8.0>
7. Common I/O Tasks [Електронний ресурс] – Режим доступу до ресурсу:
<https://learn.microsoft.com/uk-ua/dotnet/standard/io/common-i-o-tasks>
8. File and Stream I/O [Електронний ресурс] – Режим доступу до ресурсу:
<https://learn.microsoft.com/uk-ua/dotnet/standard/io>
9. ListView [Електронний ресурс] – Режим доступу до ресурсу:
<https://learn.microsoft.com/uk-ua/dotnet/api/system.windows.forms.listview?view=windowsdesktop-8.0>
10. Using threads [Електронний ресурс] – Режим доступу до ресурсу:
<https://learn.microsoft.com/uk-ua/dotnet/standard/threading/using-threads-and-threading>
11. TcpListener [Електронний ресурс] – Режим доступу до ресурсу:
<https://learn.microsoft.com/uk-ua/dotnet/api/system.net.sockets.tcplistener?view=net-8.0>
12. Events [Електронний ресурс] – Режим доступу до ресурсу:
<https://learn.microsoft.com/uk-ua/dotnet/api/system.events?view=net-8.0>

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

[ua/dotnet/api/system.componentmodel.component.events?view=net-8.0&viewFallbackFrom=windowsdesktop-8.0#system-componentmodel-component-events](https://learn.microsoft.com/uk-ua/dotnet/api/system.componentmodel.component.events?view=net-8.0&viewFallbackFrom=windowsdesktop-8.0#system-componentmodel-component-events)

13. NetworkInformation [Електронний ресурс] – Режим доступу до ресурсу:

[https://learn.microsoft.com/uk-](https://learn.microsoft.com/uk-ua/dotnet/api/system.net.networkinformation?view=net-8.0)

[ua/dotnet/api/system.net.networkinformation?view=net-8.0](https://learn.microsoft.com/uk-ua/dotnet/api/system.net.networkinformation?view=net-8.0)

14. Thread [Електронний ресурс] – Режим доступу до ресурсу:

<https://learn.microsoft.com/uk-ua/dotnet/api/system.threading.thread?view=net-8.0>

15. ConfigurationManager.AppSettings [Електронний ресурс] – Режим доступу до ресурсу:

[https://learn.microsoft.com/uk-](https://learn.microsoft.com/uk-ua/dotnet/api/system.configuration.configurationmanager.appsettings?view=net-8.0)

[ua/dotnet/api/system.configuration.configurationmanager.appsettings?view=net-8.0](https://learn.microsoft.com/uk-ua/dotnet/api/system.configuration.configurationmanager.appsettings?view=net-8.0)

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						69
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

```

using System;
using System.Configuration;
using System.Drawing;
using System.Diagnostics;
using System.IO;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using System.Linq;

namespace FileExplorer
{
    public partial class MainForm : Form
    {
        private string filePath1 = "D:\\";
        private string filePath2 = "D:\\";
        private bool isFile1 = false;
        private bool isFile2 = false;
        List<string> selectedItems1 = new List<string>();
        List<string> selectedItems2 = new List<string>();
        List<string> copyItems = new List<string>();
        private bool cut = false;
        private bool firstInFocus = true;
        SendForm tf;
        List<ProgressForm> progressForms = new List<ProgressForm>();
        private ListViewColumnSorter lvwColumnSorter;

        Thread mainThread = Thread.CurrentThread;
        Thread countThread;

        private static List<CancellationTokenSource> tokenSources = new
List<CancellationTokenSource>();
        private CancellationTokenSource searchTokenSource;

        public MainForm()
        {
            InitializeComponent();
            InitializeUI("UIMode");
            lvwColumnSorter = new ListViewColumnSorter();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            listView1.OwnerDraw = true;
            listView2.OwnerDraw = true;
            filePathTextBox1.Text = filePath1;
            filePathTextBox2.Text = filePath2;
            ShowDrives();
            loadListView1();
            loadListView2();
            checkVolumes();
        }

        private void MainForm_MouseClick(object sender, MouseEventArgs e)
        {
        }

        private void listView1_MouseClick(object sender, MouseEventArgs e)
        {
            if (contextMenuStrip1.Visible == true)
            {
                firstInFocus = true;
                Console.WriteLine(firstInFocus);
            }
        }
    }
}

```

```

        if (listView1.SelectedItems.Count > 0)
        {
            contextMenuStrip1.Items[1].Enabled = true;
            contextMenuStrip1.Items[2].Enabled = true;
            contextMenuStrip1.Items[3].Enabled = true;
            contextMenuStrip1.Items[4].Enabled = true;
            contextMenuStrip1.Items[5].Enabled = true;
            contextMenuStrip1.Items[6].Enabled = true;
            contextMenuStrip1.Items[7].Enabled = true;
        }
        else
        {
            contextMenuStrip1.Items[1].Enabled = false;
            contextMenuStrip1.Items[2].Enabled = false;
            contextMenuStrip1.Items[3].Enabled = false;
            contextMenuStrip1.Items[4].Enabled = false;
            contextMenuStrip1.Items[5].Enabled = false;
            contextMenuStrip1.Items[6].Enabled = false;
            contextMenuStrip1.Items[7].Enabled = false;
        }
    }

private void listView2_MouseClick(object sender, MouseEventArgs e)
{
    if (contextMenuStrip1.Visible == true)
    {
        firstInFocus = false;
        Console.WriteLine(firstInFocus);
        if (listView2.SelectedItems.Count > 0)
        {
            contextMenuStrip1.Items[1].Enabled = true;
            contextMenuStrip1.Items[2].Enabled = true;
            contextMenuStrip1.Items[3].Enabled = true;
            contextMenuStrip1.Items[4].Enabled = true;
            contextMenuStrip1.Items[5].Enabled = true;
            contextMenuStrip1.Items[6].Enabled = true;
            contextMenuStrip1.Items[7].Enabled = true;
        }
        else
        {
            contextMenuStrip1.Items[1].Enabled = false;
            contextMenuStrip1.Items[2].Enabled = false;
            contextMenuStrip1.Items[3].Enabled = false;
            contextMenuStrip1.Items[4].Enabled = false;
            contextMenuStrip1.Items[5].Enabled = false;
            contextMenuStrip1.Items[6].Enabled = false;
            contextMenuStrip1.Items[7].Enabled = false;
        }
    }
}

private void listView1_SelectedIndexChanged(object sender, EventArgs e)
{
    firstInFocus = true;
    Console.WriteLine(firstInFocus);
    if (listView1.SelectedItems.Count > 0)
    {
        openButton.Enabled = true;
        renameButton.Enabled = true;
        deleteButton.Enabled = true;
        sendButton.Enabled = true;
        pasteButton.Enabled = true;
        copyButton.Enabled = true;
        cutButton.Enabled = true;
    }
}

```

```

        copyPathButton.Enabled = true;

        contextMenuStrip1.Items[1].Enabled = true;
        contextMenuStrip1.Items[2].Enabled = true;
        contextMenuStrip1.Items[3].Enabled = true;
        contextMenuStrip1.Items[4].Enabled = true;
        contextMenuStrip1.Items[5].Enabled = true;
        contextMenuStrip1.Items[6].Enabled = true;
        contextMenuStrip1.Items[7].Enabled = true;
    }
    else
    {
        openButton.Enabled = false;
        renameButton.Enabled = false;
        deleteButton.Enabled = false;
        sendButton.Enabled = false;
        pasteButton.Enabled = false;
        copyButton.Enabled = false;
        cutButton.Enabled = false;
        copyPathButton.Enabled = false;

        contextMenuStrip1.Items[1].Enabled = false;
        contextMenuStrip1.Items[2].Enabled = false;
        contextMenuStrip1.Items[3].Enabled = false;
        contextMenuStrip1.Items[4].Enabled = false;
        contextMenuStrip1.Items[5].Enabled = false;
        contextMenuStrip1.Items[6].Enabled = false;
        contextMenuStrip1.Items[7].Enabled = false;
    }
    if (copyItems.Count > 0)
    {
        pasteButton.Enabled = true;
        contextMenuStrip1.Items[5].Enabled = true;
    }
}

private void listView2_SelectedIndexChanged(object sender, EventArgs e)
{
    firstInFocus = false;
    Console.WriteLine(firstInFocus);
    if (listView2.SelectedItems.Count > 0)
    {
        openButton.Enabled = true;
        renameButton.Enabled = true;
        deleteButton.Enabled = true;
        sendButton.Enabled = true;
        pasteButton.Enabled = true;
        copyButton.Enabled = true;
        cutButton.Enabled = true;
        copyPathButton.Enabled = true;

        contextMenuStrip1.Items[1].Enabled = true;
        contextMenuStrip1.Items[2].Enabled = true;
        contextMenuStrip1.Items[3].Enabled = true;
        contextMenuStrip1.Items[4].Enabled = true;
        contextMenuStrip1.Items[5].Enabled = true;
        contextMenuStrip1.Items[6].Enabled = true;
        contextMenuStrip1.Items[7].Enabled = true;
    }
    else
    {
        openButton.Enabled = false;
        renameButton.Enabled = false;
        deleteButton.Enabled = false;
        sendButton.Enabled = false;
    }
}

