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

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ПРОГРАМИ ДЛЯ АВТОМАТИЧНОГО СТВОРЕННЯ РЕЗЕРВНИХ КОПІЙ ФАЙЛІВ З ВИКОРИСТАННЯМ C# ТА .NET

DEVELOPMENT OF A PROGRAM FOR AUTOMATIC FILE BACKUP CREATION USING C# AND .NET

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗс-21
Бєлов Дмитро Миколайович

Керівник:
Ящук Андрій Анатолійович

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«20» 12 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Бєлову Дмитру Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка програми для автоматичного створення резервних копій файлів з використанням C# та .NET»*

Керівник роботи: *к.т.н., доцент Ящук А. А.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025р.*

3. Вихідні дані до роботи: *Visual Studio, C#, .NET, WPF, Google Drive, методичні вказівки до виконання кваліфікаційної роботи бакалавра*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз існуючих анлогів, формування вимог, вибір інструментів для розробки, створення зручного і зрозумілого інтерфейсу, резервне копіювання за розкладом, вибір шляху збереження, базове шифрування, логування та відновлення, тестування програми, створення користувацької документації*

5. Перелік графічного матеріалу: *13 рисунків, 1 таблиця, 4 додатків*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Ящук А. А.*		
Специфікація вимог до розробленої системи	*Ящук А. А.*		
Розробка об'єкта проектування	*Ящук А. А.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	34 %		
Академічна доброчесність	*Ящук А. А.*		

7.Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	Виконано
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	Виконано
3	Обгрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	Виконано
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	Виконано
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	Виконано
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	Виконано
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	Виконано

Здобувач вищої освіти _____ Дмитро БЄЛОВ

Керівник кваліфікаційної роботи _____ Андрій ЯЩУК

АНОТАЦІЯ

Бєлов Д. М. Розробка програми для автоматичного створення резервних копій файлів з використанням C# та .NET. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі здійснено аналіз стану проблеми і сформульовано завдання. В другому розділі здійснено аналіз вимог та вибір технологій. У третьому розділі описана розробка додатка. У висновках представлено результати роботи.

Ключові слова: C#, .NET, резервне копіювання, файли, автоматизація, програмне забезпечення, захист даних, архівування, збереження даних.

ABSTRACT

Bielov D. Development of a Program for Automatic File Backup Creation Using C# and .NET. Manuscript. Qualification work of the bachelor's program "Software engineering" in the specialty "Software engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices.

The first chapter presents an analysis of the problem was conducted and formulates the objectives. The second chapter requirements were analyzed and technologies were selected. The third chapter describes the application's development. The conclusions present the results of the work.

Keywords: C#, .NET, backup, files, automation, software, data protection, archiving, data storage.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних інформаційних системах обсяги цифрових даних безперервно зростають, що робить завдання їх надійного збереження вкрай актуальним. Непередбачувані збої апаратного забезпечення, шкідливе програмне забезпечення чи випадкові помилки користувачів можуть спричинити втрату важливої інформації, що призводить до серйозних наслідків. Тому резервне копіювання файлів є необхідним засобом захисту даних. Автоматизація процесу створення резервних копій дозволяє суттєво знизити ймовірність втрат інформації та забезпечити регулярне збереження даних без додаткового залучення користувача.

Незважаючи на наявність різних програмних засобів для резервного копіювання, потреба у нових ефективних рішеннях залишається високою. Сучасні тенденції в IT-сфері включають активне використання хмарних технологій і віддалених сховищ для зберігання даних, але локальні автоматизовані інструменти також мають важливе значення. Зростання кількості кібератак типу шифрувальників (ransomware) додатково підкреслює необхідність регулярного резервного копіювання файлів. Крім того, багато існуючих програм резервного копіювання є складними у налаштуванні або призначені переважно для корпоративних клієнтів, що створює потребу у простому та ефективному засобі автоматизації цього процесу.

Метою цієї кваліфікаційної роботи є розробка програмного продукту – десктопної програми для автоматичного створення резервних копій файлів із використанням мови програмування C# та платформи .NET, яка поєднує функціональність, безпеку та зручність використання.

Об’єктом кваліфікаційної роботи є процеси захисту інформації шляхом резервного копіювання у програмному середовищі.

Предметом кваліфікаційної роботи є програмні засоби автоматичного створення резервних копій файлів, реалізовані за допомогою технологій C# та .NET Framework.

Для реалізації поставленої мети у рамках кваліфікаційної роботи необхідно вирішити такі завдання:

– виконати аналіз сучасних програмних засобів для резервного копіювання за розкладом, модулі роботи з Google API, шифрування, інтерфейс та логування;

– сформулювати функціональні та нефункціональні вимоги до системи;

– обґрунтувати вибір мови програмування, бібліотек, API та середовища розробки;

– спроєктувати архітектуру програмного продукту з урахуванням масштабованості та захисту даних;

– створити зручний і зрозумілий інтерфейс для взаємодії з користувачем;

– реалізувати основний функціонал: створення резервних копій за розкладом, збереження копій на локальному диску та в Google Drive, базове шифрування файлів;

– протестувати програму на коректність роботи, продуктивність та стійкість до збоїв;

– розробити користувацьку документацію.

Результатом кваліфікаційної роботи є повнофункціональний застосунок, який забезпечує автоматичне резервне копіювання файлів, з можливістю шифрування, збереження у хмарному та локальному сховищах, що може бути використаний як у домашньому, так і в офісному середовищі.

РОЗДІЛ 1
АНАЛІЗ СИСТЕМ РЕЗЕРВНОГО КОПІЮВАННЯ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Захист даних від втрати при збої апаратури або через віруси/атакуючих – актуальне завдання. Існує багато безкоштовних програм для резервного копіювання, але кожна має свої особливості та обмеження. Наприклад, «Macrium Reflect Free» [1] добре зарекомендувала себе як інструмент клонування дисків та створення образів системи. Її інтерфейс не перевантажений зайвими налаштуваннями, що полегшує використання (рис. 1.1).

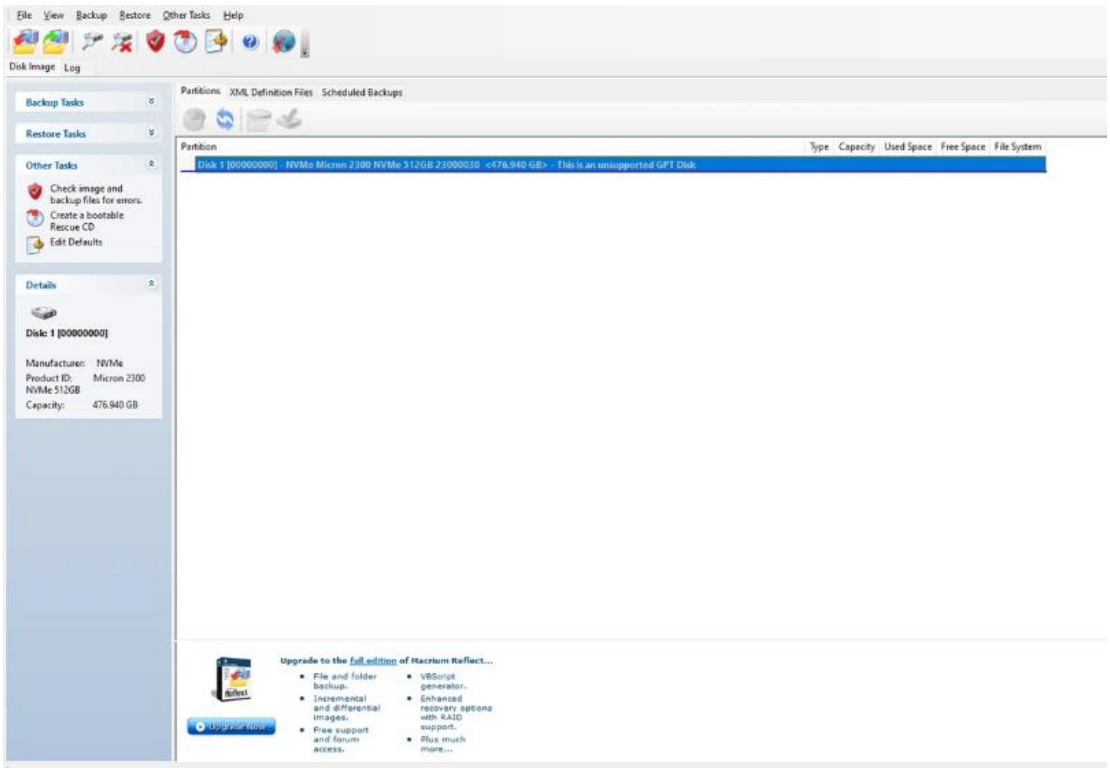

Рисунок 1.1 – Інтерфейс «Macrium Reflect Free» [1]

Проте безкоштовна версія Macrium Reflect знята з підтримки (2023 рік), тому вона менш придатна для нових проєктів.

Існують аналоги програм для резервного копіювання.

«EaseUS Todo Backup Free» [2] – відома утиліта для побайтового й файлового бекапу. Безкоштовна версія дозволяє робити повний, інкрементний і диференційний бекап, зберігати кілька версій файлів і навіть завантажувати копії у хмару. Інтерфейс програми інтуїтивний – функція відновлення здійснюється через змонтований віртуальний диск, що полегшує відновлення файлів (рис. 1.2).

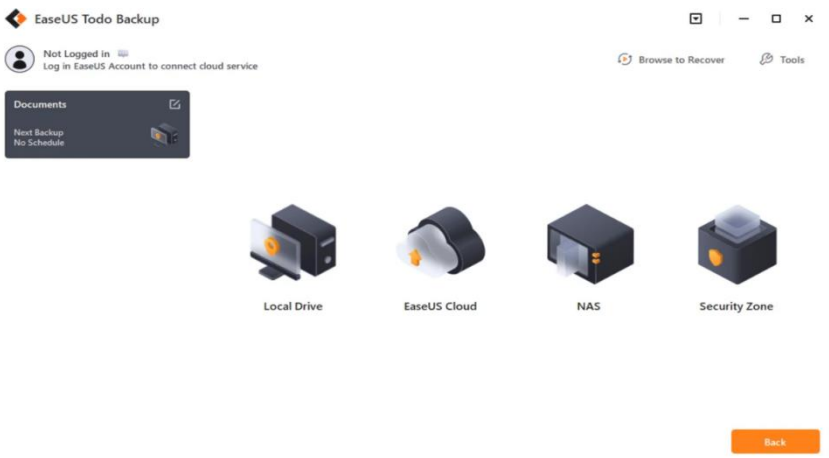

Рисунок 1.2 – Інтерфейс «EaseUS Todo Backup Free» [2]

Однак у безкоштовній версії деякі розширені опції заблоковані – наприклад, обрані режими стиснення та інші функції доступні лише у платній редакції.

«AOMEI Backupper Standard» [3] має більш сучасний дизайн (рис. 1.3).

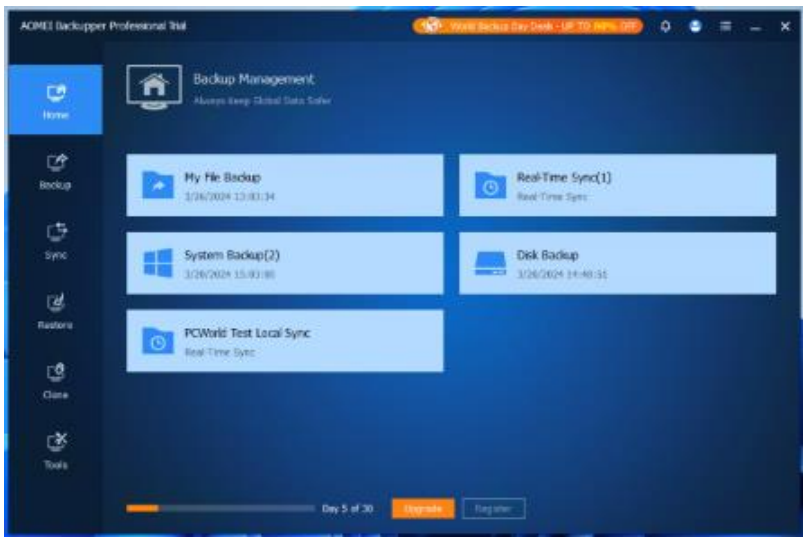

Рисунок 1.3 – Інтерфейс «AOMEI Backupper Standard» [3]

Ця програма підтримує резервне копіювання дисків, розділів або окремих файлів з використанням Volume Shadow Copy Service (щоб бекапити «гарячі» файли без перезавантаження). Доступні повний, інкрементний та диференційний режими копіювання. Можна клонувати диски й виконувати синхронізацію папок. Безкоштовна версія навіть має шифрування резервних копій (AES).

