НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
Імені ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра системного програмування і спеціалізованих комп'ютерних систем

До захисту допущено:
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ

"___" _____ 2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою
«Системне програмування та спеціалізовані комп'ютерні системи»
спеціальності 123 «Комп'ютерна інженерія»
на тему: «Клієнт-серверна система резервного копіювання зашифрованих файлів»

Виконав:
студент IV курсу, групи КВ-11
Ніщик Тарас Андрійович _____

Керівник:
доцент каф. СПСКС, д-р філ.
Сульма Ольга Костянтинівна _____

Консультант з нормоконтролю:
доц. каф. СПСКС, к.т.н., доцент, Клятченко Я.М. _____

Рецензент:
доцент каф. ПЗКС, к.т.н. Саяпіна Інна Олександрівна _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань
Студент _____

Київ – 2025 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
Імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики
Кафедра системного програмування і спеціалізованих
Комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)
Освітньо-професійною програма
«Системне програмування та спеціалізовані
комп'ютерні системи»
Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ

"___" червня 2025 р.

ЗАВДАННЯ
на дипломний проект студента
Ніщику Тарасу Андрійовичу

1. Тема проєкту «Клієнт-серверна система резервного копіювання зашифрованих файлів», керівник проекту Сулема Ольга Костянтинівна, д-р філ., затверджені наказом по університету від 29 травня 2025 р. №1808-С
2. Термін подання студентом проєкту ___
3. Вихідні дані до проєкту див. Технічне завдання
4. Зміст пояснювальної записки
   — Аналіз існуючих рішень
   — Опис використаних технологій
   — Розробка програмного забезпечення
   — Тестування програми
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо):
   — Схема взаємодії модулів програми;
   — Алгоритм обробки створення резервної копії;
   — Алгоритм створення токену для авторизації;
   — Схема збереження файлу на сервері.

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я.М., к.т.н., доцент кафедри СПСКС		

7. Дата видачі завдання 17 квітня 2025 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення літератури за тематикою роботи	17.04.2025	
2	Розроблення та узгодження технічного завдання	27.04.2025	
3	Аналіз існуючих рішень	28.04.2025	
4	Розроблення структури додатку	05.05.2025	
5	Програмна реалізація додатку	15.05.2025	
6	Розроблення дизайну додатку	20.05.2025	
7	Тестування додатку	21.05.2025	
8	Підготовка матеріалів першого розділу дипломного проєкту	25.05.2025	
9	Підготовка матеріалів другого розділу дипломного проєкту	29.05.2025	
10	Підготовка матеріалів третього розділу дипломного проєкту	07.05.2025	
11	Підготовка матеріалів графічної частини проєкту	03.06.2025	
12	Оформлення технічної документації проєкту	06.06.2025	

Студент _____ Тарас НІЩИК

Керівник проєкту _____ Ольга СУЛЕМА

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (53 с., 35 рис., 0 табл., список використаної літератури з 16 найменувань, 6 додатків).

Об’єкт розроблення – клієнт-серверна система резервного копіювання зашифрованих файлів з можливістю їх відновлення на віддаленому пристрої.

Програма дозволяє зберігати та відновлювати зашифровані файли за допомогою автоматичних та ручних резервних копій.

В ході розроблення:

– проведено аналіз наявних програм для створення резервних копій;

– проведено аналіз наявних бібліотек для роботи з складними процеси шифрування;

– розроблена система автоматичного та ручного створення резервних копій;

– розроблена система відновлення збережених резервних копій;

– розроблено зручний інтерфейс для відображення усіх деталей копіювання зашифрованих файлів.

При розробленні системи використано мову програмування C# та можливості фреймворку для роботи зі кросплатформеними застосунками .NET MAUI та ORM-фреймворком Entity Framework Core. Середовище розроблення – Visual Studio.

Ключові слова: копіювання, .NET, блок, меню, кнопка, запит.

ANNOTATION

The qualification work includes an explanatory note (53 p., 35 fig., 0 tables, a list of used literature from 16 sources, 6 appendices).

The object of development is the client-server system for encrypted file backup.

The program allows storing and restoring encrypted files using both automatic and manual backup mechanisms.

In the course of development:

– analysis of existing programs for creating backups was carried out;

– analysis of available libraries for working with complex encryption processes was conducted;

– a system for automatic and manual backup creation was developed;

– a system for restoring saved backups was developed;

– a user-friendly interface for displaying all details of encrypted file backup operations was created.

The system was developed using the C# programming language and the capabilities of the .NET MAUI framework for cross-platform applications, along with the ORM framework Entity Framework Core. The development environment is Visual Studio.

Keywords: backup, .NET, block, menu, button, request.

Зм.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045440.002 ТЗ	Клієнт-серверна система	4		
			резервного копіювання			
			зашифрованих файлів			
			Технічне завдання			
	А4	ІАЛЦ.045440.003 ТП	Клієнт-серверна система	2		
			резервного копіювання			
			зашифрованих файлів			
			Відомість технічного проєкту			
	А4	ІАЛЦ.045440.004 ПЗ	Клієнт-серверна система	53		
			резервного копіювання			
			зашифрованих файлів			
			Пояснювальна записка			
	А4	ІАЛЦ.045440.005 Д1	Клієнт-серверна система	1		
			резервного копіювання			
			зашифрованих файлів			
			Схема взаємодії модулів			
			програми			

					ІАЛЦ.045440.001 ОА	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		Ніщик Т. А.			Клієнт-серверна система резервного копіювання зашифрованих файлів **Опис альбома**	*Літ.* / *Аркуш* / *Аркушів*: 1 / 2
Перевірив		Сулема О. К.				
Н.контр.		Клятченко Я.М.				КПІ ім. Ігоря Сікорського ФПМ КВ-11
Затвердив		Романкевич В.О.				

Зм.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045440.006 Д2	Клієнт-серверна система	1		
			резервного копіювання			
			зашифрованих файлів			
			Алгоритм обробки створення			
			резервної копії			
	А4	ІАЛЦ.045440.007 Д3	Клієнт-серверна система	1		
			резервного копіювання			
			зашифрованих файлів			
			Алгоритм створення токену			
			для авторизації			
	А4	ІАЛЦ.045440.008 Д4	Клієнт-серверна система	1		
			резервного копіювання			
			зашифрованих файлів			
			Схема збереження файлу на			
			сервері			
		Диск CD-ROM	Текст пояснювальної записки	1		
			Текст анотації			
			Код програми			
			Презентація дипломного			
			проєкту			
			Графічний матеріал			

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Клієнт-серверна система резервного копіювання зашифрованих файлів».

Галузь застосування: створення швидких і масштабних резервних копій важливих файлів у зашифрованому вигляді.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розроблення є завдання на дипломне проєктування на здобуття першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою цього проєкту є створення легкої у використанні та освоєнні клієнт-серверної системи для створення резервних копій.

4. ДЖЕРЕЛА РОЗРОБКИ

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

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до програмного продукту, що розробляється

- Сумісність з будь-якою операційною системою Windows, Mac;

- Можливість автоматичного та ручного резервного копіювання;
- Легкість у використанні та освоєнні програми;

5.2 Вимоги до апаратного забезпечення

- Процесор: Intel Core i3-2100 і вище;
- Оперативна пам’ять: мінімум 4 Гб;
- Наявність відеокарти;

5.3 Вимоги до програмного та апаратного забезпечення користувача

- Операційна система Windows, або Mac;

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів
1	Вивчення літератури за тематикою роботи	16.04.2025
2	Розроблення та узгодження технічного завдання	27.04.2025
3	Аналіз наявних рішень	29.04.2025
4	Розроблення структури системи	10.05.2025
5	Програмна реалізація системи	12.05.2025
6	Розроблення дизайну системи	17.05.2025
7	Тестування системи	20.05.2025
8	Підготовка матеріалів першого розділу дипломного проєкту	22.05.2025
9	Підготовка матеріалів другого розділу дипломного проєкту	24.05.2025
10	Підготовка матеріалів третього розділу дипломного проєкту	10.05.2025
11	Підготовка матеріалів графічної частини проєкту	01.06.2025
12	Оформлення технічної документації проєкту	01.06.2025
13	Попередній огляд матеріалів диплому на кафедрі	05.06.2025

Зм.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045440.004 ПЗ	Клієнт-серверна система	53		
			резервного копіювання			
			зашифрованих файлів			
			Пояснювальна записка			
	А4	ІАЛЦ.045440.005 Д1	Клієнт-серверна система	1		
			резервного копіювання			
			зашифрованих файлів			
			Схема взаємодії модулів			
			програми			
	А4	ІАЛЦ.045440.006 Д2	Клієнт-серверна система	1		
			резервного копіювання			
			зашифрованих файлів			
			Алгоритм обробки створення			
			резервної копії			
	А4	ІАЛЦ.045440.007 Д3	Клієнт-серверна система	1		
			резервного копіювання			
			зашифрованих файлів			
			Алгоритм створення токену			
			для авторизації			

					ІАЛЦ.045440.003 ТП	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		Ніщик Т. А.			Клієнт-серверна система резервного копіювання зашифрованих файлів **Відомість технічного проєкту**	*Літ.* / *Аркуш* / *Аркушів*: 1 / 2
Перевірив		Сулема О. К.				
Н.контр.		Клятченко Я.М.				КПІ ім. Ігоря Сікорського ФПМ КВ-11
Затвердив		Романкевич В.О.				