```

```

pasteButton.Enabled = false;
copyButton.Enabled = false;
cutButton.Enabled = false;
copyPathButton.Enabled = false;

contextMenuStrip1.Items[1].Enabled = false;
contextMenuStrip1.Items[2].Enabled = false;
contextMenuStrip1.Items[3].Enabled = false;
contextMenuStrip1.Items[4].Enabled = false;
contextMenuStrip1.Items[5].Enabled = false;
contextMenuStrip1.Items[6].Enabled = false;
contextMenuStrip1.Items[7].Enabled = false;
}
if (copyItems.Count > 0)
{
    pasteButton.Enabled = true;
    contextMenuStrip1.Items[5].Enabled = true;
}
}

public void loadListView1()
{
    DirectoryInfo fileList;
    string tempFilePath = "";
    FileAttributes fileAttr;
    try
    {
        if (isFile1)
        {
            if (File.Exists(Path.GetFullPath(selectedItems1[0])))
                tempFilePath = Path.GetFullPath(selectedItems1[0]);
            else
                tempFilePath = addBackSlash() + selectedItems1[0];
            FileInfo fileDetails = new FileInfo(tempFilePath);
            fileAttr = File.GetAttributes(tempFilePath);
            Process.Start(tempFilePath);
        }
        else
        {
            fileAttr = File.GetAttributes(filePath1);
        }

        if ((fileAttr & FileAttributes.Directory) == FileAttributes.Directory)
        {
            string subOutput, date;
            ListViewItem item;
            Icon fileIcon = SystemIcons.WinLogo;
            fileList = new DirectoryInfo(filePath1);
            FileInfo[] files = fileList.GetFiles();
            DirectoryInfo[] dirs = fileList.GetDirectories();
            listView1.Items.Clear();
            listView1.BeginUpdate();

            foreach (DirectoryInfo dir in dirs)
            {
                date = dir.LastWriteTime.ToString("dd.MM.yyyy HH:mm");
                subOutput = " ";
                item = new ListViewItem(dir.Name);
                item.ImageIndex = 0;
                item.SubItems.Add(date);
                item.SubItems.Add("directory");
                item.SubItems.Add(subOutput);
                listView1.Items.Add(item);
            }
        }
    }
    catch { }
}

```

```

        foreach (FileInfo file in files)
        {
            if (!imageList1.Images.ContainsKey(file.Extension))
            {
                fileIcon =
System.Drawing.Icon.ExtractAssociatedIcon(file.FullName);
                imageList1.Images.Add(file.Extension, fileIcon);
            }
            date = file.LastWriteTime.ToString("dd.MM.yyyy HH:mm");
            subOutput = $"{file.Length / 1000:n0} kb";
            item = new ListViewItem(file.Name);
            item.ImageKey = file.Extension;
            item.SubItems.Add(date);
            item.SubItems.Add(MimeTypes.GetContentType(file.FullName));
            item.SubItems.Add(subOutput);
            listView1.Items.Add(item);
        }
        listView1.EndUpdate();
    }
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
}

public void loadListView2()
{
    DirectoryInfo fileList;
    string tempFilePath = "";
    FileAttributes fileAttr;
    try
    {
        if (isFile2)
        {
            if (File.Exists(Path.GetFullPath(selectedItems2[0])))
                tempFilePath = Path.GetFullPath(selectedItems2[0]);
            else
                tempFilePath = addBackSlash() + selectedItems2[0];
            FileInfo fileDetails = new FileInfo(tempFilePath);
            fileAttr = File.GetAttributes(tempFilePath);
            Process.Start(tempFilePath);
        }
        else
        {
            fileAttr = File.GetAttributes(filePath2);
        }

        if ((fileAttr & FileAttributes.Directory) == FileAttributes.Directory)
        {
            string subOutput, date;
            ListViewItem item;
            Icon fileIcon = SystemIcons.WinLogo;
            fileList = new DirectoryInfo(filePath2);
            FileInfo[] files = fileList.GetFiles();
            DirectoryInfo[] dirs = fileList.GetDirectories();
            listView2.Items.Clear();
            listView2.BeginUpdate();

            foreach (DirectoryInfo dir in dirs)
            {
                date = dir.LastWriteTime.ToString("dd.MM.yyyy HH:mm");
                subOutput = " ";
                item = new ListViewItem(dir.Name);
            }
        }
    }
}

```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						75
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        item.ImageIndex = 0;
        item.SubItems.Add(date);
        item.SubItems.Add("directory");
        item.SubItems.Add(" ");
        listView2.Items.Add(item);
    }

    foreach (FileInfo file in files)
    {
        if (!imageList1.Images.ContainsKey(file.Extension))
        {
            fileIcon =
System.Drawing.Icon.ExtractAssociatedIcon(file.FullName);
            imageList1.Images.Add(file.Extension, fileIcon);
        }
        date = file.LastWriteTime.ToString("dd.MM.yyyy HH:mm");
        subOutput = $"{file.Length / 1000:n0} kb";
        item = new ListViewItem(file.Name);
        item.ImageKey = file.Extension;
        item.SubItems.Add(date);
        item.SubItems.Add(MimeTypes.GetContentType(file.FullName));
        item.SubItems.Add(subOutput);
        listView2.Items.Add(item);
    }
    listView2.EndUpdate();
}
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
}

public void loadAction1()
{
    Invoke((MethodInvoker)delegate
    {
        checkVolumes();
        removeBackSlash1();
        filePath1 = filePathTextBox1.Text;
        loadListView1();
        isFile1 = false;
    });
}

public void loadAction2()
{
    Invoke((MethodInvoker)delegate
    {
        checkVolumes();
        removeBackSlash2();
        filePath2 = filePathTextBox2.Text;
        loadListView2();
        isFile2 = false;
    });
}

public void removeBackSlash1()
{
    try
    {
        string path = filePathTextBox1.Text;
        if (path.LastIndexOf("\\") == path.Length - 1 && path.LastIndexOf(":")
!= path.Length - 2)
        {

```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						76
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        filePathTextBox1.Text = path.Substring(0, path.Length - 1);
    }
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
}

public void removeBackSlash2()
{
    try
    {
        string path = filePathTextBox2.Text;
        if (path.LastIndexOf("\\") == path.Length - 1 && path.LastIndexOf(":")
!= path.Length - 2)
        {
            filePathTextBox2.Text = path.Substring(0, path.Length - 1);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}

public void goBack1()
{
    try
    {
        string path = filePathTextBox1.Text;
        path = path.Substring(0, path.LastIndexOf("\\")+1);
        this.isFile1 = false;
        filePathTextBox1.Text = path;
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}

public void goBack2()
{
    try
    {
        string path = filePathTextBox2.Text;
        path = path.Substring(0, path.LastIndexOf("\\") + 1);
        this.isFile2 = false;
        filePathTextBox2.Text = path;
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}

private void goButton1_Click(object sender, EventArgs e)
{
    loadAction1();
}
private void goButton2_Click(object sender, EventArgs e)
{
    loadAction2();
}

```

```

private void listView1_ItemSelectionChanged(object sender,
ListViewItemSelectionChangedEventArgs e)
{
    try
    {
        selectedItems1 = new List<string>();
        foreach (ListViewItem item in listView1.SelectedItems)
        {
            selectedItems1.Add(item.Text);
            Console.Write(item.Text + "\t");
        }

        FileAttributes fileAttr;
        if (File.Exists(Path.GetFullPath(selectedItems1[0])))
            fileAttr = File.GetAttributes(selectedItems1[0]);
        else
            fileAttr = File.GetAttributes(addBackSlash() + selectedItems1[0]);

        if ((fileAttr & FileAttributes.Directory) == FileAttributes.Directory)
        {
            isFile1 = false;
        }
        else
        {
            isFile1 = true;
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

private void listView2_ItemSelectionChanged(object sender,
ListViewItemSelectionChangedEventArgs e)
{
    try
    {
        selectedItems2 = new List<string>();
        foreach (ListViewItem item in listView2.SelectedItems)
        {
            selectedItems2.Add(item.Text);
            Console.Write(item.Text);
        }

        FileAttributes fileAttr;
        if (File.Exists(Path.GetFullPath(selectedItems2[0])))
            fileAttr = File.GetAttributes(selectedItems2[0]);
        else
            fileAttr = File.GetAttributes(addBackSlash() + selectedItems2[0]);

        if ((fileAttr & FileAttributes.Directory) == FileAttributes.Directory)
        {
            isFile2 = false;
        }
        else
        {
            isFile2 = true;
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine("listView2_ItemSelectionChanged " + ex.Message);
    }
}