«Paragon Backup & Recovery (Community Edition)» [4] орієнтований переважно на файлове резервне копіювання (рис. 1.4).

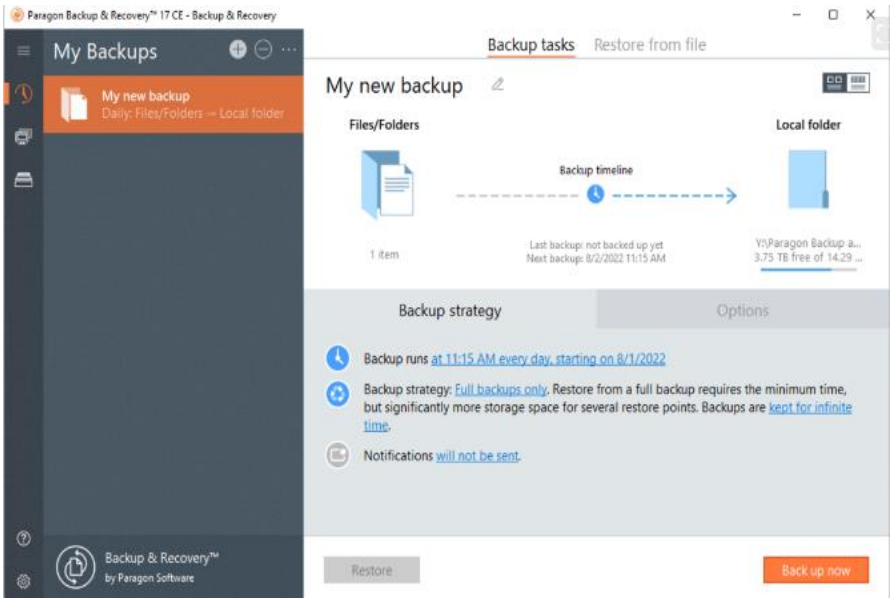

Рисунок 1.4 – Інтерфейс «Paragon Backup & Recovery (Community Edition)» [4]

Утиліта підтримує створення архівів ZIP та віртуальних дисків VHD/VHDX з повним, інкрементним і диференційним бекапом. За допомогою майстра можна налаштувати розклад (щодня, щомісяця або за подією), встановити ступінь стиснення і пароль до архіву. На відміну від інших, Paragon CE не дозволяє клонування дисків/розділів (ця функція є тільки в платній Hard Disk Manager). Також відсутня інтеграція з хмарними сховищами усі резервні копії зберігаються лише на локальному або мережевому носії. Програма підтримує інтерфейс українською мовою.

«FreeFileSync» [5] – це безкоштовний open-source інструмент для синхронізації папок (рис. 1.5).

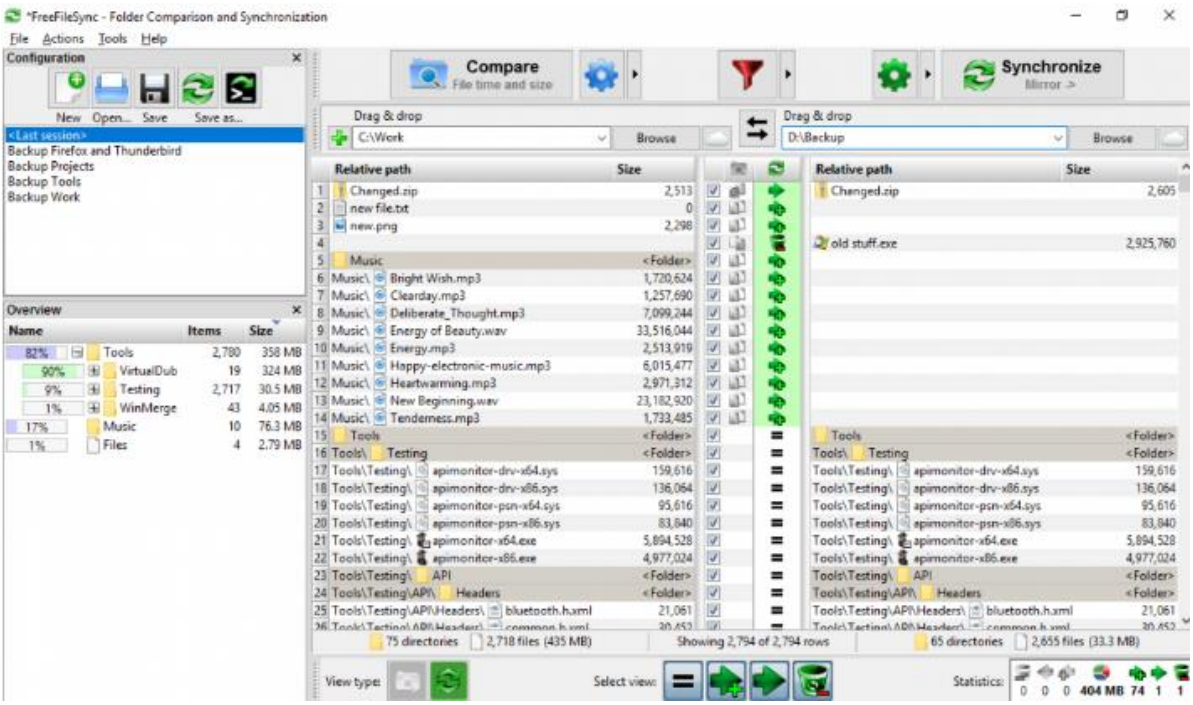

Рисунок 1.5 – Інтерфейс «FreeFileSync» [5]

Він працює на Windows, macOS та Linux і ефективно копіює лише ті файли, що змінилися. Це швидке рішення для зеркалювання даних, але важливо розуміти: «FreeFileSync» не створює образи дисків і не має вбудованого планувальника. Щоб автоматизувати роботу, користувач налаштовує завдання ОС (наприклад, за допомогою Планувальника Windows). Також «FreeFileSync» не забезпечує шифрування – він розрахований на просту синхронізацію, а не на безпечне архівування.

Аналіз програм-аналогів показує: кожне з існуючих рішень покриває деякі потреби резервного копіювання, але жодне не містить усіх бажаних можливостей одночасно. Так, деякі утиліти («AOMEI», «EaseUS») вже підтримують шифрування та бекап за розкладом, але у них є обмеження у безкоштовних версіях. Інші програми («Paragon», «Macrium») надають надійний бекап, проте без інтеграції з хмарою чи зручності сучасного UI. «FreeFileSync» дозволяє швидко синхронізувати файли, проте не автоматизує процес резервування і не шифрує дані. Наявний портфель інструментів резервного копіювання вимагає доповнення – нової програми, яка б поєднала всі ключові функції.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Для досягнення поставленої мети розробки програми резервного копіювання з використанням C# та WPF було здійснено аналіз аналогів і сформовані наступні завдання:

– провести аналіз існуючих програм-аналогів та перспектив для розвитку;

– сформулювати функціональні та нефункціональні вимоги до розроблюваної системи;

– обґрунтувати вибір мови програмування, бібліотек, API та середовища розробки. Моїм вибором мови програмування став C# та платформа .NET, бо вони поєднають функціональність, безпеку, зручність використання та легке масштабування на різні девайси;

– спроєктувати архітектуру програмного продукту з урахуванням масштабованості та захисту даних;

– створити зручний і зрозумілий інтерфейс для взаємодії з користувачем використовуючи XAML в WPF. Інтерфейс програми має бути простим і інтуїтивним для середньостатистичного користувача;

– реалізувати основний функціонал: програма повинна автоматично виконувати бекап у встановлені користувачем часи, резервні копії мають зберігатися як на локальному диску, так і на хмарному сховищі (Google Drive), дані резервної копії необхідно захищати, програма повинна фіксувати результати кожного бекапу (журнали) і забезпечувати просте відновлення файлів з резервних копій;

– протестувати програму на коректність роботи, продуктивність та стійкість до збоїв;

– розробити користувацьку документацію.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Було зроблено аналіз вимог до програми, результат аналізу наведено нижче з категорією вимог та коротким описом.

Нижче наведені функціональні вимоги до програми:

– можливість створення резервних копій вибраних файлів та папок (ручний запуск бекапу);

– налаштування графіку автоматичного резервного копіювання (щоденне/щотижневе/за іншим розкладом);

– вибір місця збереження резервних копій: локальний диск або також Google Drive;

– шифрування файла або архіву з використанням пароля (наприклад, AES-256) для захисту даних;

– збереження логів та сповіщення користувача про результати;

– можливість відновлення файлів із резервних копій через інтерфейс програми;

– реєстрація аккаунту Google та отримання авторизації (OAuth 2.0) для доступу до Google Drive.

Нижче наведені нефункціональні вимоги до програми:

– зрозумілий інтерфейс із мінімумом налаштувань. Імовірно, головне вікно з переліком завдань резервного копіювання та кнопками «Створити/Запустити/Відновити»;

– робота через планувальник Windows, що виконує заплановані завдання навіть за відсутності користувача. Захищений механізм відновлення з верифікацією цілісності файлів;

– можливість масштабування на інші платформи (.NET MAUI для мобільної версії, або Blazor WebAssembly для веб-інтерфейсу) у майбутньому;

– надійне зберігання облікових даних (OAuth-токен Google захищено). Шифрування локальних архівів і передача даних за зашифрованим каналом HTTPS через API Google.

Нижче наведені сценарії використання для програми:

– налаштування завдання де користувач запускає програму, вказує обліковий запис Google Drive, користувач вибирає які файли зберігати, куди їх зберігати та задає розклад для планувальника;

– автоматичний запуск, у встановлений час програма виконує резервне копіювання у фоновому режимі та зберігає архів локально або відсилає на Google Drive. Звіти записуються в лог;

– ручний бекап де користувач вибирає файли, куди їх завантажити, включає архівацію якщо треба, також вибирає чи треба завантажити на Google Drive та натискає «Зробити бекап» для негайного створення копії;

– розшифрування файлів де через інтерфейс програми користувач обирає зашифровані файли для відновлення, вписує ключ шифрування, користувач натискає «Зробити бекап» після чого програмне забезпечення розшифровує і копіює файли на початкове місце;

– управління обліковим записом, для доступу до Google Drive користувач входить через OAuth, програма отримує маркер доступу та періодично оновлює його, можна відвязати аккаунт в налаштуваннях програми натиснувши «Очистити токен Google Drive».

Безпека даних опціонально резервні копії, які зберігаються, будуть зашифровані за обраним паролем. Це необхідно, адже незашифровані бекапи вразливі до несанкціонованого доступу. При передачі на Google Drive використовуватиметься HTTPS, а сам доступ здійснюється через офіційний Google Drive API із захищеним OAuth 2.0 – тобто сторонні не можуть підробити авторизацію.

Досвід «AOMEI» та «EaseUS» показує, що зрозуміла головна панель і покрокові майстри підвищують зручність роботи. Початково додаток призначений для ОС Windows (10/11), проте архітектура має дозволяти в майбутньому

адаптувати код для Android (через .NET MAUI [6] або Xamarin [7]) або вебінтерфейсу (наприклад, Blazor [8]).