Зм.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045440.008 Д4	Клієнт-серверна система	1		
			резервного копіювання			
			зашифрованих файлів			
			Схема збереження файлу на			
			сервері			
		Диск CD-ROM	Текст пояснювальної записки	1		
			Текст анотації			
			Код програми			
			Презентація дипломного			
			проєкту			
			Графічний матеріал			

						Арк.
					ІАЛЦ.045440.003 ТЗ	2
Зм.	Арк	№ докум.	Підпис	Дата		

Пояснювальна записка

до дипломного проєкту

на тему: «Клієнт-серверна система резервного копіювання зашифрованих файлів»

Київ – 2025

ЗМІСТ

ДОДАТКИ

Додаток А. ІАЛЦ.045440.005 Д1. Схема взаємодії модулів програми.

Додаток Б. ІАЛЦ.045440.006 Д2. Алгоритм обробки створення резервної копії.

Додаток В. ІАЛЦ.045440.007 Д3. Алгоритм створення токени для авторизації.

Додаток Г. ІАЛЦ.045440.008 Д4. Схема збереження файлу на сервері.

Додаток Д. Презентація.

Додаток Е. Фрагмент коду.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

Backup (англ. резервне копіювання) — процес створення копій файлів або баз даних, що зберігаються окремо від основного джерела з метою їх відновлення у разі втрати, пошкодження чи знищення основних даних.

MAUI (англ. .NET Multi-platform App UI) — фреймворк від Microsoft для створення крос-платформених додатків з єдиним кодовим базисом, що підтримує Windows, Android, iOS та macOS, забезпечуючи спільний інтерфейс користувача.

.NET — програмна платформа з відкритим кодом, розроблена Microsoft, яка надає широкі можливості для створення різноманітних застосунків — від вебсайтів і десктоп-додатків до мобільних і хмарних сервісів.

EF Core (англ. Entity Framework Core) — ORM (Object-Relational Mapping) фреймворк з відкритим кодом для платформи .NET, що дозволяє працювати з базами даних за допомогою об'єктів C#, приховуючи SQL-запити та спрощуючи доступ до даних.

ВСТУП

У сучасному цифровому середовищі безпека та збереження даних є критично важливими аспектами функціонування як для окремих користувачів, так і для організацій. Щодня генеруються великі обсяги інформації, частина з якої є конфіденційною або має особливу цінність. Втрата таких даних унаслідок збоїв системи, вірусних атак, людських помилок або фізичного пошкодження обладнання може призвести до серйозних наслідків — як фінансових, так і репутаційних.

Одним з найефективніших способів запобігання втраті інформації є використання систем резервного копіювання. Водночас, з огляду на загрози несанкціонованого доступу, особливо актуальним стає питання захисту даних під час збереження та передавання. Саме тому шифрування інформації є невіддільною складовою сучасних рішень для резервного копіювання.

У цій дипломній роботі розробляється клієнт-серверна система резервного копіювання зашифрованих файлів, яка поєднує функціональність автоматизованого збереження даних з гарантією їхньої конфіденційності. Такий підхід забезпечує користувачам можливість відновлення втрачених файлів, зберігаючи при цьому їхню захищеність навіть у разі доступу до серверної частини сторонніми особами.

1. АНАЛІЗ НАЯВНИХ РІШЕНЬ

1.1 Загальна інформація про системи резервного копіювання

Системи резервного копіювання з’явилися як логічна відповідь на одну з найстаріших загроз в обчислювальній техніці — втрату даних. Ще з моменту появи перших комп’ютерів у середині XX століття стало зрозуміло, що цифрова інформація, хоч і зручна у використанні, є надзвичайно вразливою до зовнішніх факторів: від фізичного зносу носіїв і технічних збоїв до помилок користувачів. Тоді резервне копіювання виконувалося вручну — дані дублювалися на магнітні стрічки, диски або інші зовнішні носії, які зберігались окремо від основного обладнання. З роками, зі зростанням обсягів інформації та ускладненням ІТ-систем, такі підходи стали неефективними, що стимулювало розвиток автоматизованих систем резервного копіювання.

Сьогодні під системою резервного копіювання мається на увазі програмно-технічне рішення, яке забезпечує створення копій даних з можливістю їх подальшого відновлення у разі їх втрати, пошкодження або недоступності. Це не просто дублювання файлів, а комплексна інфраструктура, яка враховує періодичність копіювання, захист від несанкціонованого доступу, шифрування, оптимізацію обсягу даних та швидкість відновлення. Резервне копіювання може здійснюватися як локально (на зовнішні диски чи сервери), так і у віддаленому або хмарному середовищі, з використанням спеціалізованого програмного забезпечення.

Значення таких систем важко переоцінити. У часи, коли майже вся особиста, комерційна та державна інформація зберігається в електронному вигляді, навіть короткочасна втрата доступу до даних може спричинити серйозні наслідки: порушення бізнес-процесів, витоки конфіденційної інформації, фінансові збитки чи втрату довіри клієнтів. Особливо гостро проблема постає в умовах зростання кількості кіберзагроз, зокрема програм-вимагачів, які шифрують вміст жорсткого диска з вимогою викупу. Єдиним

надійним способом відновити дані в такому випадку лишається наявність актуальної резервної копії.

У сучасному підході до резервного копіювання важливу роль відіграє не лише збереження даних, але й їхній захист. Створення зашифрованих резервних копій, контроль доступу, перевірка цілісності файлів та використання безпечних каналів передачі — все це стає обов'язковими вимогами до сучасних рішень. Крім того, новітні системи підтримують функції інкрементального копіювання (збереження лише змінених частин даних), дедуплікації, створення графіків автоматичного бекапу та зберігання кількох історичних версій файлів.

Таким чином, системи резервного копіювання перетворилися з простого інструменту дублювання файлів на ключовий елемент комплексної стратегії збереження та захисту інформації. Їх впровадження є не рекомендацією, а необхідністю для всіх, хто має справу з цифровими даними — від окремих користувачів до великих корпорацій. У цьому контексті актуальним стає питання розробки безпечних, гнучких і доступних рішень, що відповідають сучасним вимогам до збереження і захисту даних.

1.2 Аналіз наявних систем резервного копіювання

Сучасний ринок програм для резервного копіювання вирізняється значною різноманітністю. Існують як прості утиліти для базового збереження даних, так і потужні системи, розраховані на використання в складних IT-інфраструктурах. Кожне з рішень має свої особливості, рівень функціональності та підхід до захисту інформації. У цьому розділі буде здійснено огляд найбільш популярних програм у цій галузі, з акцентом на аналіз їх сильних і слабких сторін.

1.2.1 Огляд Veeam

Серед сучасних систем резервного копіювання Veeam Backup & Replication посідає провідне місце завдяки своїй надійності, функціональності та широким можливостям масштабування. Ця система була розроблена компанією Veeam Software як рішення для резервного копіювання та відновлення віртуальних машин, однак з часом вона значно розширила сферу свого застосування, охопивши також фізичні сервери, робочі станції та хмарні середовища.

Veeam підтримує інтеграцію з такими платформами, як VMware vSphere, Microsoft Hyper-V, а також хмарними сервісами на зразок AWS, Microsoft Azure та Google Cloud. Це дає змогу створювати універсальну систему резервного копіювання, незалежно від особливостей IT-інфраструктури. У складі Veeam реалізовано підхід до “Availability for the Always-On Enterprise”, тобто постійну доступність даних для критичних бізнес-процесів без переривань(рис 1.1).

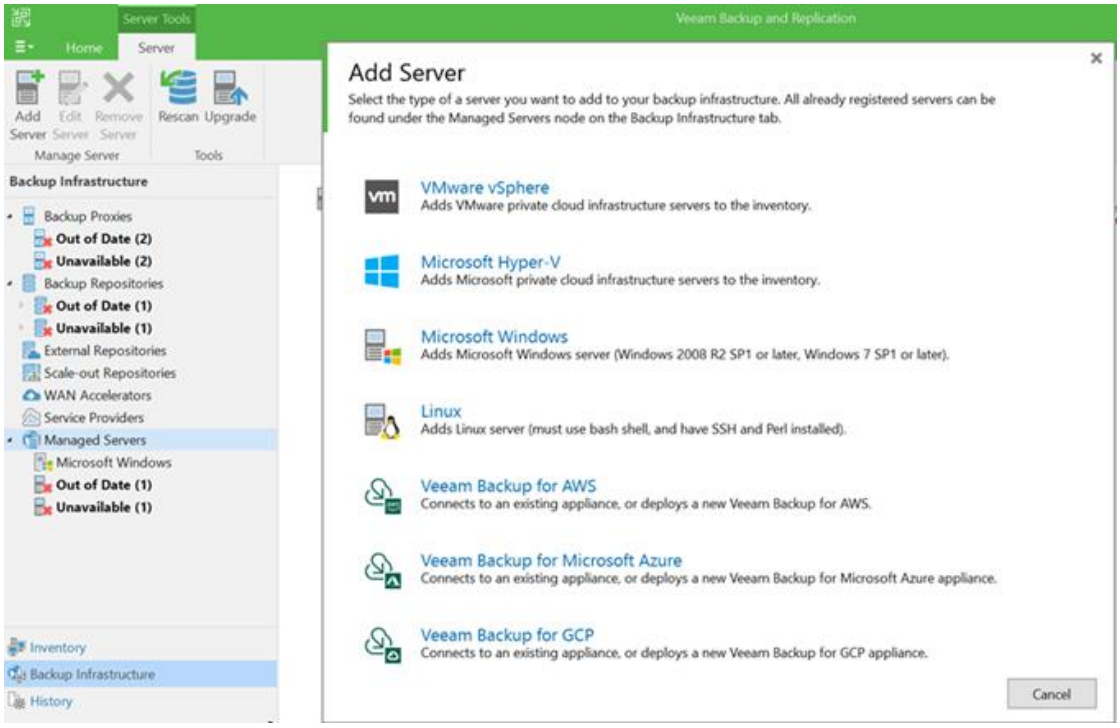


Рис 1.1 – інтеграція з хмарними платформами

Однією з ключових особливостей Veeam є інкрементальне копіювання, що означає збереження лише змінених блоків даних після початкового повного бекапу. Це суттєво скорочує час створення копії та зменшує навантаження на систему зберігання. Для подальшого зменшення обсягів збережених даних використовується дедуплікація та компресія — тобто видалення повторюваних фрагментів і стискання файлів без втрати інформації.