```

```

    }
}

private void listView1_MouseDoubleClick(object sender, MouseEventArgs e)
{
    try
    {
        if (e.Button == MouseButtons.Left)
        {
            FileAttributes fileAttr;
            if (File.Exists(Path.GetFullPath(selectedItems1[0])))
                fileAttr = File.GetAttributes(selectedItems1[0]);
            else
                fileAttr = File.GetAttributes(addBackSlash() +
selectedItems1[0]);

            if ((fileAttr & FileAttributes.Directory) ==
FileAttributes.Directory)
            {
                filePathTextBox1.Text = addBackSlash() + selectedItems1[0];
            }
            loadAction1();
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

private void listView2_MouseDoubleClick(object sender, MouseEventArgs e)
{
    try
    {
        if (e.Button == MouseButtons.Left)
        {
            FileAttributes fileAttr;
            if (File.Exists(Path.GetFullPath(selectedItems2[0])))
                fileAttr = File.GetAttributes(selectedItems2[0]);
            else
                fileAttr = File.GetAttributes(addBackSlash() +
selectedItems2[0]);

            if ((fileAttr & FileAttributes.Directory) ==
FileAttributes.Directory)
            {
                filePathTextBox2.Text = addBackSlash() + selectedItems2[0];
            }
            loadAction2();
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

private void backButton1_Click(object sender, EventArgs e)
{
    goBack1();
    loadAction1();
}

private void backButton2_Click(object sender, EventArgs e)
{
    goBack2();
}

```

```

        loadAction2();
    }

    private string addBackSlash()
    {
        string path = firstInFocus ? filePath1 : filePath2;
        if (path.Length - 2 != path.LastIndexOf(":"))
        {
            path += "\\";
        }
        return path;
    }

    private void InitializeUI(string key)
    {
        try
        {
            var uiMode = ConfigurationManager.AppSettings[key];
            if (uiMode == "light")
            {
                btnChangeMode.Text = "Dark Mode";
                this.ForeColor = Color.FromArgb(47, 54, 64);
                this.BackColor = Color.FromArgb(255, 255, 255);
                listView1.ForeColor = Color.FromArgb(47, 54, 64);
                listView1.BackColor = Color.FromArgb(255, 255, 255);
                listView2.ForeColor = Color.FromArgb(47, 54, 64);
                listView2.BackColor = Color.FromArgb(255, 255, 255);
                treeView1.ForeColor = Color.FromArgb(47, 54, 64);
                treeView1.BackColor = Color.FromArgb(255, 255, 255);
                filePathTextBox1.ForeColor = Color.FromArgb(47, 54, 64);
                filePathTextBox1.BackColor = Color.FromArgb(255, 255, 255);
                filePathTextBox2.ForeColor = Color.FromArgb(47, 54, 64);
                filePathTextBox2.BackColor = Color.FromArgb(255, 255, 255);
                searchTextBox.ForeColor = Color.FromArgb(47, 54, 64);
                searchTextBox.BackColor = Color.FromArgb(255, 255, 255);
                comboBox1.ForeColor = Color.FromArgb(47, 54, 64);
                comboBox1.BackColor = Color.FromArgb(255, 255, 255);
                comboBox2.ForeColor = Color.FromArgb(47, 54, 64);
                comboBox2.BackColor = Color.FromArgb(255, 255, 255);
                contextMenuStrip1.ForeColor = Color.FromArgb(47, 54, 64);
                contextMenuStrip1.BackColor = Color.FromArgb(255, 255, 255);
                ConfigurationManager.AppSettings[key] = "dark";
            }
            else
            {
                btnChangeMode.Text = "Light Mode";
                this.ForeColor = Color.FromArgb(255, 255, 255);
                this.BackColor = Color.FromArgb(47, 54, 64);
                listView1.ForeColor = Color.FromArgb(255, 255, 255);
                listView1.BackColor = Color.FromArgb(47, 54, 64);
                listView2.ForeColor = Color.FromArgb(255, 255, 255);
                listView2.BackColor = Color.FromArgb(47, 54, 64);
                treeView1.ForeColor = Color.FromArgb(255, 255, 255);
                treeView1.BackColor = Color.FromArgb(47, 54, 64);
                filePathTextBox1.ForeColor = Color.FromArgb(255, 255, 255);
                filePathTextBox1.BackColor = Color.FromArgb(47, 54, 64);
                filePathTextBox2.ForeColor = Color.FromArgb(255, 255, 255);
                filePathTextBox2.BackColor = Color.FromArgb(47, 54, 64);
                searchTextBox.ForeColor = Color.FromArgb(255, 255, 255);
                searchTextBox.BackColor = Color.FromArgb(47, 54, 64);
                comboBox1.ForeColor = Color.FromArgb(255, 255, 255);
                comboBox1.BackColor = Color.FromArgb(47, 54, 64);
                comboBox2.ForeColor = Color.FromArgb(255, 255, 255);
                comboBox2.BackColor = Color.FromArgb(47, 54, 64);
                contextMenuStrip1.ForeColor = Color.FromArgb(255, 255, 255);
            }
        }
    }

```

```

        contextMenuStrip1.BackColor = Color.FromArgb(47, 54, 64);
        ConfigurationManager.AppSettings[key] = "light";
    }
}
catch (Exception ex)
{
    throw;
}
}

private void btnChangeMode_Click(object sender, EventArgs e)
{
    InitializeUI("UIMode");
}

private void renameMethod()
{
    if (firstInFocus)
    {
        if (listView1.SelectedItems.Count > 0)
        {
            ListViewItem selectedItem = listView1.SelectedItems[0];
            listView1.LabelEdit = true; // Дозволити редагування
            selectedItem.BeginEdit(); // Розпочати редагування
        }
    }
    else
    {
        if (listView2.SelectedItems.Count > 0)
        {
            ListViewItem selectedItem = listView2.SelectedItems[0];
            listView2.LabelEdit = true; // Дозволити редагування
            selectedItem.BeginEdit(); // Розпочати редагування
        }
    }
}

private void listView1_AfterLabelEdit(object sender, LabelEditEventArgs e)
{
    if (e.Label != null)
    {
        string oldName = listView1.Items[e.Item].Text;
        string newName = e.Label;
        string oldFilePath, newFilePath;

        if (File.Exists(Path.GetFullPath(oldName)))
        {
            oldFilePath = Path.GetFullPath(selectedItems1[0]);
            newFilePath = newName;

            Console.WriteLine("rename" + newFilePath);
        }
        else
        {
            oldFilePath = addBackSlash() + oldName;
            newFilePath = addBackSlash() + newName;
        }
        try
        {
            FileAttributes fileAttr = File.GetAttributes(oldFilePath);
            if ((fileAttr & FileAttributes.Directory) ==
FileAttributes.Directory)
            {
                if (newName != "" && !Directory.Exists(newFilePath))
                {

```

```

        Directory.Move(oldFilePath, newFilePath);
    }
}
else
{
    if (newName != "" && !File.Exists(newFilePath))
    {
        File.Move(oldFilePath, newFilePath);
    }
}

ListView1 listView1.Items[e.Item].Text = newName; // Оновлюємо текст у
}
catch (Exception ex)
{
    MessageBox.Show($"Error renaming file: {ex.Message}");
}
}

private void listView2_AfterLabelEdit(object sender, LabelEditEventArgs e)
{
    if (e.Label != null)
    {
        string oldName = listView2.Items[e.Item].Text;
        string newName = e.Label;
        string oldFilePath, newFilePath;

        if (File.Exists(Path.GetFullPath(oldName)))
        {
            oldFilePath = Path.GetFullPath(selectedItems1[0]);
            newFilePath = oldFilePath.Substring(0,
oldFilePath.LastIndexOf(@"\"));
        }
        else
        {
            oldFilePath = addBackSlash() + oldName;
            newFilePath = addBackSlash() + newName;
        }
        try
        {
            FileAttributes fileAttr = File.GetAttributes(oldFilePath);
            if ((fileAttr & FileAttributes.Directory) ==
FileAttributes.Directory)
            {
                if (newName != "" && !Directory.Exists(newFilePath))
                {
                    Directory.Move(oldFilePath, newFilePath);
                }
            }
            else
            {
                if (newName != "" && !File.Exists(newFilePath))
                {
                    File.Move(oldFilePath, newFilePath);
                }
            }

            listView2.Items[e.Item].Text = newName; // Оновлюємо текст у
ListView2
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Error renaming file: {ex.Message}");
        }
    }
}