Розглянемо архітектуру інтерфейсу у проекті. На рисунку 2.1 зображено порядок переходу та їх наповнення – об’єкти та елементи інтерфейсу.

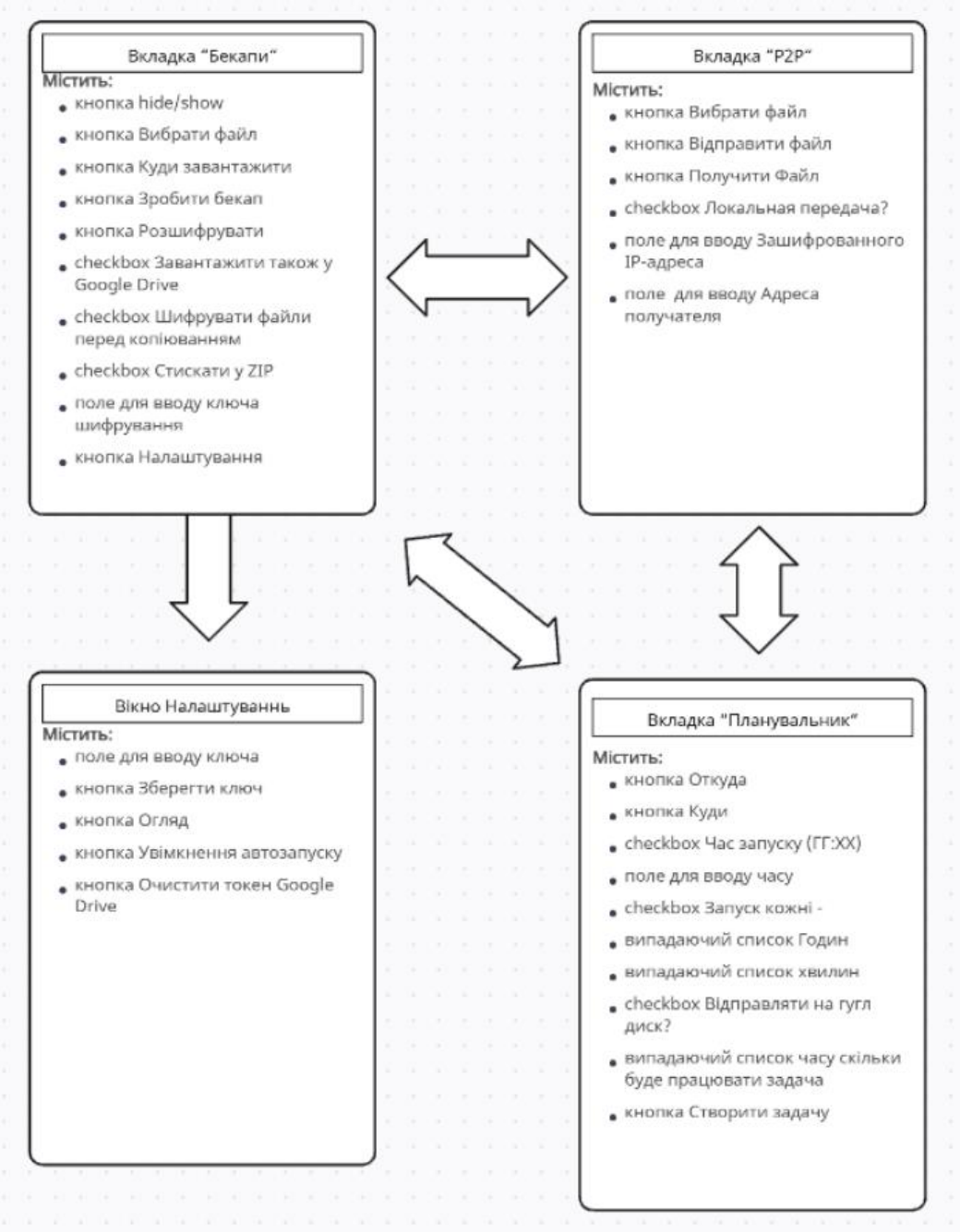

Рисунок 2.1 – Схема інтерфейсу в вкладці «Бекапи»

Структура інтерфейсу досить оптимізована і зі сторони вона виглядає достатньо просто та схематично для будь-якого користувача, що буде тестувати проект у подальшому, або ж пізнавати його вперше (рис. 2.2).

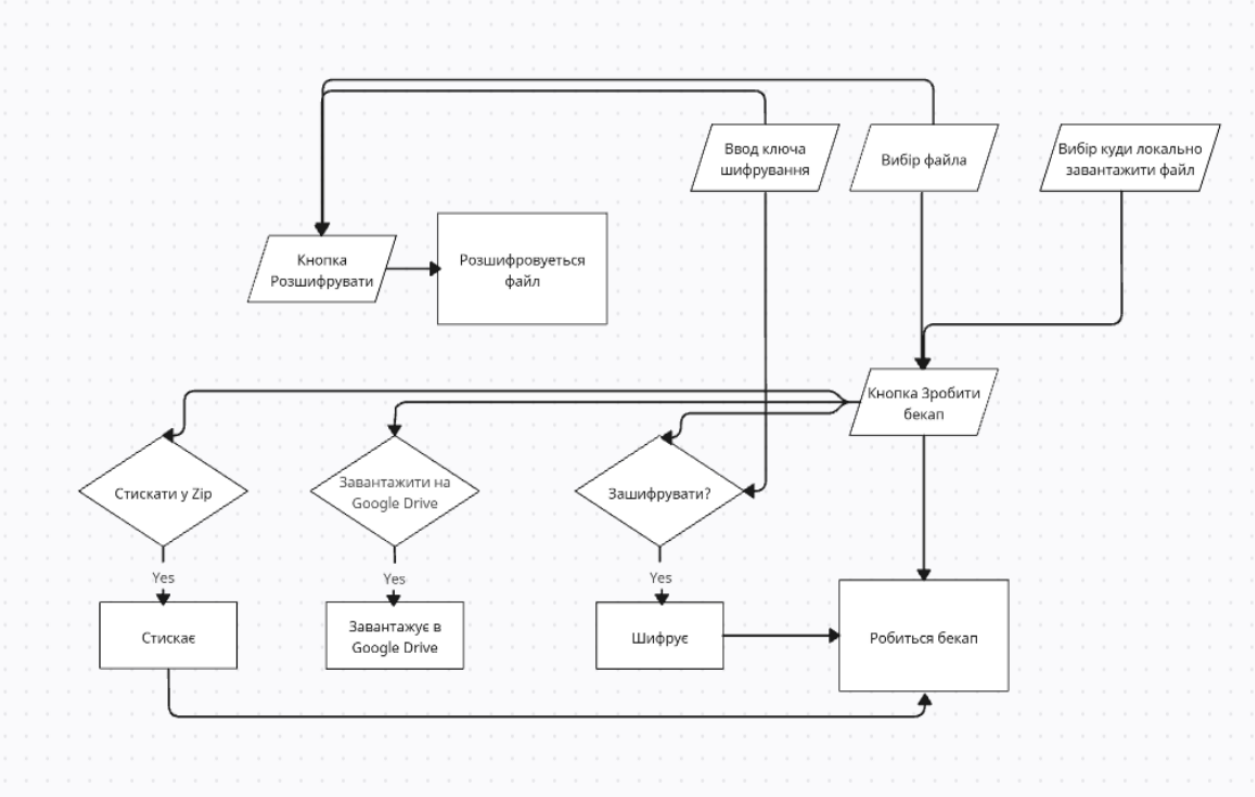

Рисунок 2.2 – Схема інтерфейсу в вкладці «Бекапи»

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Для реалізації програми резервного копіювання було обрано мову програмування C# та платформу **.**NET Framework 4.8, як основне середовище для розробки десктоп-додатку під Windows. Це рішення є логічним, оскільки .NET забезпечує:

– глибоку інтеграцію з ОС Windows;

– підтримку роботи з файловою системою, мережею, криптографією «з коробки»;

– стабільну роботу з бібліотеками для планування задач, доступу до хмарних сервісів тощо.

Платформа також гарантує продуктивне використання ресурсів, управління пам’яттю (garbage collection) і надає інструменти для зручної побудови багатопотокових операцій, що критично для резервного копіювання великих файлів або роботи з мережею (Google Drive, P2P).

Для створення графічного інтерфейсу була обрана WPF. Ця технологія дозволяє реалізувати сучасний UI з анімаціями, прив’язкою даних та гнучким макетуванням через XAML. WPF добре інтегрується з MVVM-патерном, що покращує підтримуваність і тестованість коду. Альтернативами могли б бути Windows Forms, але вони є застарілими і менш придатними для створення адаптивного інтерфейсу.

Для забезпечення взаємодії з хмарним сховищем було обрано Google Drive API [9], який реалізовано через офіційну бібліотеку Google.Apis.Drive.v3 для .NET. Це дозволяє надійно і безпечно авторизуватись за допомогою OAuth 2.0.

Шифрування реалізовано через вбудовану підтримку AES у .NET. Використання AesManaged забезпечує простоту інтеграції і одночасно гарантує високий рівень безпеки. На додачу, алгоритм AES-256 визнано міжнародним стандартом для захисту даних, що робить його логічним вибором для додатку.

В таблиці 2.1 наведені альтернативи які розглядувались під час розробки додатку.

Таблиця 2.1 – Порівняння переваг та недоліків обраних технологій з альтернативой

Задача	Обрана технологія	Альтернатива	Переваги вибраного варіанту	Недоліки альтернатив
Мова програмування	C# (.NET Framework)	Python + PyQt [10] / Java + JavaFX [11]	Висока швидкодія, глибока інтеграція з Windows, зручність роботи з API	Нижча продуктивність, складність розгортання на Windows
Інтерфейс користувача	WPF	WinForms / Electron [12]	Підтримка XAML, стилізація, прив’язка даних, MVVM-патерн	WinForms застарілий, Electron важкий

Продовження таблиці 2.1

Задача	Обрана технологія	Альтернатива	Переваги вибраного варіанту	Недоліки альтернатив
Сховище в хмарі	Google DriveAPI (.NET SDK)	Dropbox API, OneDrive API	Документована підтримка, популярність, гнучкий OAuth	Менша популярність або складніша інтеграція
Шифрування	AES-256(AesManaged)	DES / TripleDES / RSA	Надійність, швидкість, простота реалізації	DES застарілий, RSA непридатний для великих файлів
Архівація	System.IO.Compression (ZIP)	7zip (через процес)	Нативна підтримка у .NET, простота у використанні	7zip потребує зовнішнього інструменту
Автоматизація	Task Scheduler Wrapper [13]	Cron (у Linux), власна реалізація таймерів	Гарантована стабільність через системний планувальник Windows	Потрібні права адміністратора або ручна реалізація логіки
P2P передача	TCP/IP (локально), WebRTC	FTP / Socket.IO	Не потребує встановлення сторонніх серверів, підтримка NAT traversal через STUN	FTP застарілий, Socket.IO складніший для десктопу

Обраний стек технологій (C#, WPF, .NET, Google Drive API, AES-256, Task Scheduler) є оптимальним для реалізації програми резервного копіювання у форматі десктопного застосунку для ОС Windows. Він забезпечує баланс між швидкістю розробки, безпекою, функціональністю та простотою для кінцевого користувача. Обрана архітектура дозволяє масштабувати рішення, додавати нові сервіси (OneDrive, Dropbox), та при потребі портовано перенести логіку на мобільні платформи за допомогою .NET MAUI або Blazor Hybrid.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМИ ДЛЯ АВТОМАТИЧНОГО СТВОРЕННЯ РЕЗЕРВНИХ КОПІЙ ФАЙЛІВ

3.1 Практична реалізація об’єкта проєктування

Розроблений програмний продукт реалізований у вигляді десктопного застосунку на платформі .NET Framework 4.8 з використанням мови C# та WPF (Windows Presentation Foundation) для створення графічного інтерфейсу.