Архітектура системи побудована за принципом розподіленості: Backup Server виконує координацію процесів, Proxy Server відповідає за транспортування даних, а Repository — за їх зберігання. Такий підхід дає змогу масштабувати систему відповідно до обсягу даних і вимог до продуктивності. Крім того, можливо створювати кілька рівнів зберігання: локальні, віддалені, хмарні та навіть гібридні.

На окрему увагу заслуговує функціональність реплікації, яка забезпечує копіювання віртуальних машин у режимі реального часу на інші фізичні або віртуальні середовища. У разі аварійної ситуації (наприклад, виходу з ладу основного центру даних) можна швидко переключити навантаження на репліку, мінімізуючи час простою системи.

Система також надає можливості для гнучкого відновлення: користувач може відновити не лише всю машину або диск, а й окремі файли, каталоги, електронні листи чи об’єкти баз даних. Така деталізація процесу відновлення дозволяє ефективно реагувати навіть на дрібні інциденти без потреби відновлювати великі обсяги інформації (рис 1.2).

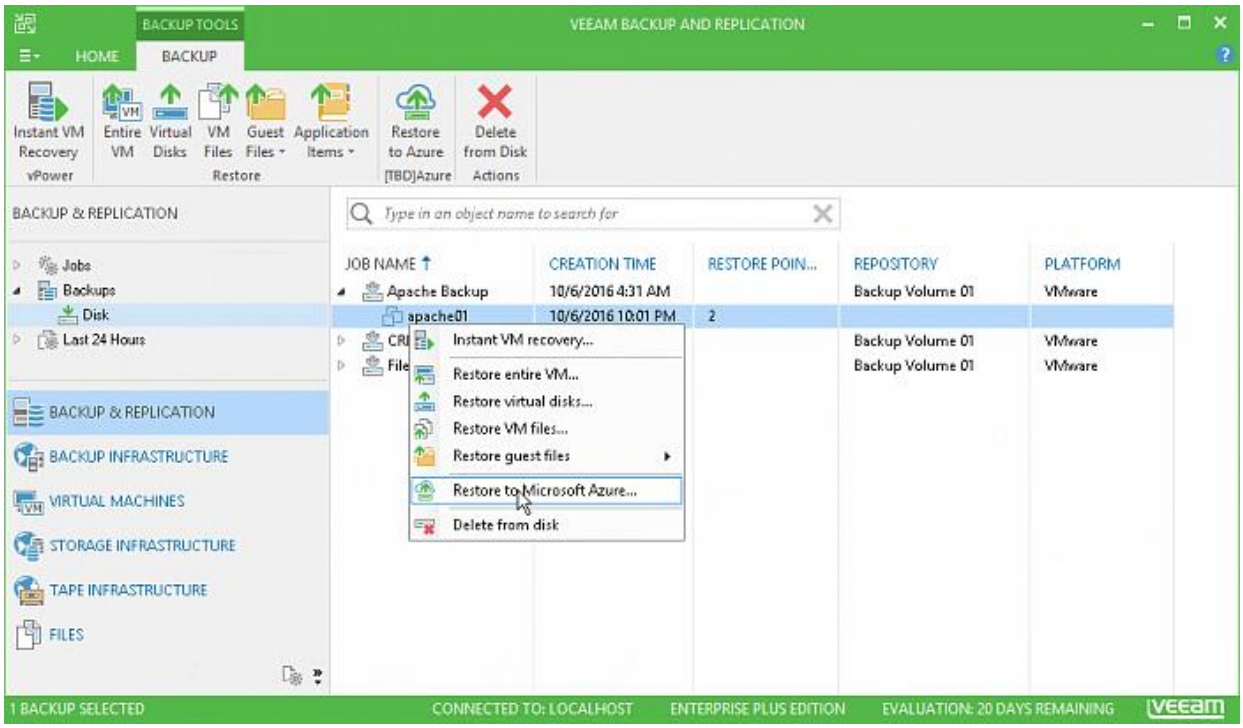


Рис. 1.2 – можливість гнучкого відновлення

Для моніторингу та аудиту в Veeam реалізовано централізовану панель керування із звітністю, журналами подій, графіками виконання та сповіщеннями про помилки. Це дозволяє адміністраторам системи бачити повну картину стану резервного копіювання в реальному часі та своєчасно реагувати на потенційні проблеми.

Veeam також підтримує зашифроване збереження даних, а також контроль доступу до кожної операції. Це дозволяє реалізувати політику безпеки згідно з сучасними вимогами — зокрема, шифрування здійснюється вже під час створення копії, ще до збереження в репозиторій.

Завдяки широким можливостям інтеграції, детальному налаштуванню процесів та надійності, Veeam Backup & Replication став стандартом де-факто для середнього і великого бізнесу у сфері резервного копіювання. Водночас, його професійна спрямованість робить систему малопридатною для домашнього використання чи малих проєктів, де не потрібна така глибина функціоналу.

Переваги Veeam:

– Потужна функціональність для резервного копіювання, реплікації та відновлення;

– Підтримка віртуальних, фізичних і хмарних середовищ;

– Інкрементальне копіювання, дедуплікація, компресія;

– Гнучкі сценарії відновлення (від файлу до цілої системи);

– Високий рівень автоматизації та планування задач;

– Реплікація з можливістю аварійного перемикання;

– Захист доступу та шифрування даних;

– Центральна консоль управління з детальною звітністю;

– Можливість масштабування під великі інфраструктури.

Недоліки Veeam:

– Висока вартість ліцензії, особливо для невеликих організацій;

– Закритий вихідний код — відсутність можливості кастомізації на низькому рівні;

– Високі вимоги до апаратного забезпечення;

– Складність початкового налаштування та впровадження;

– Надлишковість функціональності для простих сценаріїв (наприклад, резервного копіювання окремих файлів).

1.2.2 Огляд Duplicati

Duplicati — це безкоштовна система резервного копіювання з відкритим програмним кодом, призначена для створення зашифрованих резервних копій файлів з можливістю їх збереження як локально, так і на хмарних платформах. Розробка цієї системи орієнтована насамперед на кінцевого користувача та невеликі організації, яким потрібне просте, але функціональне рішення для автоматизованого резервного копіювання з підтримкою шифрування.

Duplicati підтримує збереження даних у понад 20 хмарних сховищах, серед яких Google Drive, Dropbox, Microsoft OneDrive, Amazon S3, Backblaze

B2, FTP/SFTP-сервери, WebDAV та інші. Завдяки цьому користувачі можуть гнучко обирати зручний для себе спосіб зберігання копій без прив'язки до конкретної платформи. Усі резервні копії в Duplicati створюються у вигляді інкрементальних блоків, що значно зменшує обсяг передаваних даних і пришвидшує процес копіювання.

Головною перевагою Duplicati є його вбудоване шифрування на стороні клієнта. Для цього використовується алгоритм AES-256, який забезпечує високий рівень захисту інформації ще до її передачі в зовнішнє сховище. Це означає, що навіть у разі доступу до сховища третіх осіб (наприклад, з боку провайдера хмари), дані лишаються недоступними без відповідного ключа.

Duplicati функціонує як фоновий сервіс із веб-інтерфейсом, що дозволяє керувати всіма параметрами через браузер. Користувач може створювати завдання резервного копіювання, налаштовувати розклад, шифрування, цілі сховища, правила виключення певних файлів і директорій, а також переглядати журнали виконання. Окрім цього, система підтримує перевірку цілісності резервних копій, що дозволяє своєчасно виявити пошкоджені дані (рис. 1.3).