```

```

    }
    }
}

private void openMethod()
{
    try
    {
        List<string> selectedItems = firstInFocus ? selectedItems1 :
selectedItems2;

        FileAttributes fileAttr;
        if (File.Exists(Path.GetFullPath(selectedItems[0])))
            fileAttr = File.GetAttributes(selectedItems[0]);
        else
            fileAttr = File.GetAttributes(addBackSlash() + selectedItems[0]);

        if ((fileAttr & FileAttributes.Directory) == FileAttributes.Directory)
        {
            if (firstInFocus)
                filePathTextBox1.Text = addBackSlash() + selectedItems[0];
            else
                filePathTextBox2.Text = addBackSlash() + selectedItems[0];
        }
        if (firstInFocus)
            loadAction1();
        else
            loadAction2();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

private void deleteMethod()
{
    try
    {
        List<string> selectedItems = firstInFocus ? selectedItems1 :
selectedItems2;

        DialogResult dialogResult;
        if (selectedItems.Count <= 1)
            dialogResult = MessageBox.Show("Delete file: " + selectedItems[0]
+ "?", "Warning", MessageBoxButtons.YesNo);
        else
            dialogResult = MessageBox.Show("Delete files " +
selectedItems.Count + "?", "Warning", MessageBoxButtons.YesNo);
        if (dialogResult == DialogResult.Yes)
        {
            ThreadPool.QueueUserWorkItem((o) =>
            {
                string delFilePath;

                foreach (string path in selectedItems)
                {
                    if (File.Exists(Path.GetFullPath(path)))
                        delFilePath = Path.GetFullPath(path);
                    else
                        delFilePath = addBackSlash() + path;

                    FileAttributes fileAttr = File.GetAttributes(delFilePath);
                    if ((fileAttr & FileAttributes.Directory) ==
FileAttributes.Directory)

```

```

        {
            Directory.Delete(delFilePath, true);
        }
        else
        {
            File.Delete(delFilePath);
            loadAction1();
            loadAction2();
        }
    }
    loadAction1();
    loadAction2();
}, null);
}
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
}

private void copyMethod()
{
    List<string> selectedItems = firstInFocus ? selectedItems1 :
selectedItems2;

    copyItems = new List<string>();
    string copyPath;
    foreach (string item in selectedItems)
    {
        if (File.Exists(Path.GetFullPath(item)))
            copyPath = Path.GetFullPath(item);
        else
            copyPath = addBackSlash() + item;

        copyItems.Add(copyPath);
    }
    cut = false;
}

private async Task pasteMethod(ProgressForm pf)
{
    try
    {
        var tokenSource = new CancellationTokenSource();
        string selectedItem = (selectedItems1.Count == 0 &&
selectedItems2.Count == 0) ? "" :
            selectedItem = firstInFocus ? selectedItems1[0] :
selectedItems2[0];

        Console.Out.WriteLine(selectedItem);

        pf.copyPath = copyItems[0].Substring(0, copyItems[0].LastIndexOf("\\")
+ 1);

        pf.pastePath = addBackSlash() + selectedItem;
        selectedItem += "\\";

        await Task.Run(async () =>
        {
            foreach (string copyPath in copyItems)
            {
                if (File.Exists(copyPath) || Directory.Exists(copyPath))
                {
                    tokenSources.Add(tokenSource);

```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		84

```

+ tokenSources.Count);

Console.WriteLine("Розмір списку після додавання токена: "

FileInfo fileDetails = new FileInfo(copyPath);
string name = fileDetails.Name;
string ext = fileDetails.Extension;
string pastePath;

if(File.Exists(addBackSlash() + selectedItem) ||
Directory.Exists(addBackSlash() + selectedItem))
{
    if ((File.GetAttributes(addBackSlash() + selectedItem)
& FileAttributes.Directory) == FileAttributes.Directory)
    {
        pastePath = addBackSlash() + selectedItem + name;
    }
    else
    {
        pastePath = addBackSlash() + name;
    }
}
else
{
    pastePath = addBackSlash() + name;
}

FileAttributes fileAttr = File.GetAttributes(copyPath);
if ((fileAttr & FileAttributes.Directory) ==
FileAttributes.Directory)
{
    Console.WriteLine(copyPath + " " + pastePath);
    while (Directory.Exists(pastePath))
    {
        pastePath += " - copy";
    }

    await CopyFoldereAsync(pf, copyPath, pastePath,
tokenSource.Token);

    if (cut == true)
    {
        Directory.Delete(copyPath, true);
    }
    else
    {
        while (File.Exists(pastePath))
        {
            pastePath = pastePath.Substring(0,
pastePath.LastIndexOf(".")) + " - copy" + ext;
        }
        await CopyFileAsync(pf, copyPath, pastePath,
tokenSource.Token);

        if (cut == true)
        {
            File.Delete(copyPath);
        }
    }
}
}
Invoke((Action)(() => loadListView1()));
Invoke((Action)(() => loadListView2()));
});
Invoke((Action)(() => pf?.Close()));
cut = false;

```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						85
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Invoke((Action)(() => pf?.Close()));
    }
}

private static async Task CopyFoldereAsync(ProgressForm pf, string copyPath,
string pastePath, CancellationToken token, int bufferSize = 81920)
{
    Directory.CreateDirectory(pastePath);

    foreach (string dirPath in Directory.GetDirectories(copyPath, "*",
SearchOption.AllDirectories))
    {
        Directory.CreateDirectory(dirPath.Replace(copyPath, pastePath));
    }

    foreach (string newPath in Directory.GetFiles(copyPath, "*.*",
SearchOption.AllDirectories))
    {
        await CopyFileAsync(pf, newPath, newPath.Replace(copyPath, pastePath),
token);
    }
}

private static async Task CopyFileAsync(ProgressForm pf, string sourceFile,
string destinationFile, CancellationToken token, int bufferSize = 81920)
{
    using (FileStream sourceStream = new FileStream(sourceFile, FileMode.Open,
FileAccess.Read))
    using (FileStream destinationStream = new FileStream(destinationFile,
FileMode.Create, FileAccess.Write))
    {
        byte[] buffer = new byte[bufferSize];
        int bytesRead;

        while ((bytesRead = await sourceStream.ReadAsync(buffer, 0,
buffer.Length, token)) > 0)
        {
            try
            {
                pf.Invoke((Action)(() =>
pf.UpdateProgressBar(bufferSize/1000)));
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            await destinationStream.WriteAsync(buffer, 0, bytesRead, token);
            token.ThrowIfCancellationRequested();
        }
    }
}

private async Task UpdateSize(ProgressForm pf)
{
    try
    {
        long selectedSpace = 0;
        int amount = 0;
        await Task.Run(() =>
        {

```

```

        foreach (string item in copyItems)
        {
            FileInfo fileDetails = new FileInfo(item);
            if ((fileDetails.Attributes & FileAttributes.Directory) ==
FileAttributes.Directory)
            {
                DirectoryInfo dirDetails = new DirectoryInfo(item);
                try
                {
                    foreach (FileInfo f in dirDetails.GetFiles("*.*",
SearchOption.AllDirectories))
                    {
                        selectedSpace += f.Length;
                        amount++;
                    }
                }
                catch (Exception ex)
                {
                }
            }
            else
            {
                selectedSpace += fileDetails.Length;
                amount++;
            }
        }
    });
    pf.Invoke((Action)(() =>
    {
        pf.UpdateMaxProgressBar(Convert.ToInt32(selectedSpace/1000));
        pf.UpdateLabel(amount);
    }));
}
catch (Exception ex)
{
    Console.WriteLine("updatesize" + ex.Message);
}
}

public void stopThread(int id)
{
    tokenSources[id].Cancel();
    loadListView1();
    loadListView2();
}

private void createFolderMethod()
{
    string dirName = "New Folder";
    string createPath = addBackSlash() + dirName;
    int i = 1;
    while (Directory.Exists(createPath))
    {
        if (i >= 2)
        {
            dirName = dirName.Remove(dirName.Length - 4, 4);
        }
        dirName += " (" + i + ")";
        i += 1;
        createPath = addBackSlash() + dirName;
    }
    Directory.CreateDirectory(createPath);
}

```