Основу архітектури програми становить модель MVVM (Model-View-ViewModel), що забезпечує розділення логіки інтерфейсу та обробки даних. Такий підхід дозволив розробити зручний, адаптивний та гнучкий інтерфейс користувача з повною підтримкою прив’язки даних і команд.

Під час запуску застосунку виконується ініціалізація основних компонентів: створення робочих директорій, зчитування збережених налаштувань, ініціалізація доступу до Google Drive через OAuth 2.0 та побудова інтерфейсу.

У застосунку реалізовано такі основні модулі:

– модуль вибору файлів для копіювання (OpenFileDialog);

– модуль вибору директорії призначення (CommonOpenFileDialog);

– модуль шифрування даних (на основі AES);

– модуль архівації файлів (на основі System.IO.Compression);

– модуль взаємодії з Google Drive API;

– модуль логування результатів дій у текстовий файл backup_log.txt;

– модуль планувальника задач, який інтегрується з Windows Task Scheduler;

– модуль P2P-передачі файлів, що використовує TCP-з’єднання у локальній мережі;

– модуль керування параметрами шифрування та архівації.

Застосунок має декілька вкладок (Backup, Decrypt, P2P, Scheduler), які логічно поділяють функціональність і дозволяють користувачеві швидко знаходити потрібні інструменти.

Ключовою особливістю реалізації є гнучкість, адже користувач сам обирає, чи буде виконуватися архівація, шифрування, відправлення у хмару чи лише локальне копіювання. Додатково реалізовано автоматичну перевірку введених даних та повідомлення про помилки через MessageBox.

3.1.1 Реалізація інтерфейсу користувача на WPF

Графічне відображення поняття резервного копіювання даних. Інтерфейс клієнта реалізовано з використанням технології WPF (.NET). Графічні елементи (поля введення, кнопки тощо) описано мовою XAML , що забезпечує чітку декларативну структуру представлення. Логіка роботи інтерфейсу і зв’язок з даними винесені у ViewModel відповідно до патерну MVVM, що забезпечує розділення представлення, поведінки та даних.

Головне вікно програми містить елементи управління для вибору каталогу, введення пароля шифрування та запуску процесу резервного копіювання. Зокрема передбачено діалог для вибору директорії, поле для вводу ключа шифрування, кнопку «Зробити бекап» (додаток А), кнопку «Розшифрувати» (додаток Б), кнопку «Налаштування», а також три checkbox для вибору що саме хочете зробити (рис. 3.1). Кожен елемент управління взаємодіє з відповідною логікою у ViewModel, а стан операцій виводиться через текстові повідомлення та записується в лог файл.

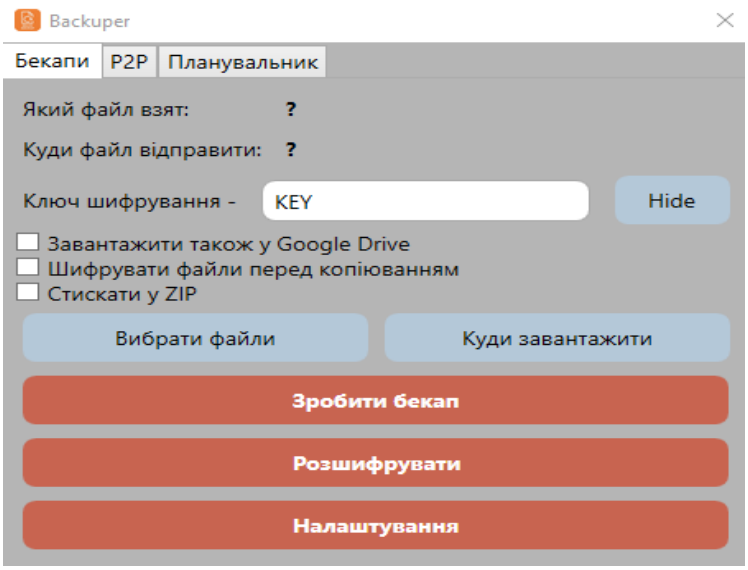

Рисунок 3.1 – Інтерфейс додатка

Для прив’язки подій та команд (натискання кнопок, зміни полів) використовуються механізми команд і зв’язування даних WPF. Такий підхід дозволяє зробити UI гнучким і легко тестованим.

Інтерфейс відображає стан додатку та надає користувачу зручні засоби налаштування. Наприклад, у кнопки вибору папки (використовується стандартний елемент OpenFileDialog) можна вказати файли, які потрібно резервувати (ліст. 3.1).

Лістинг 3.1 – Вибір файлів для резервування

```
    private void FromWhereBackup_Click(object sender, RoutedEventArgs e)
       {
           var dialog = new OpenFileDialog
           {
               Multiselect = true,
               Title = "Оберіть файл(и) для резервного копіювання"
           };
           if (dialog.ShowDialog() == true)
           {
               selectedFiles = dialog.FileNames;
               PathFromWhereToGrab.Content = dialog.SafeFileName;
     }
```

кінець лістингу 3.1

Після введення всіх параметрів та натискання відповідної кнопки запускається процес створення резервної копії та опціональной відправкою на Google Drive (ліст. 3.2).

Лістинг 3.2 – Створення копій для Google Drive

```
   private void UploadToGoogleDrive(string filePath)
{
    var fileMetadata = new Google.Apis.Drive.v3.Data.File()
    {
        Name = Path.GetFileName(filePath)
    };

    using (var stream = new FileStream(filePath, FileMode.Open))
      {
        var request = driveService.Files.Create(fileMetadata, stream,
```

```
"application/octet-stream");
         request.Fields = "id";
         request.Upload();
       }
 }
```

кінець лістингу 3.2

У сукупності реалізований UI забезпечує зручну взаємодію користувача з усіма функціями програми.

3.1.2 Інтеграція з Google Drive API

Для збереження резервних копій використовується Google Drive як хмарне сховище. Взаємодія з ним реалізована за допомогою офіційної бібліотеки Google API для .NET (Google.Apis.Drive.v3). При першому запуску програми користувач повинен авторизувати доступ до свого облікового запису Google (рис. 3.2).

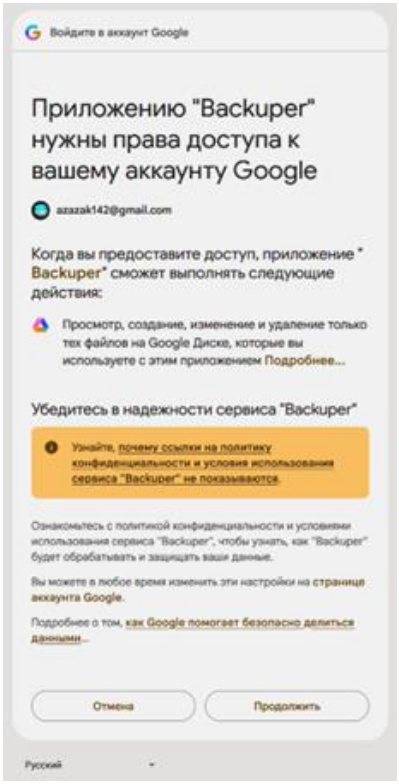

Рисунок 3.2 – Авторизація в Google Drive

Це здійснюється через OAuth 2.0: відкривається браузер з формою входу і запитом доступу. Задається мінімальний набір області дозволів (scopes),

необхідних програмі – наприклад, https://www.googleapis.com/auth/drive.file (дозволяє працювати з файлами, створеними додатком) та базова інформація профілю користувача. Після успішної авторизації Google повертає програмі код доступу, який обмінюється на токени (access token та refresh token) для роботи з API. Отримані токени зберігаються локально для повторного використання, щоб не запитувати повторну авторизацію при кожному запуску. Якщо токен прострочений чи відсутній, відбувається повторна авторизація (ліст. 3.3).

Лістинг 3.3 – Ініціалізація аккаунта Google Drive

```
    private void InitializeGoogleDrive()
{
    var credential = GoogleWebAuthorizationBroker.AuthorizeAsync(
        new ClientSecrets
        {
            ClientId = clientid,
            ClientSecret = clientsecret
        },
        new[] { DriveService.Scope.DriveFile },
        "user",
        CancellationToken.None,
        new FileDataStore(MainTokenPath, true)).Result;

    driveService = new DriveService(new BaseClientService.Initializer()
    {
          HttpClientInitializer = credential,
        ApplicationName = "Backuper"
    });

}
```

кінець лістингу 3.3

Після проходження процесу авторизації та отримання діючого токена доступу, додаток отримує повноцінний доступ до файлової системи користувача на Google Drive у межах наданих дозволів. Це дозволяє виконувати основні операції створення, оновлення та передавання файлів. Реалізація завантаження файлів виконується через виклик методу driveService.Files.Create бібліотеки Drive API. Перед відправкою файли резервної копії (опціонально ZIP-архіви)

передаються у Google Drive: можна вказати чи шифрувати файл та (опційно) архівацію в ZIP. Зазвичай створюється окрема папка (наприклад, «Backuper») у корені Google Drive, куди зберігатимуться всі резервні файли. Після створення резервної копії програма отримує підтвердження успіху або повідомлення про помилки й відображає їх у своєму інтерфейсі. Оскільки в проекті не використовується база даних, інформація про здійснені резервні копії не зберігається на стороні програми, окрім як через файли на Google Drive та локальні налаштування. Для користувача передбачається простий інтерфейс перевірки результату – наприклад, через повідомлення чи backup_log який записує, що саме копіювалось та коли (рис. 3.3).

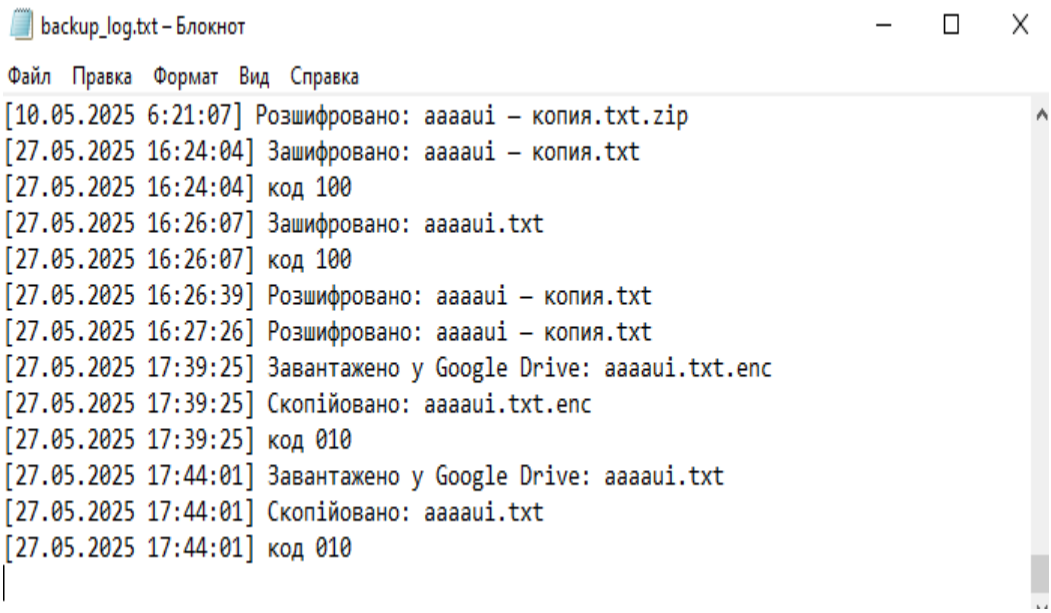

Рисунок 3.3 – backup_log

3.1.3 Відправка файлів по P2P

Однією з додаткових можливостей реалізованого застосунку є функціонал прямої передачі файлів між двома пристроями за допомогою одноранговогo з’єднання (P2P) у локальній мережі. Такий підхід дозволяє користувачам обмінюватися файлами без необхідності використання зовнішніх серверів або хмарних сервісів, що є зручним та безпечним варіантом в обмеженому середовищі, наприклад, в офісі або домашній мережі.