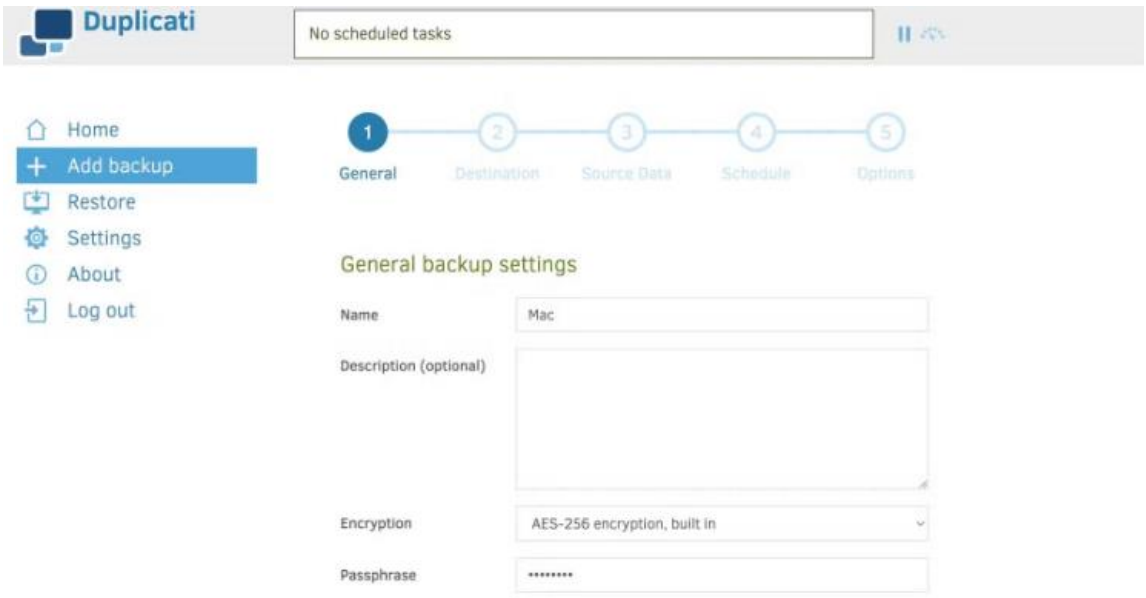


Рис. 1.3 – веб-версія додатку Duplicati

Архітектурно Duplicati реалізований як кросплатформенна програма, що працює на Windows, macOS і Linux. В основі лежить .NET/Mono, що забезпечує стабільну роботу на більшості систем, хоча іноді це може спричиняти додаткові залежності у Linux-середовищах. Duplicati підтримує автоматичну очистку старих версій копій на основі заданих політик, а також створення резервних копій за розкладом з можливістю надсилання email-сповіщень про результати.

Попри свою орієнтацію на простоту, система дозволяє налаштувати багато параметрів вручну, що оцінять досвідчені користувачі. Проте інтерфейс і філософія продукту залишаються достатньо зрозумілими навіть для тих, хто не має глибоких технічних знань (рис 1.4).

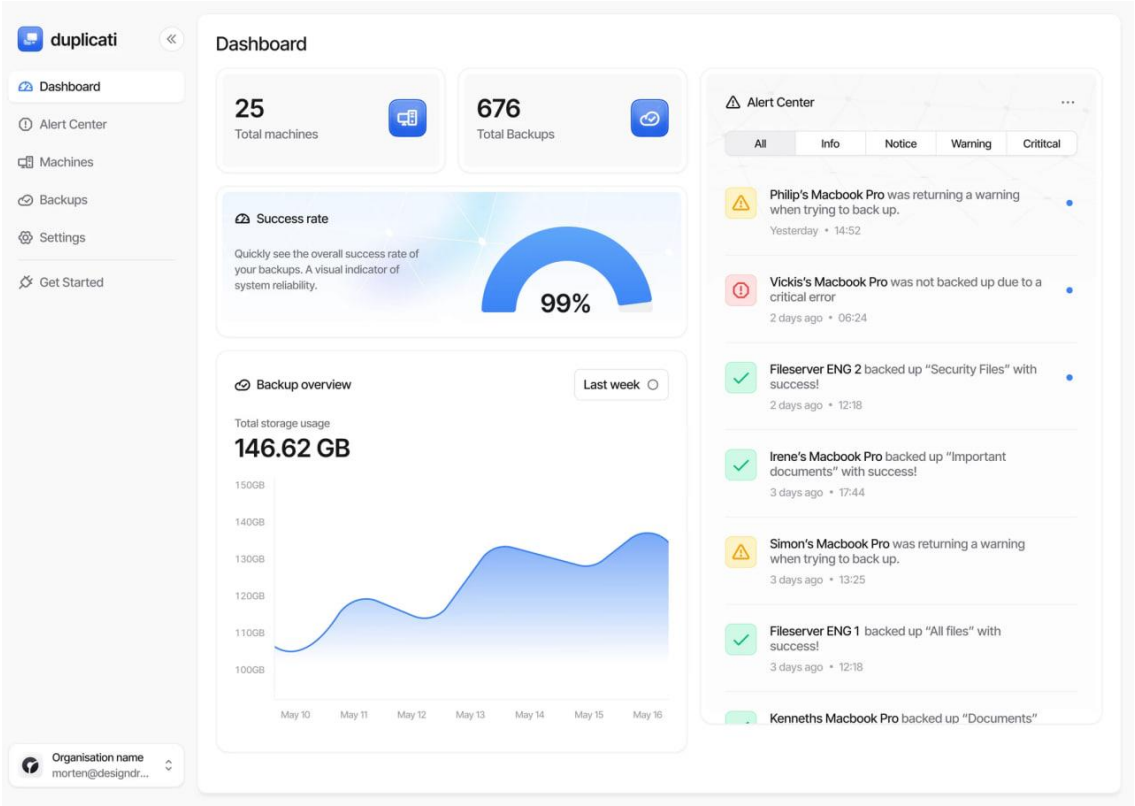


Рис 1.4 – графічний інтерфейс додатку для ПК

Duplicati добре підходить для резервного копіювання файлів і папок, але не підтримує збереження цілих образів системи або віртуальних машин. Також слід враховувати, що хоча проєкт є активним, його розвиток повільніший у

порівнянні з комерційними рішеннями, і в певних випадках користувачам доводиться самостійно усувати проблеми, покладаючись на документацію або спільноту.

Переваги Duplicati:

– Повністю безкоштовна і з відкритим кодом;

– Шифрування на стороні клієнта (AES-256);

– Підтримка великої кількості хмарних і локальних сховищ;

– Інкрементальне копіювання з компресією;

– Вебінтерфейс для налаштувань і моніторингу;

– Кросплатформенність (Windows, macOS, Linux);

– Гнучке налаштування завдань, виключень і розкладу;

– Підтримка перевірки цілісності та автоматичного очищення старих копій.

Недоліки Duplicati:

– Не підтримує резервне копіювання цілих дисків або системних образів;

– Потребує встановлення додаткових компонентів у Linux-середовищах (Mono/.NET);

– Відсутній повноцінний модуль відновлення системи після критичного збою;

– Менш стабільна у порівнянні з комерційними рішеннями при великих обсягах даних;

– Інтерфейс не перекладено повністю на всі мови, документація — переважно англійською;

– Відсутність офіційної технічної підтримки — допомога лише через спільноту.

1.2.3 Огляд Microsoft OneDrive

Microsoft OneDrive — це хмарний сервіс для зберігання файлів, розроблений корпорацією Microsoft. Хоча його первинне призначення полягає в синхронізації документів між пристроями та зберіганні даних у хмарі, OneDrive також часто використовується як інструмент резервного копіювання для кінцевих користувачів. Його інтеграція з операційною системою Windows та продуктами Microsoft 365 робить цей сервіс одним із найпоширеніших способів збереження персональних файлів у хмарному середовищі.

Основна ідея OneDrive — забезпечити користувачу безперервний доступ до власних файлів з будь-якого пристрою, підключеного до Інтернету. Це реалізується через синхронізацію локальної папки OneDrive з хмарним сховищем, завдяки чому усі зміни у вмісті автоматично відображаються в обох середовищах. Сервіс підтримує збереження документів, фотографій, відео, а також автоматичне резервне копіювання певних системних тек Windows (наприклад, «Робочий стіл», «Документи», «Зображення») (рис 1.5).

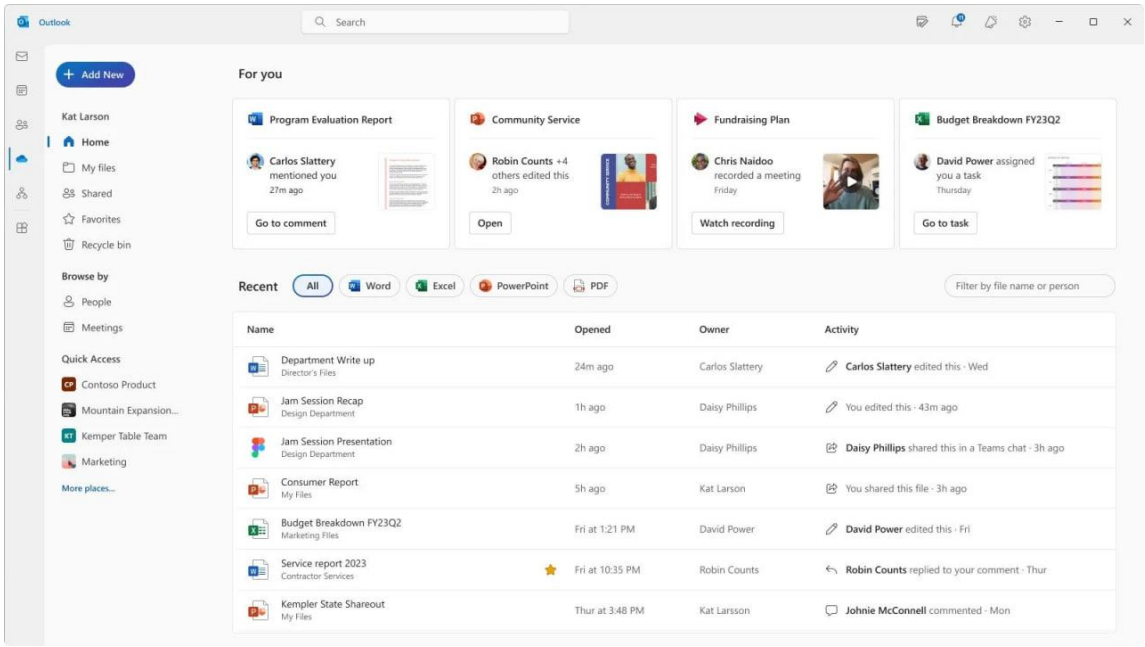


Рис 1.5 – графічний інтерфейс Microsoft OneDrive

Як інструмент резервного копіювання, OneDrive є доволі простим — він не має гнучких механізмів створення інкрементальних копій, версіонування на рівні блоків або повного збереження системних образів. Натомість сервіс зберігає версії змінених файлів, що дозволяє повернутися до попереднього стану документа, якщо була зроблена помилкова зміна або видалення. Кількість доступних попередніх версій залежить від типу облікового запису (персональний чи корпоративний) і тарифного плану.