```

        loadListView1();
        loadListView2();
    }

    private void createTxtMethod()
    {
        string fileName = "New_File";
        string createPath = addBackSlash() + fileName + ".txt";
        int i = 1;
        while (File.Exists(createPath))
        {
            if (i >= 2)
            {
                fileName = fileName.Remove(fileName.Length - 4, 4);
            }
            fileName += " (" + i + ")";
            i += 1;
            createPath = addBackSlash() + fileName + ".txt";
        }
        File.CreateText(createPath).Dispose();
        loadListView1();
        loadListView2();
    }

    private void sendMethod()
    {
        List<string> selectedItems = firstInFocus ? selectedItems1 :
selectedItems2;
        string sendPath;
        if (File.Exists(Path.GetFullPath(selectedItems[0])))
            sendPath = Path.GetFullPath(selectedItems[0]);
        else
            sendPath = addBackSlash() + selectedItems[0];

        tf = new SendForm(sendPath, Path.GetFileName(sendPath));
        tf.ShowDialog();
    }

    private void cutMethod()
    {
        List<string> selectedItems = firstInFocus ? selectedItems1 :
selectedItems2;

        copyItems = new List<string>();
        string copyPath;
        foreach (string item in selectedItems)
        {
            if (File.Exists(Path.GetFullPath(item)))
                copyPath = Path.GetFullPath(item);
            else
                copyPath = addBackSlash() + item;

            copyItems.Add(copyPath);
        }
        cut = true;
    }

    private void renameToolStripMenuItem_Click(object sender, EventArgs e)
    {
        renameMethod();
    }

    private void openToolStripMenuItem_Click(object sender, EventArgs e)
    {
        openMethod();
    }

```

```

    }

    private void deleteToolStripMenuItem_Click(object sender, EventArgs e)
    {
        deleteMethod();
    }

    private void copyToolStripMenuItem_Click(object sender, EventArgs e)
    {
        copyMethod();
    }

    private async void pasteToolStripMenuItem_Click(object sender, EventArgs e)
    {
        ProgressForm pf = new ProgressForm();
        progressForms.Add(pf);
        pf.ID = tokenSources.Count;
        pf.Show();
        await Task.WhenAll(UpdateSize(pf), Task.Run(() => pasteMethod(pf)));
        loadListView1();
        loadListView2();
    }

    private void folderToolStripMenuItem_Click(object sender, EventArgs e)
    {
        createFolderMethod();
    }

    private void txtToolStripMenuItem_Click(object sender, EventArgs e)
    {
        createTxtMethod();
    }

    private void sendToolStripMenuItem_Click(object sender, EventArgs e)
    {
        sendMethod();
    }

    private void cutToolStripMenuItem_Click(object sender, EventArgs e)
    {
        cutMethod();
    }

    private void storageMemoryToolStripMenuItem_Click(object sender, EventArgs e)
    {
        countThread = new Thread(new ThreadStart(storageMemoryCount));
        countThread.Start();
    }

    private void openButton_Click(object sender, EventArgs e)
    {
        openMethod();
    }

    private void renameButton_Click(object sender, EventArgs e)
    {
        renameMethod();
    }

    private void deleteButton_Click(object sender, EventArgs e)
    {
        deleteMethod();
    }

    private void sendButton_Click(object sender, EventArgs e)

```

```

    {
        sendMethod();
    }

private async void pasteButton_Click(object sender, EventArgs e)
{
    ProgressForm pf = new ProgressForm();
    progressForms.Add(pf);
    pf.ID = tokenSources.Count;
    pf.Show();
    await Task.WhenAll(UpdateSize(pf), Task.Run(() => pasteMethod(pf)));
    loadListView1();
    loadListView2();
}

private void copyButton_Click(object sender, EventArgs e)
{
    copyMethod();
}

private void cutButton_Click(object sender, EventArgs e)
{
    cutMethod();
}

private void copyPathButton_Click(object sender, EventArgs e)
{
    List<string> selectedItems = firstInFocus ? selectedItems1 :
selectedItems2;
    Clipboard.SetText(addBackSlash() + selectedItems[0]);
}

private void checkVolumes()
{
    foreach (var drive in DriveInfo.GetDrives())
    {
        if (comboBox1.Items.Contains(drive.Name) == false)
            comboBox1.Items.Add(drive.Name);
        if (comboBox1.Text == "")
            comboBox1.Text = comboBox1.Items[0].ToString();
        if (comboBox2.Items.Contains(drive.Name) == false)
            comboBox2.Items.Add(drive.Name);
        if (comboBox2.Text == "")
            comboBox2.Text = comboBox1.Items[0].ToString();
    }
}

private void comboBox1_ContextMenuStripChanged(object sender, EventArgs e)
{
    foreach (var drive in DriveInfo.GetDrives())
    {
        if (drive.Name == comboBox1.Text)
        {
            double freeSpace = drive.TotalFreeSpace;
            double totalSpace = drive.TotalSize;
            double percentFree = (freeSpace / totalSpace) * 100;
            percentFree = Math.Round(percentFree, 2,
MidpointRounding.AwayFromZero);
            freeSpace /= 1000;
            totalSpace /= 1000;
            float freeSpacef = (float)freeSpace;
            float totalSpacef = (float)totalSpace;
            float percentFreef = (float)percentFree;

```

```

        volumeSizeLabel1.Text = $"{freeSpacef:n0} kb of {totalSpacef:n0} k
{percentFreef}% free ";
    }
    }
    firstInFocus = true;
    filePathTextBox1.Text = comboBox1.Text;
    selectedItem1 = new List<string>();
    loadAction1();
}

private void comboBox2_ContextMenuStripChanged(object sender, EventArgs e)
{
    foreach (var drive in DriveInfo.GetDrives())
    {
        if (drive.Name == comboBox2.Text)
        {
            double freeSpace = drive.TotalFreeSpace;
            double totalSpace = drive.TotalSize;
            double percentFree = (freeSpace / totalSpace) * 100;
            percentFree = Math.Round(percentFree, 2,
MidpointRounding.AwayFromZero);
            freeSpace /= 1000;
            totalSpace /= 1000;
            float freeSpacef = (float)freeSpace;
            float totalSpacef = (float)totalSpace;
            float percentFreef = (float)percentFree;

            volumeSizeLabel2.Text = $"{freeSpacef:n0} kb of {totalSpacef:n0} k
{percentFreef}% free ";
        }
    }
    firstInFocus = false;
    filePathTextBox2.Text = comboBox2.Text;
    selectedItem2 = new List<string>();
    loadAction2();
}

private void storageMemoryCount()
{
    try
    {
        long selectedSpace = 0;
        long eachSpace = 0;
        string subOutput;
        int amount = 0;
        List<string> selectedItem = firstInFocus ? selectedItem1 :
selectedItem2;

        foreach (string item in selectedItem)
        {
            FileInfo fileDetails = new FileInfo(addBackSlash() + item);
            if ((fileDetails.Attributes & FileAttributes.Directory) ==
FileAttributes.Directory)
            {
                DirectoryInfo dirDetails = new DirectoryInfo(addBackSlash() +
item);

                try
                {
                    foreach (FileInfo f in dirDetails.GetFiles("*.*",
SearchOption.AllDirectories))
                    {
                        selectedSpace += f.Length;
                        eachSpace += f.Length;
                        amount++;
                    }
                }
            }
        }
    }
}

```

```

        subOutput = $"{eachSpace / 1000:n0} kb";
        eachSpace = 0;
    }
    catch (Exception ex)
    {
        subOutput = "no access";
    }