Передача реалізована на основі протоколу TCP (додаток В). Один з комп’ютерів виступає в ролі відправника, інший – у ролі отримувача. Взаємодія між ними забезпечується за допомогою ручного введення зашифрованої IP-адреси отримувача (рис. 3.4), яка передається відправнику для встановлення з’єднання.

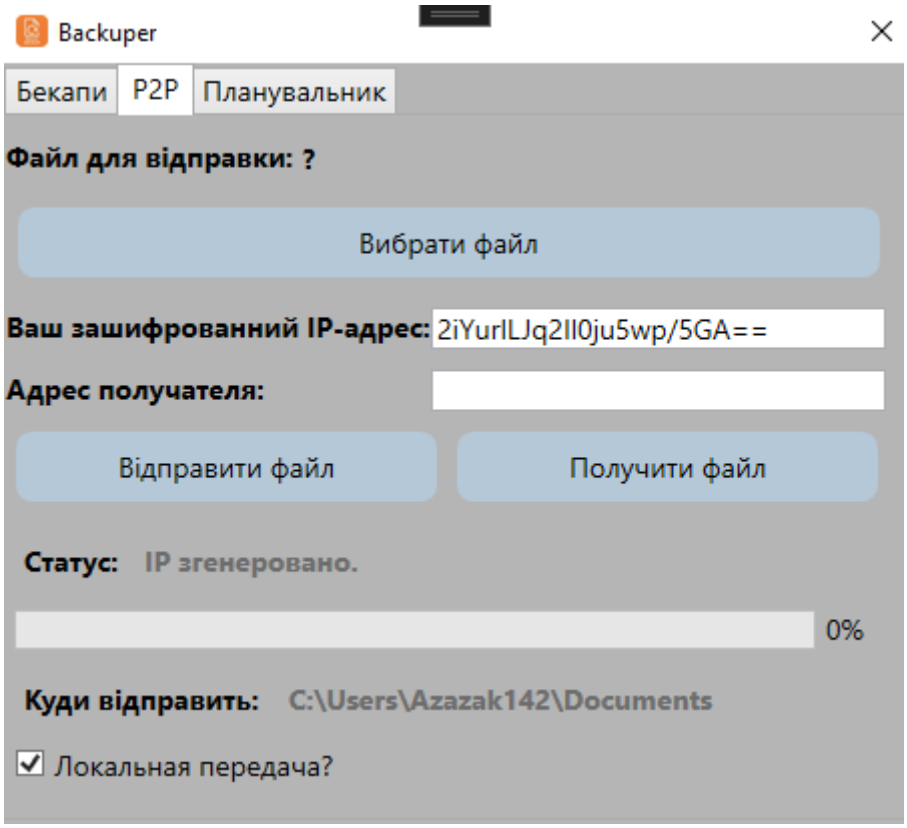

Рисунок 3.4 – Інтерфейс вкладки P2P

Шифрування IP здійснюється за допомогою алгоритму AES, а саме шифрування та дешифрування реалізовані у методах EncryptString та DecryptString (ліст. 3.4).

Лістинг 3.4 – Відправка файлу через TCP

```
private string EncryptString(string plainText)
{
    using (Aes aes = Aes.Create())
    {
        aes.Key = Encoding.UTF8.GetBytes(encryptionKey);
        aes.IV = new byte[16];

        using (MemoryStream ms = new MemoryStream())
```

```
        {
            using (CryptoStream cs = new CryptoStream(ms,
aes.CreateEncryptor(), CryptoStreamMode.Write))
            using (StreamWriter sw = new StreamWriter(cs))
            {
                sw.Write(plainText);
            }
            return Convert.ToBase64String(ms.ToArray());
        }
    }
}

private string DecryptString(string cipherText)
{
    using (Aes aes = Aes.Create())
    {
        aes.Key = Encoding.UTF8.GetBytes(encryptionKey);
        aes.IV = new byte[16];

        using (MemoryStream ms = new
MemoryStream(Convert.FromBase64String(cipherText)))
        using (CryptoStream cs = new CryptoStream(ms, aes.CreateDecryptor(),
CryptoStreamMode.Read))
        using (StreamReader sr = new StreamReader(cs))
        {
            return sr.ReadToEnd();
        }
    }
}
```

кінець лістингу 3.4

Для встановлення з’єднання користувач, який надсилає файл, спочатку обирає файл за допомогою графічного інтерфейсу. Після цього в полі введення «Адрес отримувача» вставляється зашифрована IP-адреса, надана іншим користувачем. Після натискання кнопки «Відправити файл» (ліст. 3.5), застосунок ініціює з’єднання з IP-адресою отримувача, використовуючи порт 9000, та поетапно виконує передачу:

– передається довжина імені файлу та саме ім’я;

– передається розмір файлу у байтах;

– поступово передається весь вміст файлу з відображенням прогресу в інтерфейсі.

Лістинг 3.5 – Відправка файлу через TCP

```
private async void SendFileP2P_Click(object sender, RoutedEventArgs e)
{
    string ip = DecryptString(SomeonesCipheredIP.Text);
    using (TcpClient client = new TcpClient())
    {
        await client.ConnectAsync(IPAddress.Parse(ip), 9000);
        using (NetworkStream stream = client.GetStream())
        using (FileStream fileStream = new FileStream(selectedFilePath,
FileMode.Open))
        {
            string fileName = Path.GetFileName(selectedFilePath);
            byte[] fileNameBytes = Encoding.UTF8.GetBytes(fileName);
            byte[] fileNameLength =
BitConverter.GetBytes(fileNameBytes.Length);

            await stream.WriteAsync(fileNameLength, 0,
fileNameLength.Length);
            await stream.WriteAsync(fileNameBytes, 0,
fileNameBytes.Length);

            long totalBytes = fileStream.Length;
            byte[] lengthBytes = BitConverter.GetBytes(totalBytes);
            await stream.WriteAsync(lengthBytes, 0, lengthBytes.Length);
            byte[] buffer = new byte[81920];
            int bytesRead;
            while ((bytesRead = await fileStream.ReadAsync(buffer, 0,
buffer.Length)) > 0)
            {
                await stream.WriteAsync(buffer, 0, bytesRead);
            }
        }
    }
}
```

кінець лістингу 3.5

На стороні отримувача відкривається TCP-слухач на порту 9000, який очікує вхідне з’єднання. Після його прийняття застосунок приймає метадані файлу, розмір та сам файл, який зберігається у папці, шлях до якої зберігається у локальному файлі p2p_path.txt (ліст. 3.6). Якщо користувач не змінив цей шлях, файли зберігаються за замовчуванням у папці Downloads.

Лістинг 3.6 – Зчитування шляху для збереження отриманих файлів

```
private void LoadSavedPathForP2P()
{
    if (System.IO.File.Exists(p2pPathFile))
    {
        savedReceivePath = System.IO.File.ReadAllText(p2pPathFile);
        P2PPathToFileDesc.Content = savedReceivePath;
    }
}
```

кінець лістингу 3.6

Реалізований механізм P2P-передачі забезпечує ефективний спосіб обміну файлами в локальному середовищі, що може бути корисним для організацій або користувачів, які не мають доступу до інтернету або бажають уникати використання сторонніх хмарних сервісів.

3.1.4 Шифрування та архівація даних

Перед відправкою в хмару дані, що резервуються, спочатку опціонально упаковуються у ZIP-архів. Для цього використовується стандартний клас ZipFile із простору імен System.IO.Compression [14] (ліст. 3.7).

Лістинг 3.7 – Архівація файлів

```
   private static string CreateZipFromFile(string filePath)
{
    if (!System.IO.File.Exists(filePath))
        throw new FileNotFoundException("Файл не найден для архівації",
filePath);

    string zipPath = filePath + ".zip";
    if (System.IO.File.Exists(zipPath))
        System.IO.File.Delete(zipPath);

    using (FileStream zipToOpen = new FileStream(zipPath, FileMode.Create))
    using (ZipArchive archive = new ZipArchive(zipToOpen,
ZipArchiveMode.Create))
    {
        ZipArchiveEntry entry =
archive.CreateEntry(System.IO.Path.GetFileName(filePath));

        using (Stream entryStream = entry.Open())
        using (FileStream originalFile = new FileStream(filePath,
```

```
FileMode.Open, FileAccess.Read, FileShare.Read))
        {
            originalFile.CopyTo(entryStream);
        }
    }
    return zipPath;
}
```

кінець лістингу 3.7

Далі цей архів може бути зашифрований для безпеки. У реалізації використовується симетричне шифрування за алгоритмом AES (Advanced Encryption Standard) [15]. У .NET забезпечено клас Aes (абстракція симетричного алгоритму), який генерує ключ та ініціалізаційний вектор (IV) для шифрування/дешифрування. Зазвичай користувач вводить пароль, який згодом перетворюється на ключ AES. Процес шифрування можна уявити так: створюється об’єкт Aes, генеруються Key та IV, після чого викликається метод CreateEncryptor(Key, IV) та CryptoStream для проходу даних архіву (ліст. 3.8).

Лістинг 3.8 – Шифрування файлів

```
  private void EncryptFile(string inputPath, string outputPath, string key)
{
    using (FileStream input = new FileStream(inputPath, FileMode.Open,
FileAccess.Read, FileShare.Read))
    using (FileStream output = new FileStream(outputPath, FileMode.Create,
FileAccess.Write, FileShare.None))
    using (Aes aes = Aes.Create())
    {
         aes.Key = Encoding.UTF8.GetBytes(key);
         aes.IV = new byte[16];

        using (CryptoStream cryptoStream = new CryptoStream(output,
aes.CreateEncryptor(), CryptoStreamMode.Write))
        {
            input.CopyTo(cryptoStream);
        }
    }
}
```

кінець лістингу 3.8

Як результат роботи алгоритму шифрування, отримується масив байтів, який містить зашифровану версію оригінального файлу Цей масив записується у новий файл з розширенням .enc, що вказує на його зашифрований статус. Наприклад, якщо оригінальний файл мав назву backup.zip, після шифрування він перетворюється на backup.zip.enc. Для розшифрування (при відновленні) виконується зворотня процедура з використанням того самого ключа і IV (ліст. 3.9).

Лістинг 3.9 – Розшифрування файлів

```
 private void DecryptFile(string inputPath, string outputPath,string key)
 {
     using (FileStream input = new FileStream(inputPath, FileMode.Open))
     using (FileStream output = new FileStream(outputPath,
FileMode.Create))

     using (Aes aes = Aes.Create())
     {
         aes.Key = Encoding.UTF8.GetBytes(key);
         aes.IV = new byte[16];

         using (CryptoStream cryptoStream = new CryptoStream(input,
aes.CreateDecryptor(), CryptoStreamMode.Read))
         {
             cryptoStream.CopyTo(output);
         }

     }

 }
```

кінець лістингу 3.9

Важливо, що пароль шифрування не зберігається у відкритому вигляді – користувач повинен ввести його під час кожного створення резервної копії, щоб виключити несанкціонований доступ до даних але якщо треба то можна зайти в налаштування та зберегти ключ (рис. 3.5).