Однією з важливих функцій OneDrive є вбудований захист від втрати даних у випадку шкідливого програмного забезпечення, такого як програми-вимагачі (ransomware). Сервіс може виявляти підозрілу активність (наприклад, масову зміну файлів) і пропонує користувачу відновити попередній стан усієї файлової системи OneDrive.

Крім того, OneDrive підтримує шифрування даних під час передачі (TLS) та шифрування в хмарному сховищі (AES-256). Проте варто зазначити, що шифрування здійснюється на стороні сервера, а не клієнта, тобто Microsoft технічно має доступ до незашифрованих даних, що є критичним моментом у порівнянні з рішеннями, де реалізовано шифрування на стороні клієнта.

OneDrive зручно інтегрований із Microsoft Office, що дозволяє працювати з документами безпосередньо з хмари в режимі реального часу. Сервіс також пропонує мобільні застосунки та веб-інтерфейс для доступу до файлів, а також функції спільного доступу з налаштуванням прав (читання/редагування, термін дії посилання тощо) (рис 1.6).

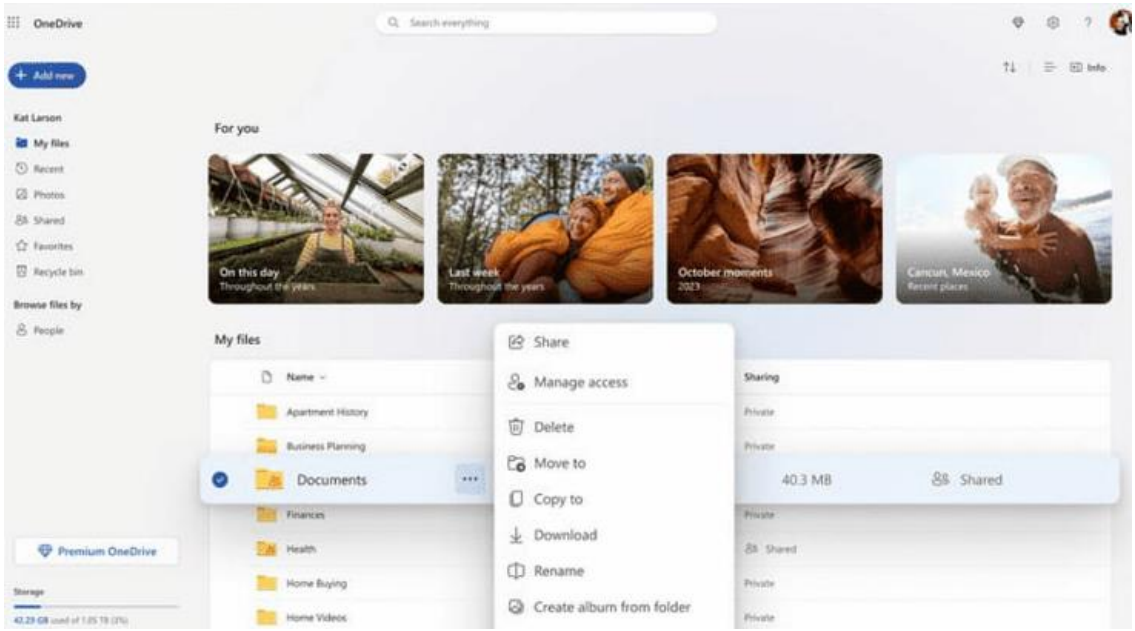


Рис 1.6 – процес роботи з файлами з хмари

Попри простоту використання, Microsoft OneDrive обмежений у порівнянні з повноцінними системами резервного копіювання. Він не дозволяє створювати копії системного реєстру, повних образів системи або вести незалежну історію резервних копій. Тому в дипломному контексті Microsoft OneDrive доцільно розглядати як хмарне сховище з можливостями часткового резервного копіювання, орієнтоване насамперед на збереження персональних даних.

Переваги Microsoft OneDrive:

- Інтеграція з Windows і Microsoft 365;
- Автоматична синхронізація файлів і системних тек;
- Зберігання кількох версій файлів з можливістю відновлення;
- Виявлення атак програм-вимагачів і швидке відновлення;
- Простота використання та доступ із будь-якого пристрою;
- Можливість надання спільного доступу до файлів;
- Вбудоване шифрування передачі та зберігання даних;
- Мобільні та веб-додатки для зручного керування файлами.

Недоліки Microsoft OneDrive:

– Відсутність шифрування на стороні клієнта (дані доступні провайдеру);

– Немає повноцінного резервного копіювання системи або додатків;

– Обмежене версіонування залежно від типу облікового запису;

– Не підтримує гнучкі сценарії резервного копіювання чи політики зберігання;

– Немає можливості повного резервного копіювання на власні сервери;

– Залежність від стабільного підключення до Інтернету;

– Обмежений обсяг хмарного сховища у безкоштовній версії.

1.3 Обґрунтування теми диплому

Попри велику кількість доступних рішень для резервного копіювання, більшість із них виявляються надто складними для пересічного користувача. Вони часто потребують глибоких технічних знань, складного налаштування та значних затрат часу для повноцінного використання. Це створює бар’єр для тих, хто хоче просто і надійно захистити свої дані без занурення в технічні деталі. У зв’язку з цим темою дипломного проєкту було обрано розробку клієнт-серверної системи резервного копіювання, яка буде поєднувати зручність, зрозумілий інтерфейс та базову функціональність, необхідну для надійного збереження інформації. Проєкт передбачає створення інтуїтивно зрозумілого графічного інтерфейсу, що дозволить користувачеві здійснювати резервне копіювання та відновлення без потреби спеціальних знань у сфері IT або кібербезпеки.

2. ОПИС ТЕХНОЛОГІЙ

2.1 База даних PostgreSQL

PostgreSQL — це потужна, високонадійна об'єктно-реляційна система керування базами даних (СКБД) з відкритим програмним кодом, яка відома своєю стабільністю, масштабованістю та відповідністю сучасним вимогам до обробки і збереження даних. Проєкт бере свій початок ще з 1986 року в Каліфорнійському університеті в Берклі, де він розвивався як дослідницький проєкт POSTGRES, а з 1996 року отримав назву PostgreSQL. З того часу система активно підтримується міжнародною спільнотою та постійно розвивається, залишаючись одним із найпопулярніших рішень серед СКБД з відкритим кодом.

На відміну від багатьох інших СКБД, PostgreSQL поєднує в собі традиційний реляційний підхід із потужними об'єктно-орієнтованими можливостями. Це дозволяє не лише працювати з таблицями та зв’язками, а й створювати користувацькі типи даних, функції, тригери, індекси, а також розширювати базу власними модулями. Такий рівень гнучкості робить PostgreSQL ідеальним інструментом як для простих додатків, так і для складних інформаційних систем.

Однією з ключових переваг PostgreSQL є відповідність стандартам SQL, зокрема SQL:2008. Це означає, що більшість складних запитів, підзапитів, віконних функцій, об’єднань та умовних операторів підтримується "з коробки", без необхідності додаткових налаштувань. PostgreSQL також підтримує ACID-транзакції, що гарантує надійність та цілісність даних навіть у разі збоїв або паралельного доступу до однієї таблиці.

На окрему увагу заслуговує підтримка розширених типів даних. PostgreSQL дозволяє працювати з такими структурами, як масиви, JSON, XML, UUID, tsvector (для повнотекстового пошуку) та багатьма іншими. Така різноманітність дає змогу безпосередньо зберігати складні дані без потреби у додатковій обробці на рівні застосунку. У сучасному світі, де все частіше

використовуються змішані моделі зберігання (реляційно + документно), ця гнучкість має велике практичне значення.

PostgreSQL також вирізняється відмінною масштабованістю. Система підтримує реплікацію (як синхронну, так і асинхронну), шардінг, розподілені транзакції та паралельне виконання запитів. Це дає змогу масштабувати проєкти горизонтально та вертикально залежно від потреб, без втрати продуктивності чи стабільності. Саме тому PostgreSQL часто використовується у великих фінансових, телекомунікаційних та наукових системах, де йдеться про обробку мільйонів записів щодня.

Ще одна вагома перевага — високий рівень безпеки. PostgreSQL підтримує різноманітні механізми автентифікації: паролі, SSL/TLS, GSSAPI, LDAP, Kerberos. Адміністратор бази даних може детально налаштовувати права доступу до об'єктів — на рівні таблиць, колонок, функцій тощо. Завдяки цьому забезпечується гнучкий контроль над тим, хто і до яких даних має доступ, що є критично важливим у сучасних умовах зростання кіберзагроз.

Крім того, PostgreSQL є відкритим програмним продуктом, що поширюється під ліцензією PostgreSQL License (сумісною з ліцензією MIT). Це означає, що система повністю безкоштовна для комерційного та особистого використання, і може бути адаптована під будь-які задачі без обмежень. Відкритий код також дозволяє аудіювати безпеку, розширювати функціональність або інтегрувати систему в інші рішення на глибокому рівні.

Важливо відзначити й активну міжнародну спільноту, яка підтримує PostgreSQL. Кожен реліз проходить ретельне тестування, а нові можливості додаються лише після широкої апробації. Завдяки цьому PostgreSQL не лише надійна, а й постійно вдосконалюється відповідно до потреб сучасної розробки.