    Invoke((MethodInvoker)delegate
    {
        if(firstInFocus)

listView1.FindItemWithText(dirDetails.Name).SubItems[3].Text = subOutput;
        else

listView2.FindItemWithText(dirDetails.Name).SubItems[3].Text = subOutput;
    });
    }
    else
    {
        selectedSpace += fileDetails.Length;
        amount++;
    }
    Invoke((MethodInvoker)delegate
    {
        selectedItemsLabel.Text = $"{selectedSpace / 1000:n0} kb in
{amount} files";
    });
    }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

public void AddDirs(TreeNode tn)
{
    string path = tn.FullPath;
    DirectoryInfo di = new DirectoryInfo(path);
    DirectoryInfo[] diarray = { };

    try
    {
        if (di.Exists)
            diarray = di.GetDirectories();
    }
    catch
    {
        return;
    }

    foreach (DirectoryInfo d in diarray)
    {
        TreeNode tndir = new TreeNode(d.Name);
        tn.Nodes.Add(tndir);
    }
}

public void ShowDrives()
{
    treeView1.BeginUpdate();
    string[] drives = Directory.GetLogicalDrives();
    foreach (string adrive in drives)
    {

```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						92
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        TreeNode tn = new TreeNode(adrive);
        treeView1.Nodes.Add(tn);
        AddDirs(tn);
    }
    treeView1.EndUpdate();
}

private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
{
    Console.WriteLine(treeView1.SelectedNode.FullPath);
    if (firstInFocus)
    {
        filePathTextBox1.Text = treeView1.SelectedNode.FullPath;
        comboBox1.Text = Path.GetPathRoot(treeView1.SelectedNode.FullPath);
        loadAction1();
    }
    else
    {
        filePathTextBox2.Text = treeView1.SelectedNode.FullPath;
        comboBox2.Text = Path.GetPathRoot(treeView1.SelectedNode.FullPath);
        loadAction2();
    }
}

private void treeView1_BeforeExpand(object sender, TreeViewCancelEventArgs e)
{
    treeView1.BeginUpdate();
    foreach (TreeNode tn in e.Node.Nodes)
    {
        AddDirs(tn);
    }
    treeView1.EndUpdate();
}

private void listView1_ColumnClick(object sender, ColumnClickEventArgs e)
{
    try
    {
        this.listView1.ListViewItemSorter = lvwColumnSorter;
        if (e.Column == lvwColumnSorter.SortColumn)
        {
            // Змінити поточний напрямок сортування для цього стовпця.
            if (lvwColumnSorter.Order == SortOrder.Ascending)
            {
                lvwColumnSorter.Order = SortOrder.Descending;
            }
            else
            {
                lvwColumnSorter.Order = SortOrder.Ascending;
            }
        }
        else
        {
            // Встановити номер стовпця, який потрібно відсортувати; за
            // замовчуванням у порядку зростання.
            lvwColumnSorter.SortColumn = e.Column;
            lvwColumnSorter.Order = SortOrder.Ascending;

            if (e.Column == 3)
            {
                lvwColumnSorter.Date = false;
                lvwColumnSorter.Integer = true;
            }
            if (e.Column == 1)
            {
                lvwColumnSorter.Integer = false;
                lvwColumnSorter.Date = true;
            }
        }
    }
}

```

```

        // Виконати сортування за допомогою цих нових параметрів сортування.
        this.listView1.Sort();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

private void listView2_ColumnClick(object sender, ColumnClickEventArgs e)
{
    try
    {
        this.listView2.ListViewItemSorter = lvwColumnSorter;
        if (e.Column == lvwColumnSorter.SortColumn)
        {
            // Змінити поточний напрямок сортування для цього стовпця.
            if (lvwColumnSorter.Order == SortOrder.Ascending)
            {
                lvwColumnSorter.Order = SortOrder.Descending;
            }
            else
            {
                lvwColumnSorter.Order = SortOrder.Ascending;
            }
        }
        else
        {
            // Встановити номер стовпця, який потрібно відсортувати; за
            // замовчуванням у порядку зростання.
            lvwColumnSorter.SortColumn = e.Column;
            lvwColumnSorter.Order = SortOrder.Ascending;

            if (e.Column == 3)
            {
                lvwColumnSorter.Date = false;
                lvwColumnSorter.Integer = true;
            }
            if (e.Column == 1)
            {
                lvwColumnSorter.Integer = false;
                lvwColumnSorter.Date = true;
            }
        }

        // Виконати сортування за допомогою цих нових параметрів сортування.
        this.listView2.Sort();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

private void listView1_DrawColumnHeader(object sender,
DrawListViewColumnHeaderEventArgs e)
{
    try
    {
        SolidBrush dark = new SolidBrush(Color.FromArgb(255, 255, 255));
        SolidBrush light = new SolidBrush(Color.FromArgb(47, 54, 64));

        var uiMode = ConfigurationManager.AppSettings["UIMode"];
        if (uiMode == "light")
        {
            e.Graphics.FillRectangle(light, e.Bounds);
            e.Graphics.DrawString(e.Header.Text, e.Font, dark, e.Bounds);
        }
    }
}

```

```

        else
        {
            e.Graphics.FillRectangle(dark, e.Bounds);
            e.Graphics.DrawString(e.Header.Text, e.Font, light, e.Bounds);
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

private void listView1_DrawItem(object sender, DrawListViewItemEventArgs e)
{
    e.DrawDefault = true;
}

private void listView2_DrawColumnHeader(object sender,
DrawListViewColumnHeaderEventArgs e)
{
    try
    {
        SolidBrush dark = new SolidBrush(Color.FromArgb(255, 255, 255));
        SolidBrush light = new SolidBrush(Color.FromArgb(47, 54, 64));

        var uiMode = ConfigurationManager.AppSettings["UIMode"];
        if (uiMode == "light")
        {
            e.Graphics.FillRectangle(light, e.Bounds);
            e.Graphics.DrawString(e.Header.Text, e.Font, dark, e.Bounds);
        }
        else
        {
            e.Graphics.FillRectangle(dark, e.Bounds);
            e.Graphics.DrawString(e.Header.Text, e.Font, light, e.Bounds);
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

private void listView2_DrawItem(object sender, DrawListViewItemEventArgs e)
{
    e.DrawDefault = true;
}

// Пошук
private async void searchTextBox_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
    {
        searchTokenSource?.Cancel();
        searchTokenSource = new CancellationTokensource();
        var token = searchTokenSource.Token;
        try
        {
            string searchQuery = searchTextBox.Text;
            string directory = firstInFocus ? comboBox1.Text : comboBox2.Text;
            Console.WriteLine(directory);

            if (string.IsNullOrEmpty(searchQuery) ||
string.IsNullOrEmpty(directory))

```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						95
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        {
            MessageBox.Show("Будь ласка, введіть пошуковий запит та  

виберіть директорію.");
            return;
        }

        bool firstSearchWin = firstInFocus;
        // Асинхронний пошук файлів
        List<string> foundFiles = await Task.Run(() =>
SearchFilesAsync(directory, searchQuery, token));

        await DisplaySearchResultsAsync(foundFiles, firstSearchWin,
token);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

private async Task DisplaySearchResultsAsync(List<string> foundFiles, bool
firstSearchWin, CancellationToken token)
{
    // Очищення ListView перед оновленням результатів
    if (firstSearchWin)
    {
        listView1.Items.Clear();
        listView1.BeginUpdate();
    }
    else
    {
        listView2.Items.Clear();
        listView2.BeginUpdate();
    }

    // Додавання результатів у ListView
    foreach (var file in foundFiles)
    {
        await Task.Run(() =>
        {
            ListViewItem item = CreateListViewItem(file);
            if (firstSearchWin)
                listView1.Invoke((Action)(() => listView1.Items.Add(item)));
            else
                listView2.Invoke((Action)(() => listView2.Items.Add(item)));
        });
    }
    if (firstSearchWin)
        listView1.EndUpdate();
    else
        listView2.EndUpdate();
}

// Метод створення елемента ListViewItem для відображення в ListView
private ListViewItem CreateListViewItem(string file)
{
    string subOutput, date;
    Icon fileIcon = SystemIcons.WinLogo;
    ListViewItem item = new ListViewItem(Path.GetFullPath(file));

    if ((File.GetAttributes(file) & FileAttributes.Directory) ==
FileAttributes.Directory)
    {
        DirectoryInfo di = new DirectoryInfo(file);

```

```

        date = di.LastWriteTime.ToString("dd.MM.yyyy HH:mm");
        subOutput = " ";
        item.ImageIndex = 0;
        item.SubItems.Add(di.LastWriteTime.ToString());
        item.SubItems.Add("directory");
        item.SubItems.Add(" ");
    }
    else
    {
        FileInfo fi = new FileInfo(file);
        if (!imageList1.Images.ContainsKey(fi.Extension))
        {
            fileIcon = System.Drawing.Icon.ExtractAssociatedIcon(fi.FullName);
            Invoke((Action)(() => imageList1.Images.Add(fi.Extension,
fileIcon))));
        }
        date = fi.LastWriteTime.ToString("dd.MM.yyyy HH:mm");
        subOutput = $"{fi.Length / 1000:n0} kb";
        item.ImageKey = fi.Extension;
        item.SubItems.Add(date);
        item.SubItems.Add(MimeTypes.GetContentType(fi.FullName));
        item.SubItems.Add(subOutput);
    }

    return item;
}

private static async Task<List<string>> SearchFilesAsync(string directory,
string searchQuery, CancellationToken token)
{
    List<string> foundFiles = new List<string>();

    try
    {
        if (!Directory.Exists(directory))
        {
            Console.WriteLine($"Директорії {directory} не існує.");
            return foundFiles;
        }