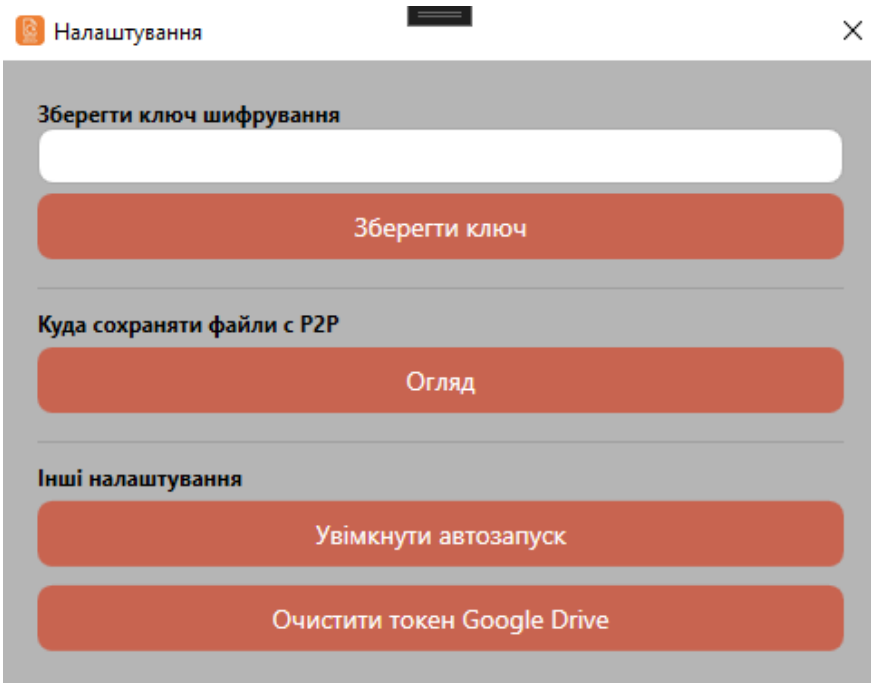

Рисунок 3.5 – Налаштування додатка

3.1.5 Планувальник для збереження файлів

Однією з важливих функціональних можливостей програмного забезпечення є автоматизація процесу створення резервних копій за заданим графіком. Для реалізації цієї функціональності була створена підсистема планування, яка взаємодіє з вбудованим планувальником завдань операційної системи Windows через бібліотеку TaskScheduler [13]. Ця бібліотека є обгорткою над API Windows Task Scheduler і дозволяє створювати, редагувати, видаляти та запускати заплановані задачі безпосередньо з коду C#. Інтерфейс користувача (рис. 3.6), був розроблений із дотриманням принципів простоти та доступності. У ньому передбачено елементи керування, за допомогою яких користувач може:

– обрати звідки та куди копіювати у діалогових вікнах;

– вказати параметри запуску задачі згідно з потребами;

– щоденне копіювання у вказаний час;

– повторюване копіювання через вказаний інтервал (наприклад, кожні 30 хвилин).

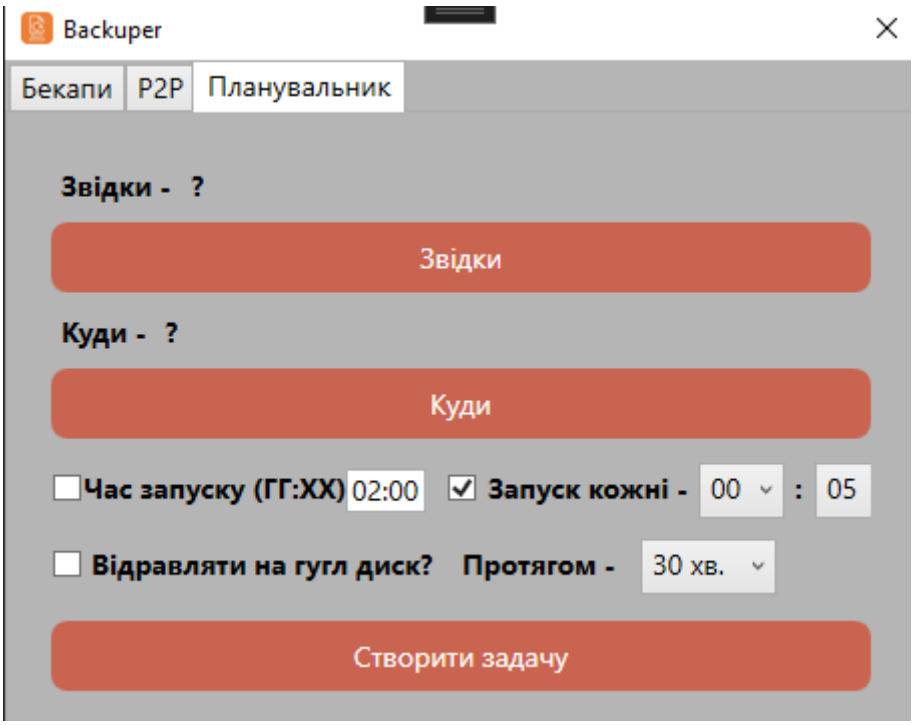

Рисунок 3.6 – Інтерфейс вкладки планувальника

Додатково користувач може вказати, чи потрібно завантажувати копії у Google Drive. Це реалізується шляхом передачі аргументів (ліст. 3.10) до окремого консольного модуля BackuperWorker.exe.

Лістинг 3.10 – Приймання та обробка аргументів у BackuperWorker.exe

```
try
            {
                string filesArg = GetArgValue(args, "--files");
                string toArg = GetArgValue(args, "--to");
                string googleArg = GetArgValue(args, "--google");
                string tokenPath = GetArgValue(args, "--token"); if
(string.IsNullOrWhiteSpace(filesArg) || string.IsNullOrWhiteSpace(toArg))
                {
                    Log("Аргументи не вказано. Завершення.");
                    return;
                }
                string[] files = filesArg.Split(new[] { ';' },
StringSplitOptions.RemoveEmptyEntries);
                string destinationFolder = toArg;

                foreach (string filePath in files)
                {
                    if (File.Exists(filePath))
                    {
```

```
                        string fileName = Path.GetFileName(filePath);
                        string destPath = Path.Combine(destinationFolder,
fileName);
                        File.Copy(filePath, destPath, true);
                        Log($"Скопійовано: {fileName}");
                    }
                    else
                    {
                        Log($"Не знайдено файл: {filePath}");
                    }
                    if (googleArg == "true")
                    {
                        Log("Відправлення в Google Диск...");
                        UploadToGoogleDrive(filePath, tokenPath);
                    }
                }
            }
            catch (Exception ex)
            {
                Log("Помилка: " + ex.ToString());
        }
```

кінець лістингу 3.10

Після натискання кнопки «Створити задачу» (додаток Г) відбувається перевірка заповнення полів: список файлів selectedFiles та шлях призначення selectedFolder. Якщо дані валідні, створюється об’єкт задачі за допомогою TaskDefinition.

Якщо обрано щоденний режим, то до задачі додається тригер DailyTrigger, який активується щодня у вказаний час. Якщо обрано повторюваний режим, то створюється TimeTrigger із заданим інтервалом повторення та можливістю обмеження тривалості (наприклад, 1 година, 1 день тощо). Тривалість обирається з випадаючого списку, а обробляється методом GetDurationFromComboBox() (ліст. 3.11).

Лістинг 3.11 – Отримання тривалості задачі з випадаючого списку

```
private TimeSpan GetDurationFromComboBox()
{
    var selectedItem = ForHowLongChoice.SelectedItem as ComboBoxItem;
    if (selectedItem != null)
    {
```

```
        string value = selectedItem.Content.ToString();
        switch (value)
        {
            case "15 хв.": return TimeSpan.FromMinutes(15);
            case "30 хв.": return TimeSpan.FromMinutes(30);
            case "1 год.": return TimeSpan.FromHours(1);
            case "12 год.": return TimeSpan.FromHours(12);
            case "1 день": return TimeSpan.FromDays(1);
            case "Infinite": return TimeSpan.MaxValue;
        }
    }
    return TimeSpan.MaxValue;
}
```

кінець лістингу 3.11

Після формування тригерів до задачі додається дія запуску окремого виконавчого файлу BackuperWorker.exe з передачею аргументів:

– --files шляхи до файлів через « ; »;

– --to папка призначення;

– --google ознака чи завантажити у хмару;

– --token шлях до токена Google Drive для авторизації.

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

Для забезпечення можливості зберігання резервних копій у хмарі у програмі реалізовано інтеграцію з Google Drive API. Ця інтеграція надає змогу користувачам зберігати резервні копії у своєму обліковому записі Google безпосередньо з інтерфейсу застосунку.

Аутентифікація користувача виконується за допомогою OAuth 2.0, що є стандартом для безпечного доступу до сервісів Google. При першому запуску відкривається браузер, де користувач надає програмі дозвіл на доступ до файлів, після чого токен зберігається локально в папці google-token, що дозволяє уникати повторного входу.

Метод виконування ініціалізації доступу здійснюється через метод InitializeGoogleDrive() в якому є поля аунтефікації (ліст. 3.12)

Лістинг 3.12 – Ініціалізація доступу до Google

```
var credential = GoogleWebAuthorizationBroker.AuthorizeAsync( new
ClientSecrets {
ClientId = "...",
 ClientSecret = "..." },
new[] { DriveService.Scope.DriveFile },
"user",
CancellationToken.None,
new FileDataStore(tokenPath, true)).Result;
```

кінець лістингу 3.12

Цей токен використовується для створення об’єкта DriveService, який виконує всі подальші дії: створення файлу, завантаження файлу, оновлення тощо.

Файли перед відправкою можуть бути:

– заархівовані у формат .zip;

– зашифровані за алгоритмом AES (.zip.enc);

– скопійовані локально або збережені тільки в хмарі.

Результат завантаження записується у лог, а користувачу відображається відповідне повідомлення.

Додатково передбачена передача шляху до токена як аргумент у фоновий консольний модуль BackuperWorker, що дозволяє здійснювати хмарне резервне копіювання навіть без активного GUI (використовується в задачах Windows Task Scheduler).

Реалізована інтеграція повністю відповідає вимогам безпеки Google, забезпечує безперебійну роботу з Google Drive без зайвих дій користувача та дозволяє автоматизувати збереження даних у хмарі.

3.3 Захист інформаційно-комп’ютерної системи

У сучасних інформаційно-комп’ютерних системах захист даних є критично важливою складовою, особливо коли мова йде про обробку резервних копій конфіденційних файлів користувача. У межах даної кваліфікаційної роботи

реалізовано низку механізмів, що забезпечують автентифікацію, шифрування даних, захист від несанкціонованого доступу та запобігання аналізу коду сторонніми особами.

Доступ до хмарного сховища Google Drive реалізовано через офіційний API за допомогою механізму авторизації OAuth 2.0. При першому запуску користувач проходить процедуру входу через захищений веб-інтерфейс Google та підтверджує запит доступу до певних ресурсів (у нашому випадку лише до файлів, створених додатком). У результаті цього обміну додаток отримує токени доступу (access token та refresh token), які зберігаються у локальному зашифрованому сховищі та використовуються при наступних сесіях без повторного запиту.

Таким чином, система гарантує, що лише власник акаунта має змогу передавати файли у свій Google Drive, що повністю відповідає сучасним вимогам до безпеки аутентифікації користувача.

Для захисту змісту резервних копій передбачено модуль шифрування, який використовує стандарт AES (Advanced Encryption Standard) з довжиною ключа 256 біт. Це симетричний блочний шифр, який працює з блоками даних по 128 біт, забезпечуючи високий рівень криптографічної стійкості. Процес шифрування виконується перед передачею файлу в хмарне сховище або перед його збереженням локально.

Алгоритм шифрування реалізовано на основі стандартного класу Aes з простору імен System.Security.Cryptography. Пароль користувача конвертується в криптографічний ключ та використовується разом із вектором ініціалізації для побудови потоку шифрування (CryptoStream). Результатом є новий зашифрований файл із розширенням .enc, що є недоступним для перегляду без знання оригінального ключа.

Для зворотного перетворення (дешифрування) використовується аналогічна процедура з застосуванням методу CreateDecryptor. Пароль не зберігається у відкритому вигляді він або вводиться користувачем вручну, або зберігається у зашифрованому конфігураційному файлі за бажанням користувача.