Загалом, PostgreSQL — це поєднання надійності, відкритості, гнучкості та сучасного функціоналу, що робить її ідеальним вибором для створення широкого спектра програм — від особистих застосунків до великих

корпоративних систем, які потребують високого рівня захисту даних, складних запитів і стабільної роботи.

2.2 .NET 8

.NET 8 — це сучасна кросплатформенна програмна платформа, яка розробляється та підтримується компанією Microsoft. Вона є продовженням єдиної об’єднаної платформи .NET, яка об'єднала можливості попередніх проєктів .NET Framework, .NET Core і Xamarin. Офіційний реліз .NET 8 відбувся у листопаді 2023 року, і ця версія має статус LTS (Long-Term Support), що гарантує її підтримку та оновлення протягом щонайменше трьох років.

.NET 8 є універсальним середовищем розробки, яке дозволяє створювати додатки різного типу — вебзастосунки, десктопні програми, мобільні застосунки, мікросервіси, серверні API, фонові служби, IoT-рішення тощо. Платформа підтримує запуск на різних операційних системах, зокрема Windows, Linux і macOS, та на різних архітектурах, включно з x64 і ARM64.

Однією з ключових переваг .NET 8 є його висока продуктивність. Завдяки численним оптимізаціям на рівні компілятора та виконання, ця версія демонструє значно кращу ефективність у порівнянні з попередніми релізами. Зокрема, удосконалено систему JIT-компіляції, управління пам’яттю та підтримку багатопотокової обробки. Окрім того, .NET 8 підтримує Ahead-of-Time компіляцію (AOT) — механізм, що дозволяє створювати нативні виконувані файли без залежності від середовища виконання .NET, що покращує швидкість запуску та зменшує розмір додатку (рис 2.1).

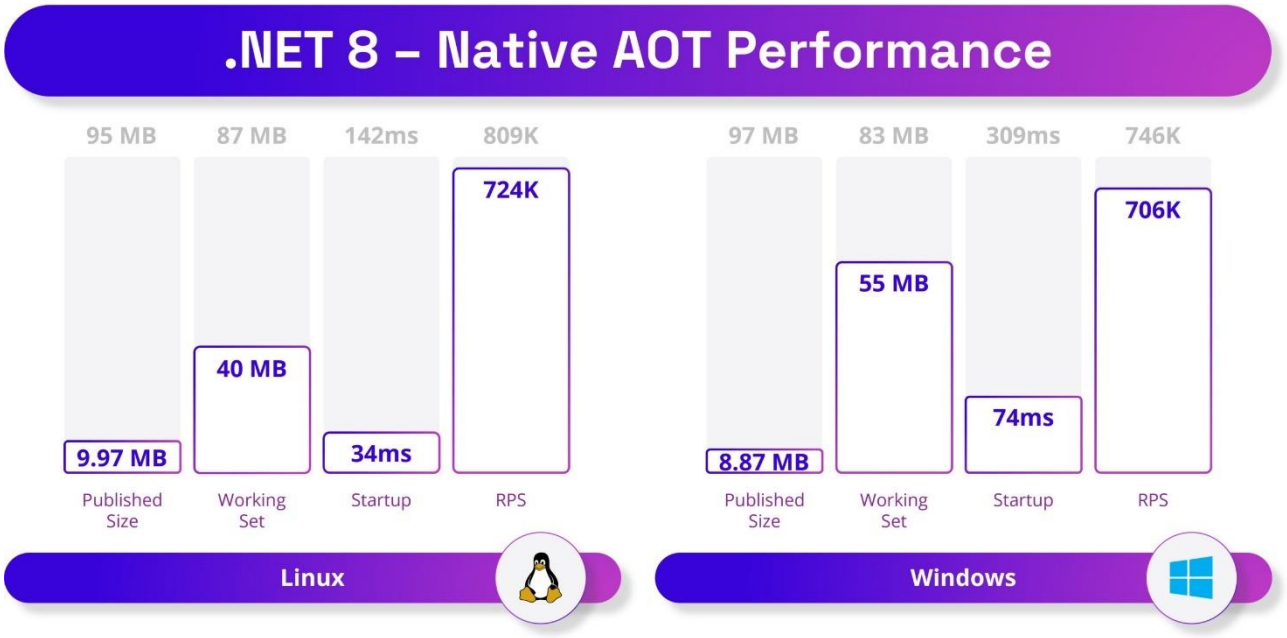


Рис 2.1 – графік результатів покращення продутивності за допомогою AOT

У .NET 8 інтегровано мову програмування C# 12, яка містить низку нововведень для спрощення синтаксису та зменшення шаблонного коду. До нових можливостей C# 12 належать primary constructors для класів, нові способи роботи з колекціями, вдосконалене pattern matching, а також покращення для роботи з інтерфейсами та виразами.

Серцем веброзробки у .NET 8 є ASP.NET Core — високопродуктивний фреймворк для створення вебсайтів, REST API та реального часу додатків із використанням SignalR. ASP.NET Core у цій версії отримав додаткові покращення маршрутизації, продуктивності, підтримки кешування, конфігурації та безпеки. Крім того, активно розвивається технологія Blazor, яка дозволяє створювати інтерактивні вебінтерфейси з використанням C# замість JavaScript — як на сервері, так і в браузері (через WebAssembly) (рис 2.2).

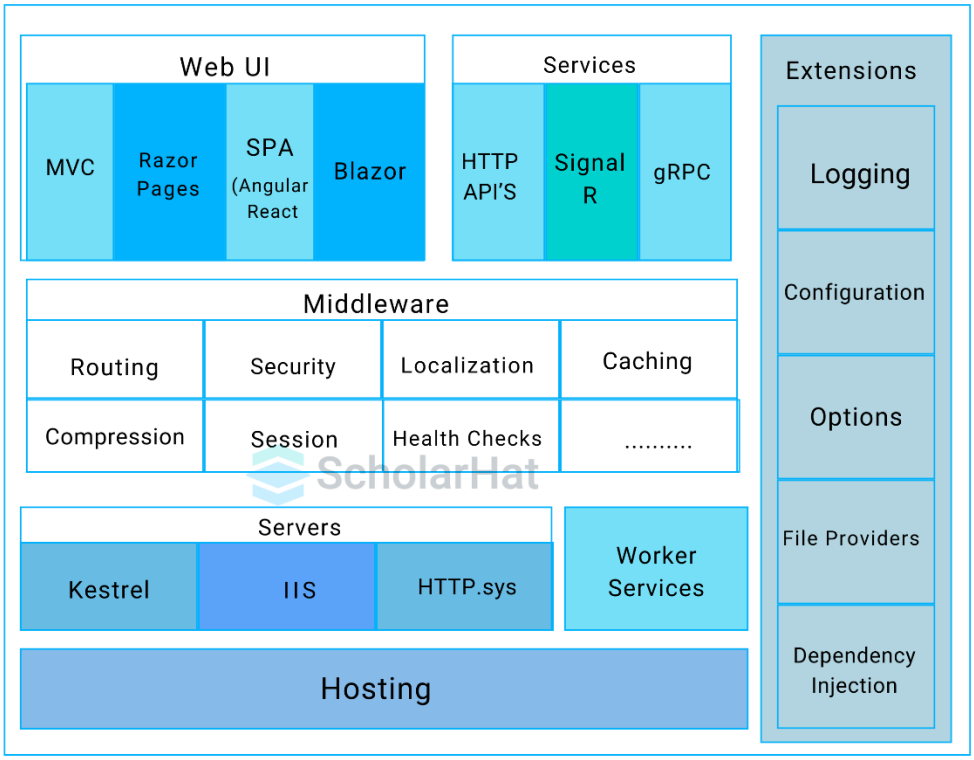


Рис 2.2 – діаграма функціональності ASP.NET Core

Для роботи з базами даних .NET 8 включає Entity Framework Core 8 (EF Core 8) — ORM-фреймворк, що дозволяє взаємодіяти з реляційними СКБД за допомогою об'єктно-орієнтованого підходу. У цій версії EF Core підтримує нові типи даних, роботу з JSON-колонками, покращену генерацію SQL-запитів та продуктивніше відстеження змін у сутностях (рис 2.3).

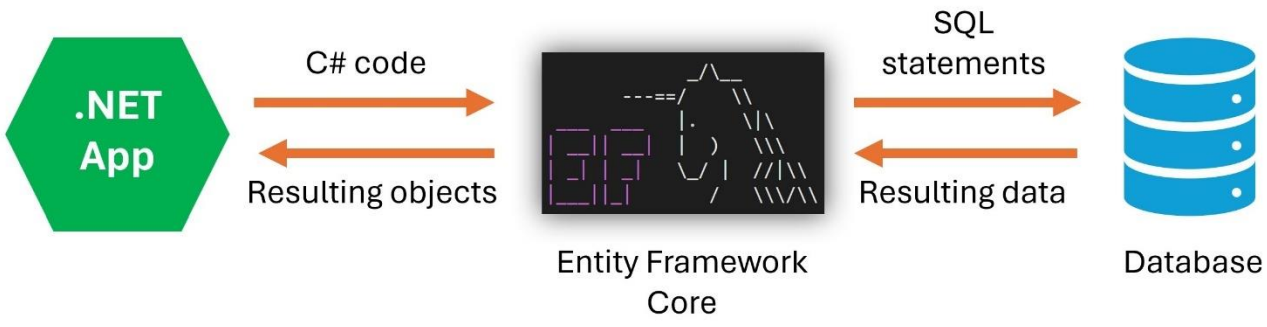


Рис 2.3 – діаграма роботи Entity Framework Core

Ще однією важливою перевагою .NET 8 є широка підтримка хмарних сервісів, зокрема інтеграція з Azure, покращена робота в Docker-контейнерах,

зменшені розміри образів, підтримка розподіленого логування та телеметрії. Завдяки цьому платформа ідеально підходить для побудови сучасних хмарних архітектур, зокрема в Kubernetes-середовищах.