        // Додаємо файли з доступом у поточній директорії
        foundFiles.AddRange(await Task.Run(() => GetAccessibleFiles(directory,
searchQuery, token)));
        var drives = DriveInfo.GetDrives();
        // Пошук файлів у підпапках з перевіркою доступу
        var accessibleDirectories = await Task.Run(() =>
GetAccessibleDirectories(directory, token));
        var tasks = accessibleDirectories.Select(subDirectory =>
SearchFilesAsync(subDirectory, searchQuery, token));
        var results = await Task.WhenAll(tasks);

        // Додаємо результати з підкаталогів
        foreach (var result in results)
        {
            foundFiles.AddRange(result);
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Помилка в каталозі {directory}: {ex.Message}");
    }

    return foundFiles;
}

```

```

    /// <summary>
    /// Отримує файли у папці, до яких є доступ, і які відповідають пошуковому
запиту.
    /// </summary>
    private static IEnumerable<string> GetAccessibleFiles(string directory, string
searchQuery, CancellationToken token)
    {
        try
        {
            return Directory.GetFiles(directory, "*",
SearchOption.TopDirectoryOnly)
                .Where(f => Path.GetFileName(f).IndexOf(searchQuery,
StringComparison.OrdinalIgnoreCase) >= 0);
        }
        catch (UnauthorizedAccessException)
        {
            Console.WriteLine($"Немає доступу до файлів у папці {directory}");
            return Enumerable.Empty<string>();
        }
        catch (System.Security.SecurityException)
        {
            Console.WriteLine($"Безпекова помилка при доступі до файлів у папці
{directory}");
            return Enumerable.Empty<string>();
        }
    }

    /// <summary>
    /// Отримує підкаталоги, до яких є доступ.
    /// </summary>
    private static List<string> GetAccessibleDirectories(string directory,
CancellationToken token)
    {
        List<string> accessibleDirectories = new List<string>();
        try
        {
            var directories = Directory.GetDirectories(directory, "*",
SearchOption.TopDirectoryOnly);

            foreach (var dir in directories)
            {
                token.ThrowIfCancellationRequested(); // Перевірка на скасування
                accessibleDirectories.Add(dir);
            }
        }
        catch (OperationCanceledException)
        {
            Console.WriteLine("Пошук скасовано.");
        }
        catch (UnauthorizedAccessException)
        {
            Console.WriteLine($"Немає доступу до підкаталогів у {directory}");
        }
        catch (System.Security.SecurityException)
        {
            Console.WriteLine($"Безпекова помилка при доступі до підкаталогів у
{directory}");
        }

        return accessibleDirectories;
    }
}

```

Додаток Б

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						98
Змн.	Арк.	№ докум.	Підпис	Дата		

```

using System;
using System.Configuration;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Net.NetworkInformation;
using System.Threading;

namespace FileExplorer
{
    public partial class SendForm : Form
    {
        public SendForm(string path, string name)
        {
            InitializeComponent();
            InitializeUI("UIMode");
            fileNameLabel.Text = path;
            fileNameLabel.Tag = name;
        }

        private string IP = "127.0.0.1";
        TcpListener tcplistener;
        TcpClient tcpclient;
        Socket clientSocket;
        private Thread serverThread;
        private Thread findPC;
        private Thread notification;
        int flag = 0;
        string fileName = "";
        private bool serverRunning = false;
        private bool isConnected = false;
        int x = 9;
        int y = 308;
        int fileReceived = 0;
        string savePath = "";
        string senderIP;
        string senderName;
        string receiverIP;
        string receiverName;

        private void mainForm_Load(object sender, EventArgs e)
        {
            notificationLabel.ForeColor = Color.Red;
            notificationLabel.Text = "Offline";
        }

        void startServer()
        {
            try
            {
                serverRunning = true;
                tcplistener = new TcpListener(IPAddress.Parse(IP), 11000);
                tcplistener.Start();
                serverThread = new Thread(new ThreadStart(serverTasks));
                serverThread.Start();
                while (!serverThread.IsAlive) ;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}

```

```

    }
}

void serverTasks()
{
    try
    {
        while (true)
        {
            if (fileReceived == 1)
            {
                if (MessageBox.Show("Save File?", "File received",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.No)
                {
                    File.Delete(savePath);
                    fileReceived = 0;
                }
                else
                {
                    fileReceived = 0;
                }
            }

            tcpclient = tcplistener.AcceptTcpClient();
            Invoke((MethodInvoker)delegate
            {
                fileNotificationLabel.Text = "File Coming from " + senderIP +
" " + senderName;
            });
            isConnected = true;
            NetworkStream stream = tcpclient.GetStream();
            if (flag == 1 && isConnected)
            {
                savePath = savePathLabel.Text + "\\ " + fileName;
                using (var output = File.Create(savePath))
                {
                    var buffer = new byte[1024];
                    int bytesRead;
                    while ((bytesRead = stream.Read(buffer, 0, buffer.Length))
> 0)
                    {
                        output.Write(buffer, 0, bytesRead);
                    }
                    flag = 0;
                    tcpclient.Close();
                    isConnected = false;
                    fileName = "";
                    Invoke((MethodInvoker)delegate
                    {
                        fileNotificationLabel.Text = "";
                    });
                    fileReceived = 1;
                }
            }
            else if (flag == 0 && isConnected)
            {
                Byte[] bytes = new Byte[256];
                String data = null;
                int i;
                while ((i = stream.Read(bytes, 0, bytes.Length)) != 0)
                {
                    data = System.Text.Encoding.ASCII.GetString(bytes, 0, i);
                }
                string[] msg = data.Split('@');
                fileName = msg[0];
            }
        }
    }
}

```

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		100

```

        senderIP = msg[1];
        senderName = msg[2];
        tcpclient.Close();
        isConnected = false;
        flag = 1;
    }
}
}
catch (Exception)
{
    flag = 0;
    isConnected = false;
    if (tcpclient != null)
        tcpclient.Close();
}
}

private void startButton_Click(object sender, EventArgs e)
{
    ipBox.Text = "";
    onlinePCList.Items.Clear();
    notificationLabel.ForeColor = Color.Green;
    notificationLabel.Text = "Finding...";
    try
    {
        findPC = new Thread(new ThreadStart(searchPC));
        findPC.Start();
        while (!findPC.IsAlive) ;
    }
    catch (Exception e1)
    {
        MessageBox.Show(e1.Message);
    }
}

void searchPC()
{
    bool isNetworkUp = NetworkInterface.GetIsNetworkAvailable();
    if (isNetworkUp)
    {
        var host = Dns.GetHostEntry(Dns.GetHostName());
        foreach (var ip in host.AddressList)
        {
            if (ip.AddressFamily == AddressFamily.InterNetwork)
            {
                this.IP = ip.ToString();
                Console.WriteLine(this.IP);
            }
        }
        Invoke((MethodInvoker)delegate
        {
            infoLabel.Text = "This Computer: " + this.IP;
        });
        string[] ipRange = IP.Split('.');
        for (int i = 100; i < 255; i++)
        {
            Ping ping = new Ping();
            string testIP = ipRange[0] + '.' + ipRange[1] + '.' + ipRange[2] +
            '.' + i.ToString();
            if (testIP != this.IP)
            {
                ping.PingCompleted += new
                PingCompletedEventHandler(pingCompletedEvent);
                ping.SendAsync(testIP, 100, testIP);
            }
        }
    }
}

```