Вся передача даних у хмару або по P2P-каналах здійснюється вже після шифрування, тому навіть при можливості перехоплення трафіку третім особам дані будуть недоступні без ключа розшифрування. Це дозволяє уникнути витоку інформації внаслідок атак типу «man-in-the-middle».

У випадку передачі файлів через P2P-модуль, реалізовано можливість шифрування IP-адрес користувачів перед їх передачею (AES-шифрування адреси у вигляді строки), що запобігає розкриттю мережевої інфраструктури в незахищеному середовищі.

Оскільки програма реалізована мовою C# на базі .NET Framework, вона компілюється у проміжний байт-код, який теоретично може бути декомпільований. Для запобігання цьому передбачено можливість застосування обфускації (наприклад, із використанням Dotfuscator, ConfuserEx, ILMerge тощо). Обфускація дозволяє заплутати вихідний код, змінивши імена класів, методів, змінних, а також вставити псевдологіку, що ускладнює реверс-інженеринг.

Усі конфігураційні файли, які використовує програма (включно з шляхами до файлів, останнім вибраним каталогом тощо), зберігаються в локальному каталозі додатка та не містять чутливої інформації у відкритому вигляді. У випадку збереження ключа шифрування, цей ключ може бути зашифрований аналогічним методом AES перед записом у файл.

Хоча програма не має серверної частини та не приймає вхідних запитів із зовнішньої мережі (окрім ініціативи користувача в P2P-модулі), потенційні DDoS-атаки виключаються через відсутність постійно відкритого порту або серверу. Всі операції ініціюються локально, тому додаток не є вразливим до типових атак через публічний API.

3.4 SEO інформаційно-комп’ютерної системи

Хоча програмне забезпечення Backuper є настільним застосунком для Windows, його ефективне просування серед цільової аудиторії напряму залежить від впізнаваності в інтернет-просторі. Тому важливою складовою подальшого

розгортання продукту є реалізація стратегії SEO (Search Engine Optimization) та загального інформаційного супроводу в мережі.

У межах цього проєкту SEO-аспекти стосуються створення власної презентаційної сторінки, публікацій у профільних спільнотах і забезпечення наявності програми в результатах пошуку за релевантними запитами.

З метою підвищення видимості програми рекомендовано реалізувати лендінг-сторінку, яка міститиме:

– короткий опис програми та її переваг;

– скріншоти користувацького інтерфейсу;

– демонстраційне відео з прикладом резервного копіювання;

– посилання на інсталяційний файл;

– відповіді на поширені запитання (FAQ);

– контактну форму для зворотного зв’язку.

Розмістити таку сторінку доцільно на безкоштовному хостингу або сервісах типу GitHub Pages, Netlify, Firebase Hosting чи Tilda, що забезпечують базові засоби для SEO та статистику переглядів.

ля того, щоб користувачі змогли знайти програму через пошукові системи (наприклад, Google), необхідно:

– використовувати ключові слова у назві, описі та мета-тегах сторінки;

– написати структурований опис із підзаголовками;

– реалізувати адаптивну верстку сайту для відображення на мобільних пристроях;

– надати текстову документацію українською та англійською мовами.

Також бажано, щоб у коді сайту були присутні структуровані дані (наприклад, за стандартом Schema.org), що допоможе пошуковим системам краще індексувати сторінку.

Оскільки програма є написаною на .NET і має структуру, зручною для відкритої дистрибуції, доцільно розмістити репозиторій на GitHub з відкритим вихідним кодом або принаймні з демонстраційною збіркою та README-файлом, який описує функціонал.

3.5 Інструкція з використання додатком

Нижче наведено послідовність дій для користувача при роботі із програмою:

– відкрити Backuper (двічі клацнути по ярлику або через меню «Пуск»);

– якщо програма запускається вперше або немає токена Google, відкриється браузер де треба ввести дані свого облікового запису Google. Після надання дозволу браузер можна закрити (програма отримає токен доступу);

– натиснути кнопку «Вибрати файли» та вказати файли, які потрібно зберегти;

– натиснути кнопку «Куди завантажити» та вказати директорію, куди потрібно зберегти;

– в полі «Ключ шифрування» ввести ключ, який буде використаний для шифрування файла (якщо шифрування ввімкнена);

– за потреби ввімкнути створення ZIP-архіву;

– натиснути кнопку «Зробити бекап». Програма автоматично створить ZIP-архів (якщо архівація ввімкнена) із вмістом обраної папки, при необхідності зашифрує його заданим паролем (якщо шифрування ввімкнена), а потім передасть результат на Google Drive (якщо передача на Google Drive ввімкнена);

– у процесі виконання відображатимуться повідомлення про стан операцій (успіх/помилки) а також можна переглянути «backup_log» в папці з додатком в якому прописані всі операції по часу. Після завершення операції можна перевірити свій Google Drive з’явиться новий файл із резервною копією в обраної директорії;

– якщо потрібне автоматичне копіювання з часом, можна нажати на вкладку планувальника та створити задачу яка наприклад автоматично резервно копіює вибраний вами файл кожні 30 хвилин куди ви вкажете з опціональною відправкою на Google Drive;

– увімкнення автозапуску (опційно). Якщо потрібен автоматичний запуск, у налаштуваннях програми можна ввімкнути опцію автозапуску. Це призведе до

створення запису в реєстрі (HKCU\…\Run), і Backuper запускатиметься з Windows. Для відключення автозапуску повернути налаштування назад;

– передача файлів по P2P (опційно). Якщо треба комусь відправити файли напряму, можна перейти на вкладку P2P де обираєш файл який хочеш відправити вписуєш адресу отримувача та натискаєш кнопку «Відправити файл» а сам отримувач перед цим має відправити вам свою зашифровану адресу та нажати на кнопку «Получити файл». Якщо передача локальна то обидва користувача мають поставити галочку що це локальна передача.

Після налаштування цих параметрів користувач може в один клік створювати зашифровані резервні копії своїх даних. Програма не потребує додаткового обслуговування, тому для базової експлуатації достатньо вказати папку, пароль і запустити резервне копіювання. При цьому конфіденційність забезпечується використанням AES-шифрування, а збереження копій у хмарі здійснюється через Google Drive API безпосередньо з інтерфейсу додатку.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи та розробки програми резервного копіювання на основі C# і WPF були досягнуті поставлені цілі і виконані всі завдання.

Здійснено аналіз існуючих програм-аналогів, зокрема Macrium Reflect, AOMEI Backupper, EaseUS Todo Backup та інших, що дозволило виявити ключові недоліки та сформулювати унікальні вимоги до майбутнього рішення. Основна увага була зосереджена на поєднанні простоти інтерфейсу, безпеки збереження, підтримки хмарних технологій і автоматизації резервного копіювання.

Визначено функціональні і нефункціональні вимоги, а саме до основних функціональних вимог увійшли: автоматизація резервного копіювання, підтримка архівації, шифрування, інтеграція з Google Drive, логування та можливість відновлення. Серед нефункціональних: зручність інтерфейсу, безпека, стабільність і можливість масштабування на інші платформи.

Обрано мову програмування C# і платформу .Net., бо вони поєднають функціональність, безпеку, зручність використання та легке масштабування на різні девайси. Для розробки використовувалось середовище Visual Studio 2022. В якості інтерфейсної частини застосовано WPF із використанням шаблонів XAML, що забезпечило розділення логіки та представлення (MVVM).

Спроектовано архітектуру, яка розділяє ключові компоненти: модулі копіювання, шифрування, Google Drive, планування, інтерфейс користувача, логування та P2P. Це забезпечує масштабованість і легке супроводження коду. Під час реалізації дотримано принципів модульності та повторного використання.

Розроблено зручний і зрозумілий інтерфейс для взаємодії з користувачем використовуючи XAML в WPF. Серед особливостей інтерфейсу можна виділити інтуїтивну побудову форми, наявність логічно згрупованих панелей, індикатор прогресу, кнопку вибору файлів і директорій, шифрування за ключем, відображення статусу виконання та кнопку активації бекапу в один клік.

Реалізовано основний функціонал, автоматичне резервне копіювання за розкладом, вибір шляху збереження локально та в хмарі, базове шифрування даних, Простий і інтуїтивний інтерфейс та логування.

Здійснено тестування програми, в результаті чого підтверджено її стабільність, надійність і функціональність у типових сценаріях використання. Під час перевірки були враховані граничні випадки, обробка помилок, перевірено збереження даних при збоях, протестовано роботу із Google Drive та планувальником завдань.

Розроблено користувацьку документацію, яка містить інструкції з налаштування та приклади використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Macrium Software. Macrium Reflect Free. URL: https://www.macrium.com/reflectfree (дата звернення: 04.02.2025).

2. EaseUS. EaseUS Todo Backup Free. URL: https://www.easeus.com/backup-software/tb-free.html (дата звернення: 04.02.2025).

3. AOMEI Technology. AOMEI Backupper Standard. URL: https://www.aomeitech.com/ab/standard.html (дата звернення: 04.02.2025).

4. Paragon Software Group. Paragon Backup & Recovery Community Edition. URL: https://www.paragon-software.com/free/br-free/ (дата звернення: 04.02.2025).

5. FreeFileSync Development Team. FreeFileSync. URL: https://freefilesync.org/ (дата звернення: 01.03.2025).

6. .NET MAUI. URL: https://dotnet.microsoft.com/en-us/apps/maui (дата звернення: 15.03.2025).

7. Xamarin. URL: https://dotnet.microsoft.com/en-us/apps/xamarin (дата звернення: 15.03.2025).

8. Blazor. URL: https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor (дата звернення: 15.03.2025).

9. Google LLC. Drive API (v3) – Google Developers. URL: https://developers.google.com/drive/api/v3 (дата звернення: 29.03.2025).

10. PyQt. URL: https://wiki.python.org/moin/PyQt (дата звернення: 29.03.2025).

11. JavaFx. URL: https://openjfx.io/ (дата звернення: 29.03.2025).

12. Electron. URL: https://www.electronjs.org/ (дата звернення: 29.03.2025).

13. TaskScheduler. URL: https://www.nuget.org/packages/TaskScheduler/ (дата звернення: 26.04.2025).

14. Microsoft. System.IO.Compression Namespace. Документація .NET. URL: https://docs.microsoft.com/dotnet/api/system.io.compression (дата звернення: 26.04.2025).

15. Microsoft. Aes Class (System.Security.Cryptography). URL:

https://docs.microsoft.com/dotnet/api/system.security.cryptography.aes (дата звернення: 26.04.2025).