Оскільки .NET 8 є відкритим програмним продуктом з відкритим вихідним кодом, кожен охочий може переглядати, тестувати або навіть робити внески в його розробку через GitHub. Це забезпечує прозорість процесу розвитку, активну підтримку спільноти, а також доступ до тисяч готових бібліотек і пакетів через систему NuGet.

Завдяки своїй гнучкості, високій продуктивності та кросплатформенності, .NET 8 є чудовим вибором для створення як простих, так і складних систем. У контексті дипломного проєкту ця платформа дозволяє реалізувати захищене клієнт-серверне рішення, побудувати REST API, забезпечити авторизацію користувачів, обробку файлів, взаємодію з базою даних PostgreSQL та гнучке масштабування з використанням контейнерних технологій.

2.3 Entity Framework Core

Entity Framework Core (EF Core) — це сучасний об’єктно-реляційний фреймворк (ORM), розроблений компанією Microsoft як частина платформи .NET. Його основна мета — спростити взаємодію з реляційними базами даних, дозволяючи розробникам працювати з інформацією на рівні об'єктів і класів, а не через пряме написання SQL-запитів. EF Core дає змогу ефективно будувати, зчитувати, оновлювати та видаляти дані, використовуючи знайомі засоби мови програмування C#.

На відміну від класичної версії Entity Framework, яка була тісно пов’язана з Windows і .NET Framework, EF Core є кросплатформенним, тобто працює в середовищі .NET Core та .NET 5/6/7/8, і підтримує Windows, Linux та macOS. Такий підхід робить EF Core універсальним інструментом для

розробки серверних рішень, вебзастосунків, мікросервісів і API, що мають потребу у збереженні структурованих даних.

EF Core використовує концепцію Code First, яка дозволяє створювати структуру бази даних безпосередньо з C#-коду, описуючи сутності у вигляді класів, що відображають таблиці, а їхні властивості — як стовпці. На основі цього коду фреймворк генерує міграції, які застосовуються до бази даних і дозволяють поступово змінювати її структуру в процесі розробки. Також підтримується підхід Database First, коли база вже існує, і на її основі генеруються класи моделей.

Важливою складовою EF Core є LINQ (Language Integrated Query) — синтаксис, який дозволяє писати запити до бази у вигляді звичайного C#-коду. Це підвищує безпеку та читабельність запитів, автоматично захищає від SQL-ін’єкцій та значно прискорює процес розробки. Також підтримуються завантаження зв’язаних об’єктів (eager/lazy/explicit loading), транзакції, кешування в контексті та обробка змін у сутностях через трекінг стану.

EF Core підтримує роботу з різними базами даних завдяки системі провайдерів. Найпопулярніші серед них: Microsoft SQL Server, PostgreSQL (через Npgsql), MySQL, Sqlite, а також провайдери для Oracle, Cosmos DB та інших джерел. Завдяки цьому розробник має свободу вибору інфраструктури залежно від завдань і середовища виконання.

Ще однією важливою перевагою EF Core є відкритість і розширюваність. Усі компоненти фреймворку мають відкритий код, активно підтримуються спільнотою та постійно оновлюються. Доступ до документації, прикладів і багатьох бібліотек-розширень робить роботу з EF Core зручною та ефективною навіть для початківців.

У контексті дипломного проєкту EF Core відіграє ключову роль як проміжна ланка між прикладною логікою .NET-застосунку та базою даних PostgreSQL. Саме через EF Core реалізується створення, читання, оновлення та видалення об’єктів, ведення міграцій бази, перевірка зв'язків між сутностями, а також взаємодія з транзакціями. Такий підхід дозволяє значно прискорити

розробку і зменшити кількість помилок, що виникають при ручній роботі з SQL.

2.4 NET MAUI

.NET MAUI (Multi-platform App UI) — це сучасна кросплатформенна платформа для розробки інтерфейсних застосунків від компанії Microsoft. Вона є прямим наступником Xamarin.Forms та офіційно увійшла до складу .NET, починаючи з версії 6, а з релізом .NET 8 стала стабільною, зрілою технологією для створення нативних додатків під Windows, Android, iOS та macOS з єдиною базою коду. Основна ідея .NET MAUI полягає в тому, щоб дати змогу розробникам створювати інтерфейсні застосунки для кількох платформ одночасно, використовуючи C# та XAML — без необхідності писати окремий код для кожної платформи (рис 2.4).

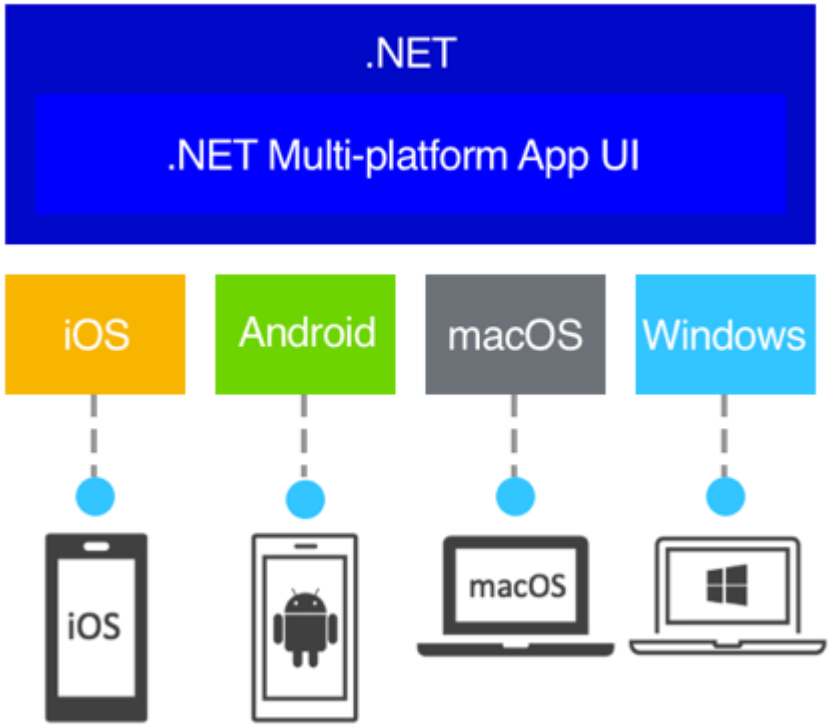


Рис 2.4 – кросплатформенність технології .NET MAUI

.NET MAUI побудовано на базі .NET, з використанням єдиного проєкту для всіх платформ. Це означає, що розробник має єдиний вихідний код, у якому можна розділяти або об’єднувати логіку та елементи інтерфейсу для різних операційних систем. Такий підхід значно спрощує супровід і розвиток застосунку, зменшує дублювання коду та підвищує швидкість розробки.

В основі .NET MAUI лежить архітектура MVU (Model-View-Update), яка дозволяє описувати інтерфейс декларативно, і класична MVVM (Model-View-ViewModel), що підтримується через бібліотеки типу CommunityToolkit.Mvvm. Це дає розробникам можливість вибору підходу, який найбільше відповідає структурі їхнього застосунку та особистим уподобанням.

Однією з переваг MAUI є повна інтеграція з нативними елементами інтерфейсу кожної платформи. Наприклад, додаток на Android використовує реальні Android-компоненти (Button, Entry, ListView), що гарантує нативний вигляд і поведінку, як у застосунках, створених мовами Swift, Kotlin чи Java. Так само на Windows застосунок працює як WinUI 3-застосунок, з повноцінною підтримкою системних можливостей.

.NET MAUI підтримує роботу з широким спектром елементів інтерфейсу: кнопки, текстові поля, списки, навігаційні панелі, вкладки, сповіщення тощо. Також реалізовано роботу з мультимедіа, файловою системою, GPS, камерами, датчиками та іншими функціями пристроїв, що робить MAUI придатним не лише для базових утиліт, а й для повноцінних мобільних і десктопних застосунків.

У .NET MAUI також передбачена система ресурсів і стилів, що дозволяє гнучко налаштовувати вигляд застосунку, включно з темною/світлою темою, кастомними кольорами, шрифтами та адаптивною верстkою. За допомогою XAML-розмітки або C# можна створювати як прості UI-екрани, так і складні, багаторівневі макети з анімаціями та переходами.

З точки зору розгортання, MAUI-проєкти легко пакуються в стандартні формати для відповідних платформ — APK для Android, APP для macOS, MSI або EXE для Windows. Це дає змогу створювати один застосунок, який працює

на кількох пристроях, що особливо зручно у випадках, коли потрібно забезпечити однакову логіку й взаємодію в різних середовищах (наприклад, локальний клієнт резервного копіювання на ПК та смартфоні).

3. РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розроблення бази даних

У процесі розроблення клієнт-серверної системи резервного копіювання було використано реляційну модель зберігання даних, реалізовану за допомогою системи управління базами даних PostgreSQL. Як уже зазначалося в розділі 2, вибір саме цієї СКБД зумовлений її стабільністю, високою продуктивністю, відповідністю стандартам SQL, широкою підтримкою інструментів для розробки, а також відкритою ліцензією. Додатково важливою перевагою є безшовна інтеграція PostgreSQL з платформою .NET через ORM-фреймворк Entity Framework Core.