```

        Invoke((MethodInvoker)delegate
        {
            notificationLabel.ForeColor = Color.Green;
            notificationLabel.Text = "Online";
        });

        if (!serverRunning)
            startServer();
    }
    else
    {
        Invoke((MethodInvoker)delegate
        {
            notificationLabel.ForeColor = Color.Red;
            notificationLabel.Text = "Offline";
        });
        MessageBox.Show("No connection to LAN");
    }
}

void pingCompletedEvent(object sender, PingCompletedEventArgs e)
{
    string ip = (string)e.UserState;
    if (e.Reply.Status == IPStatus.Success)
    {
        string name;
        try
        {
            IPEndPoint hostEntry = Dns.GetHostEntry(ip);
            name = hostEntry.HostName;
        }
        catch (SocketException ex)
        {
            name = ex.Message;
        }
        Invoke((MethodInvoker)delegate
        {
            ListViewItem item = new ListViewItem();
            item.Text = ip;
            item.SubItems.Add(name);
            onlinePCList.Items.Add(item);
        });
    }
}

private void browseButton_Click(object sender, EventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog();
    openFileDialog.Filter = "All Files|*.*";
    openFileDialog.Title = "Select a File";
    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        fileNameLabel.Text = openFileDialog.FileName;
        fileNameLabel.Tag = openFileDialog.SafeFileName;
    }
    timer1.Start();
}

private void sendFileButton_Click(object sender, EventArgs e)
{
    receiverIP = null;
    receiverName = null;
    if ((onlinePCList.SelectedIndices.Count > 0 || ipBox.Text != "") &&
        serverRunning && fileNameLabel.Text != ".")
    {

```

```

        if (ipBox.Text != "")
        {
            receiverIP = ipBox.Text;
            receiverName = "";
        }
        else
        {
            receiverIP = onlinePCList.SelectedItems[0].Text;
            receiverName = onlinePCList.SelectedItems[0].SubItems[1].Text;
        }
        try
        {
            Ping p = new Ping();
            PingReply r;
            r = p.Send(receiverIP);
            if (!(r.Status == IPStatus.Success))
            {
                MessageBox.Show("Target computer is not available.", "",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            else
            {
                notification = new Thread(new ThreadStart(showNotification));
                notification.Start();
                fileNotificationLabel.Text = "Please don't do other tasks.
                File sending to " + receiverIP + " " + receiverName + "...";
                tcplistener.Stop();
                serverThread.Abort();
                serverThread.Join();
                serverRunning = false;
                clientSocket = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
                clientSocket.Connect(new
                IPEndPoint(IPAddress.Parse(receiverIP), 11000));
                string fileName = fileNameLabel.Tag.ToString();
                byte[] fileNameData = Encoding.Default.GetBytes(fileName + "@"
                + this.IP + "@" + Environment.MachineName);
                clientSocket.Send(fileNameData);
                clientSocket.Shutdown(SocketShutdown.Both);
                clientSocket.Close();
                clientSocket = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
                clientSocket.Connect(new
                IPEndPoint(IPAddress.Parse(receiverIP), 11000));
                clientSocket.SendFile(fileNameLabel.Text);
                clientSocket.Shutdown(SocketShutdown.Both);
                clientSocket.Close();
                Invoke((MethodInvoker)delegate
                {
                    });
                MessageBox.Show("File sent to " + receiverIP + " " +
                receiverName, "Confirmation", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
            if (clientSocket != null)
            {
                clientSocket.Shutdown(SocketShutdown.Both);
                clientSocket.Close();
            }
        }
        finally

```

```

        {
            for (int i = 0; i < onlinePCList.SelectedIndices.Count; i++)
            {
onlinePCList.Items[this.onlinePCList.SelectedIndices[i]].Selected = false;
            }
            fileNotificationLabel.Text = ".";
            startServer();
        }
    }
}
void showNotification()
{
}
private void mainForm_FormClosing(object sender, FormClosingEventArgs e)
{
    if (serverRunning)
    {
        tcplistener.Stop();
        if (serverThread != null)
        {
            serverThread.Abort();
            serverThread.Join();
        }
    }
}

private void stopButton_Click(object sender, EventArgs e)
{
    if (serverRunning)
    {
        serverRunning = false;
        onlinePCList.Items.Clear();
        if (tcplistener != null)
            tcplistener.Stop();
        if (serverThread != null)
        {
            serverThread.Abort();
            serverThread.Join();
        }

        notificationLabel.ForeColor = Color.Red;
        notificationLabel.Text = "Offline";
        infoLabel.Text = "";
        fileNameLabel.Text = ".";
    }
}

private void timer1_Tick(object sender, EventArgs e)
{
    x = x - 5;
    fileNameLabel.Location = new Point(x, y);
    if (x < (fileNameLabel.Text.Length * (-1)))
        x = 545;
}

private void clearButton_Click(object sender, EventArgs e)
{
    fileNameLabel.Text = ".";
    timer1.Stop();
}

private void changeSaveLocButton_Click(object sender, EventArgs e)

```

```

{
    FolderBrowserDialog browse = new FolderBrowserDialog();
    if (browse.ShowDialog() == DialogResult.OK)
    {
        string savePath = browse.SelectedPath;
        savePathLabel.Text = savePath;
    }
}

private void exitButton_Click(object sender, EventArgs e)
{
    if (serverRunning)
    {
        if (tcplistener != null)
            tcplistener.Stop();
        if (serverThread != null)
        {
            serverThread.Abort();
            serverThread.Join();
        }
    }
    this.Close();
}

private void InitializeUI(string key)
{
    try
    {
        var uiMode = ConfigurationManager.AppSettings[key];
        if (uiMode == "light")
        {
            exitButton.ForeColor = Color.FromArgb(245, 246, 250);
            exitButton.BackColor = Color.FromArgb(47, 54, 64);
            changeSaveLocButton.ForeColor = Color.FromArgb(245, 246, 250);
            changeSaveLocButton.BackColor = Color.FromArgb(47, 54, 64);
            browseButton.ForeColor = Color.FromArgb(245, 246, 250);
            browseButton.BackColor = Color.FromArgb(47, 54, 64);
            clearButton.ForeColor = Color.FromArgb(245, 246, 250);
            clearButton.BackColor = Color.FromArgb(47, 54, 64);
            stopButton.ForeColor = Color.FromArgb(245, 246, 250);
            stopButton.BackColor = Color.FromArgb(47, 54, 64);
            startButton.ForeColor = Color.FromArgb(245, 246, 250);
            startButton.BackColor = Color.FromArgb(47, 54, 64);
            sendFileButton.ForeColor = Color.FromArgb(245, 246, 250);
            sendFileButton.BackColor = Color.FromArgb(47, 54, 64);
            this.ForeColor = Color.FromArgb(245, 246, 250);
            this.BackColor = Color.FromArgb(47, 54, 64);
            onlinePCList.ForeColor = Color.FromArgb(245, 246, 250);
            onlinePCList.BackColor = Color.FromArgb(47, 54, 64);
            ConfigurationManager.AppSettings[key] = "light";
        }
        else
        {
            exitButton.ForeColor = Color.FromArgb(47, 54, 64);
            exitButton.BackColor = Color.FromArgb(245, 246, 250);
            changeSaveLocButton.ForeColor = Color.FromArgb(47, 54, 64);
            changeSaveLocButton.BackColor = Color.FromArgb(245, 246, 250);
            browseButton.ForeColor = Color.FromArgb(47, 54, 64);
            browseButton.BackColor = Color.FromArgb(245, 246, 250);
            clearButton.ForeColor = Color.FromArgb(47, 54, 64);
            clearButton.BackColor = Color.FromArgb(245, 246, 250);
            stopButton.ForeColor = Color.FromArgb(47, 54, 64);
            stopButton.BackColor = Color.FromArgb(245, 246, 250);
            startButton.ForeColor = Color.FromArgb(47, 54, 64);
        }
    }
}

```

```

        startButton.BackColor = Color.FromArgb(245, 246, 250);
        sendFileButton.ForeColor = Color.FromArgb(47, 54, 64);
        sendFileButton.BackColor = Color.FromArgb(245, 246, 250);
        this.ForeColor = Color.FromArgb(47, 54, 64);
        this.BackColor = Color.FromArgb(245, 246, 250);
        onlinePCList.ForeColor = Color.FromArgb(47, 54, 64);
        onlinePCList.BackColor = Color.FromArgb(245, 246, 250);
        ConfigurationManager.AppSettings[key] = "dark";
    }
}
catch (Exception ex)
{
    throw;
}
}

private void savePathLabel_Click(object sender, EventArgs e)
{
}

private void onlinePCList_SelectedIndexChanged(object sender, EventArgs e)
{
}

private void timer2_Tick(object sender, EventArgs e)
{
}

protected override void OnFormClosing(FormClosingEventArgs e)
{
    if (serverRunning)
    {
        if (tcplistener != null)
            tcplistener.Stop();
        if (serverThread != null)
        {
            serverThread.Abort();
            serverThread.Join();
        }
    }
}
}
}
}

```