ДОДАТКИ

Додаток А – Кнопка зробити бекап

```
 private async void MakeBackup_Click(object sender, RoutedEventArgs e)
 {
     if (selectedFiles == null || selectedFiles.Length == 0 ||
string.IsNullOrEmpty(selectedFolder))
     {
         MessageBox.Show("Не вибрано файли або папку.");
         return;
     }

     bool uploadToGoogleDrive = GoogleDriveCheckbox.IsChecked == true;
     bool encryptFiles = EncryptCheckbox.IsChecked == true;
     bool zipBeforeUpload = ZipCheckbox.IsChecked == true;

     string currentKey = Dispatcher.Invoke(() => GetCurrentKey());

     foreach (var filePath in selectedFiles)
     {
         string fileName = Path.GetFileName(filePath);
         string destinationPath = Path.Combine(selectedFolder, fileName);
         string encryptedPath = destinationPath + ".enc";
         string uploadPath = destinationPath;
         string tempZip = null;



             string combination = $"{(encryptFiles ? "1" :
"0")}{(uploadToGoogleDrive ? "1" : "0")}{(zipBeforeUpload ? "1" : "0")}";

             switch (combination)
             {
                 case "000":
                 if (!System.IO.File.Exists(destinationPath))
                         System.IO.File.Copy(filePath, destinationPath);
                 uploadPath = destinationPath;
                     Log("[" + DateTime.Now + "] Скопійовано: " + fileName);
                     Log("[" + DateTime.Now + "] код 000");
                     break;
                 case "001":
                 if (!System.IO.File.Exists(destinationPath))
                     System.IO.File.Copy(filePath, destinationPath);
                 uploadPath = destinationPath;

                     tempZip = CreateZipFromFile(uploadPath);
                     await System.Threading.Tasks.Task.Delay(150);
                     Log("[" + DateTime.Now + "] архівовано ZIP: " +
Path.GetFileName(tempZip));
                     Log("[" + DateTime.Now + "] код 001");
                     break;
```

```
                case "010":
                if (!System.IO.File.Exists(destinationPath))
                        System.IO.File.Copy(filePath, destinationPath);
                uploadPath = destinationPath;

                    UploadToGoogleDrive(uploadPath);
                    Log("[" + DateTime.Now + "] Завантажено у Google Drive:
" + Path.GetFileName(uploadPath));
                    Log("[" + DateTime.Now + "] Скопійовано: " + fileName);
                    Log("[" + DateTime.Now + "] код 010");
                    break;
                case "011":
                if (!System.IO.File.Exists(destinationPath))
                    System.IO.File.Copy(filePath, destinationPath);
                uploadPath = destinationPath;

                    tempZip = CreateZipFromFile(uploadPath);
                    await System.Threading.Tasks.Task.Delay(150);

                    UploadToGoogleDrive(tempZip);
                    Log("[" + DateTime.Now + "] Завантажено у Google Drive
ZIP: " + Path.GetFileName(tempZip));
                    Log("[" + DateTime.Now + "] код 011");
                    break;
                case "100":
                    EncryptFile(filePath, encryptedPath, currentKey);
                    uploadPath = encryptedPath;
                    Log("[" + DateTime.Now + "] Зашифровано: " + fileName);



Log("[" + DateTime.Now + "] код 100");
                    break;
                case "101":
                tempZip = CreateZipFromFile(filePath);
                await Task.Delay(150);
                Log("[" + DateTime.Now + "] архівовано ZIP: " +
Path.GetFileName(tempZip));


                encryptedPath = tempZip + ".enc";
                EncryptFile(tempZip, encryptedPath, currentKey);
                Log("[" + DateTime.Now + "] Зашифровано ZIP: " +
Path.GetFileName(encryptedPath));


                if (System.IO.File.Exists(tempZip))
                    System.IO.File.Delete(tempZip);

                string finalName = Path.GetFileName(encryptedPath);   //fix
kostili
```

```
                destinationPath = Path.Combine(selectedFolder, finalName);
                if (!System.IO.File.Exists(destinationPath))
                    System.IO.File.Copy(encryptedPath, destinationPath);
                Log("[" + DateTime.Now + "] Скопійовано: " +
destinationPath);
                Log("[" + DateTime.Now + "] код 101");
                break;
                case "110":
                    EncryptFile(filePath, encryptedPath,
currentKey);
                    uploadPath = encryptedPath;

                    UploadToGoogleDrive(uploadPath);
                    Log("[" + DateTime.Now + "] Завантажено у Google Drive:
" + Path.GetFileName(uploadPath));
                    Log("[" + DateTime.Now + "] Зашифровано: " + fileName);
                    Log("[" + DateTime.Now + "] код 110");
                    break;
                case "111":
                tempZip = CreateZipFromFile(filePath);
                await Task.Delay(150);
                Log("[" + DateTime.Now + "] архівовано ZIP: " +
Path.GetFileName(tempZip));


                encryptedPath = tempZip + ".enc";
                EncryptFile(tempZip, encryptedPath, currentKey);
                Log("[" + DateTime.Now + "] Зашифровано ZIP: " +
Path.GetFileName(encryptedPath));


                if (System.IO.File.Exists(tempZip))
                    System.IO.File.Delete(tempZip);

                finalName = Path.GetFileName(encryptedPath);
                destinationPath = Path.Combine(selectedFolder, finalName);
                if (!System.IO.File.Exists(destinationPath))
```



```
                    System.IO.File.Copy(encryptedPath, destinationPath);
                Log("[" + DateTime.Now + "] Скопійовано: " +
destinationPath);

                UploadToGoogleDrive(encryptedPath);
                Log("[" + DateTime.Now + "] Завантажено у Google Drive: "
+ Path.GetFileName(encryptedPath));
                Log("[" + DateTime.Now + "] код 111");



                break;
```

```
            }

    }

MessageBox.Show("Резервне копіювання завершено.", "Успіх");
}
```

Додаток Б – Кнопка розшифрувати

```
 private async void Decipher_Click(object sender, RoutedEventArgs e)
 {
     string key = Dispatcher.Invoke(() => GetCurrentKey());
     bool uploadToGoogleDrive = GoogleDriveCheckbox.IsChecked == true;

     if (selectedFiles == null || selectedFiles.Length == 0 ||
string.IsNullOrEmpty(selectedFolder))
     {
         MessageBox.Show("Не вибрано файли або папку для розшифрування.");

         return;
     }

     foreach (var filePath in selectedFiles)
     {
         if (!filePath.EndsWith(".enc")) continue;

         string fileName = Path.GetFileNameWithoutExtension(filePath);
         string destination = Path.Combine(selectedFolder, fileName);

         try
         {
             await          System.Threading.Tasks.Task.Run(()          =>
DecryptFile(filePath, destination, key));

             GC.Collect();
             GC.WaitForPendingFinalizers();

             Log("[" + DateTime.Now + "] Розшифровано: " + fileName);

             if (uploadToGoogleDrive)
             {
                 UploadToGoogleDrive(destination);     //     Завантаження
результату розшифрування

                 Log("[" + DateTime.Now + "] Завантажено розшифрований
файл у Google Drive: " + fileName);
             }
         }
         catch (Exception ex)
         {
             MessageBox.Show("Помилка при розшифруванні: " + ex.Message);

             Log("[" + DateTime.Now + "] ПОМИЛКА при розшифруванні " +
fileName + ": " + ex.Message);
         }
     }
     GC.Collect();
```

```
    GC.WaitForPendingFinalizers();
    MessageBox.Show("Розшифрування завершено.", "Успіх");
}
```

Додаток В – Кнопка Відправки по локальному P2P

```
 private async void SendFileP2P_Click(object sender, RoutedEventArgs e)
{
    try
    {
        string ip = DecryptString(SomeonesCipheredIP.Text);

        using (TcpClient client = new TcpClient())
        {
            await client.ConnectAsync(IPAddress.Parse(ip), 9000);
            using (NetworkStream stream = client.GetStream())
            using (FileStream fileStream = new FileStream(selectedFilePath,
FileMode.Open))
            {

                string fileName = Path.GetFileName(selectedFilePath);
                byte[] fileNameBytes = Encoding.UTF8.GetBytes(fileName);
                byte[]                   fileNameLength                   =
BitConverter.GetBytes(fileNameBytes.Length);
                await          stream.WriteAsync(fileNameLength,          0,
fileNameLength.Length);
                await           stream.WriteAsync(fileNameBytes,           0,
fileNameBytes.Length);

                long totalBytes = fileStream.Length;
                byte[] lengthBytes = BitConverter.GetBytes(totalBytes);
                await            stream.WriteAsync(lengthBytes,            0,
lengthBytes.Length);
                long totalRead = 0;
                byte[] buffer = new byte[81920];
                int bytesRead;
                while ((bytesRead = await fileStream.ReadAsync(buffer, 0,
buffer.Length)) > 0)
                {
                    await stream.WriteAsync(buffer, 0, bytesRead);
                    totalRead += bytesRead;

                    double progress = (double)totalRead / totalBytes * 100;
                    Dispatcher.Invoke(() =>
                    {
                    FileTransferProgressBar.Value = progress;
                    ProgressPercentageText.Text = $"{progress:F2}%";
                    });
                }
            }
            client.Close();
            P2PStatusText.Content = "Файл надіслано успішно.";
            ProgressPercentageText.Text = "0%";
            FileTransferProgressBar.Value = 0;
```

```
            await System.Threading.Tasks.Task.Delay(2000);
            P2PStatusText.Content = "?";
        }
    }
    catch (Exception ex)
    {
    System.Windows.MessageBox.Show("Помилка: " + ex.Message);
    P2PStatusText.Content = "?";
    }
}
```

Додаток Г – Кнопка створення задачі за допомогою TaskSheduler

```
 private void CreateScheduledTask_Click(object sender, RoutedEventArgs e)
 {

     try
     {
         if (selectedFiles == null || selectedFiles.Length == 0 ||
string.IsNullOrWhiteSpace(selectedFolder))
         {
             MessageBox.Show("Оберіть файли та папку призначення.");
             return;
         }

         using (TaskService ts = new TaskService())
         {
             TaskDefinition td = ts.NewTask();
             string fileArgs = string.Join(";", selectedFiles);
             bool IsGglChecked = false;
             td.RegistrationInfo.Description = "Backuper – копіювання
файлів";


             if (OnceADayTask.IsChecked == true)
             {
                 if (!TimeSpan.TryParse(TimeBox.Text, out TimeSpan time))
                 {
                     MessageBox.Show("Невірний формат часу. Використовуйте
формат ГГ:ХХ");
                     return;
                 }

                 var dailyTrigger = new DailyTrigger
                 {
                     DaysInterval = 1,
                     StartBoundary = DateTime.Today + time
                 };
                 td.Triggers.Add(dailyTrigger);
             }
             else if (EveryTimeTask.IsChecked == true)
             {
                 if (HoursChoice.SelectedItem == null ||
MinutesChoice.SelectedItem == null)
                 {
                     MessageBox.Show("Оберіть інтервал повторення.");
                     return;
                 }

                 int hours =
int.Parse(((ComboBoxItem)HoursChoice.SelectedItem).Content.ToString());
```

```
                int                          minutes                          =
int.Parse(((ComboBoxItem)MinutesChoice.SelectedItem).Content.ToString());

                var repeatInterval = new TimeSpan(hours, minutes, 0);
                if (repeatInterval.TotalMinutes < 1 || hours == 0 && minutes
== 0)
                {
                    MessageBox.Show("Інтервал  повинен  бути  не  менше  1
хвилини.");
                    return;
                }

                var trigger = new TimeTrigger
                {
                    StartBoundary = DateTime.Now + TimeSpan.FromMinutes(1),
                    Repetition  =  new  RepetitionPattern(repeatInterval,
TimeSpan.FromDays(1))
                };

                if(CheckSendToGoogle.IsChecked==true)
                {
                UploadToGoogleDrive(fileArgs);
                IsGglChecked= true;
                }

                TimeSpan duration = GetDurationFromComboBox();
                if (duration != TimeSpan.MaxValue)
                {
                    trigger.EndBoundary = DateTime.Now + duration;
                }
                td.Triggers.Add(trigger);
            }

            else
            {
                MessageBox.Show("Оберіть режим запуску.");
                return;
            }
            string                       tokenPath                       =
Path.Combine(AppDomain.CurrentDomain.BaseDirectory,              "google",
"token.json");
            string     arguments     =     $"--files=\"{fileArgs}\"     --
to=\"{selectedFolder}\"           --google=\"{IsGglChecked}\"           --
token=\"{tokenPath}\"";
            string workerPath = System.IO.Path.Combine(
            AppDomain.CurrentDomain.BaseDirectory,
            "ReleaseWorker1",
            "BackuperWorker.exe");
            string workingDir = Path.GetDirectoryName(workerPath);

            td.Actions.Add(new      ExecAction(workerPath,      arguments,
```

```
workingDir));
            ts.RootFolder.RegisterTaskDefinition("Backuper_Custom_File",
td);
            MessageBox.Show("Планувальник задач створено.");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Помилка при створенні планувальника: " +
ex.Message);
    }
 }
```