Архітектура бази даних побудована на основі реляційного підходу, що передбачає зберігання даних у взаємопов’язаних таблицях з чітко визначеними зовнішніми ключами. Це дозволяє гарантувати логічну цілісність, уникати дублювання, забезпечити консистентність і оптимізувати виконання запитів.

Загальна модель складається з шести таблиць: Users, Roles, Backups, Files, FileInfos і Roles. Кожна таблиця виконує окрему логічну функцію:

Users — зберігає облікові дані користувачів, включаючи адресу електронної пошти, пароль (у вигляді гешу), ім’я, прізвище та ідентифікатор ролі.

Roles — довідкова таблиця, що містить список доступних ролей у системі.

Backups — містить дані про кожну створену резервну копію, включаючи час створення, користувача, який ініціював копіювання, та ознаку ручного запуску (IsManual).

Files — відображає список файлів, які були включені до певної копії, разом із відносним шляхом до них на файловій системі.

FileInfos — зберігає метаінформацію про кожен файл: тип, розмір, дату створення та шлях.

Також модель повністю відповідає стандартам і правилам побудови баз даних (рис 3.1).

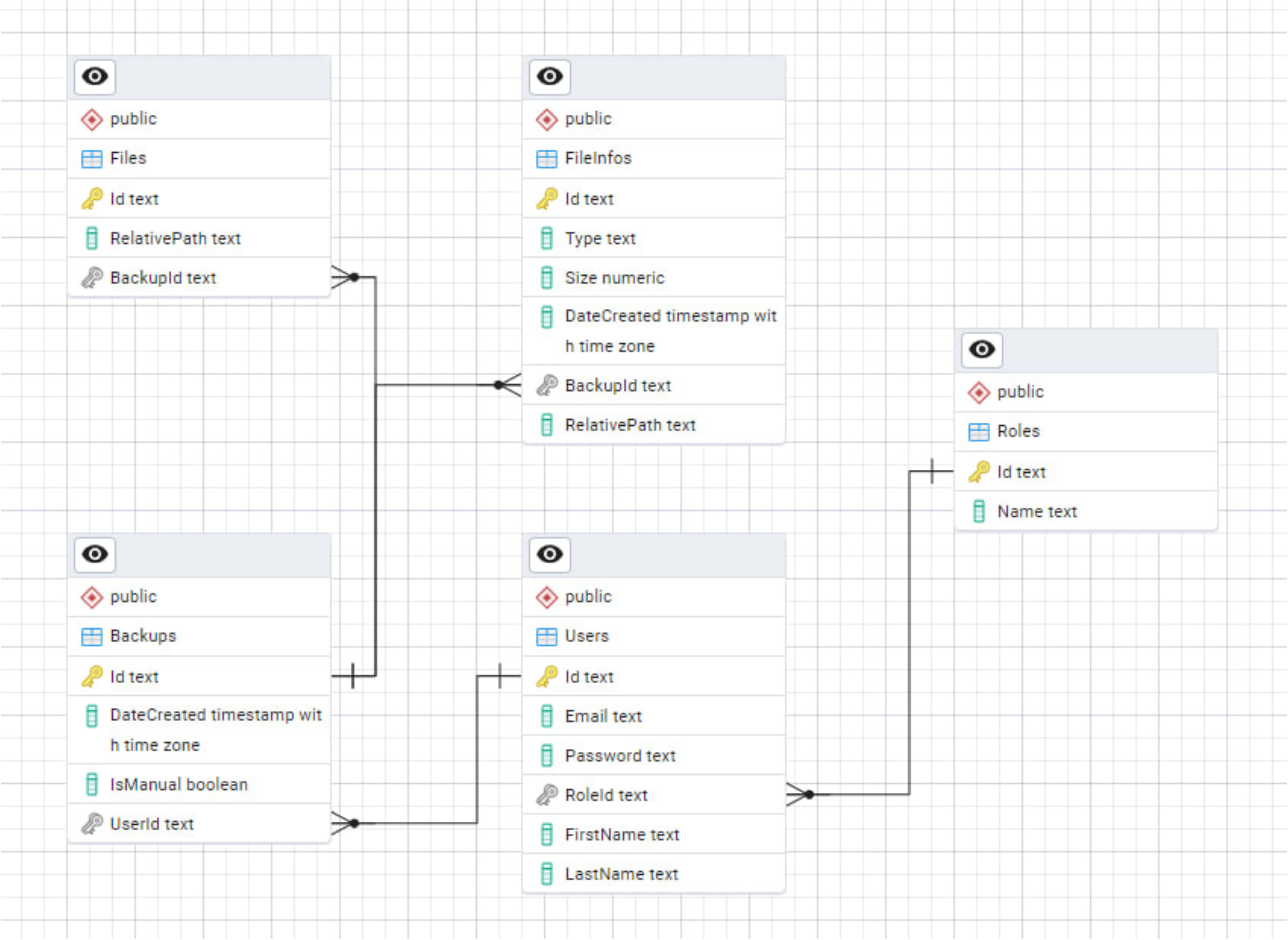


Рис 3.1 – ERM-діаграма бази даних

Для ідентифікації записів у всіх таблицях використано поля Id з типом text, які містять значення формату UUID. Генерація UUID здійснюється на стороні бази даних за допомогою функції gen_random_uuid() з розширення pgcrypto. Це дозволяє забезпечити унікальність ключів без необхідності генерації на рівні застосунку, а також гарантує безпеку і зручність масштабування, зокрема у випадках розподілених систем або паралельної обробки.

Усі поля, що відповідають за зберігання часових міток (DateCreated), мають тип timestamp with time zone. Усі значення зберігаються у форматі UTC, що гарантує однакову інтерпретацію часу незалежно від часової зони, в якій знаходиться користувач або сервер. Дати також генеруються автоматично під

час створення запису, що спрощує логіку застосунку та забезпечує достовірність історії змін.

Особливу увагу приділено захисту персональних даних. У таблиці Users поле Password не зберігає відкритого пароля — замість цього використовується геш пароля, створений за допомогою алгоритму bcrypt. Процес гешування відбувається на стороні окремого мікросервісу, що відповідає за автентифікацію користувачів. Bcrypt забезпечує стійкість до атак перебору завдяки внутрішньому механізму сольового гешування та параметру складності, який можна регулювати залежно від потреб.

Щодо рольової моделі, то вона реалізована через таблицю Roles, яка пов’язана з таблицею Users за допомогою зовнішнього ключа RoleId. У системі передбачено дві ролі:

– User — звичайний користувач із базовим функціоналом і лімітом у 5 копій;

– Premium — користувач із розширеними правами, зокрема більшими обсягами доступного простору, можливістю зберігати до 30 окремих копій.

Ця структура дозволяє легко реалізовувати обмеження та фільтрувати функціональність в інтерфейсі, ґрунтуючись на ролі користувача.

Зберігання файлів у цій системі реалізовано не в базі даних, а на серверній файловій системі. У таблиці Files поле RelativePath містить шлях до файлу відносно кореневої директорії збереження. Такий підхід дозволяє значно зменшити навантаження на базу даних, уникнути використання BLOB-типів, полегшити резервне копіювання самої бази, а також дозволяє масштабувати файлову частину системи незалежно від структури даних. Окремо таблиця FileInfos дозволяє зберігати метадані про файли, не дублюючи фізичні дані, та забезпечує гнучкість у побудові запитів, сортування, статистики та фільтрації.

Кожен запис у таблиці Backups пов’язаний із записами в Files та FileInfos, що дозволяє відновлювати повну структуру копії за будь-який

період, перевіряти її склад, переглядати хронологію резервування та отримувати аналітичну інформацію про обсяг збережених даних.

Завдяки використанню зовнішніх ключів, усі таблиці пов’язані між собою, що гарантує логічну цілісність даних. Така структура є стабільною, розширюваною та безпечною. Вона дозволяє ефективно управляти інформацією про користувачів, резервні копії та їхній вміст, забезпечуючи необхідний рівень захисту, зручності та масштабованості.

3.2 Розроблення серверної частини

Серверна частина клієнт-серверної системи резервного копіювання була створена з використанням ASP.NET Core — сучасного фреймворку для розробки веб- і API-застосунків. Цей вибір обумовлений його відкритою природою, підтримкою кросплатформенності, високою продуктивністю та широкими можливостями для побудови безпечних і масштабованих рішень. У процесі розробки основна увага була зосереджена на модульності, читабельності та підтримуваності коду, що досягалося за рахунок впровадження перевірених архітектурних шаблонів і принципів.

Центральним елементом структури серверного застосунку є застосування патерна «Репозиторій» (Repository pattern). Він відіграє ключову роль у відокремленні логіки доступу до даних від решти частин застосунку. Кожна сутність бази даних (користувачі, бекапи, файли, ролі) має відповідний інтерфейс репозиторію та реалізацію, яка інкапсулює всю логіку взаємодії з базою. Таким чином, контролери та бізнес-логіка працюють не напряму з контекстом Entity Framework Core, а лише з абстракціями, що відповідають за отримання й обробку даних. Це дозволяє спростити модульне тестування, легко замінювати джерело даних (наприклад, для тестів або в майбутньому на іншу СКБД), а також повторно використовувати логіку роботи з даними в різних частинах системи. Репозиторії реалізовують як базові CRUD-операції, так і спеціалізовані методи для конкретних сценаріїв — наприклад, отримання

всіх файлів певної резервної копії, фільтрація за датами, перевірка прав користувача на ресурс тощо. Усі залежності впроваджуються через механізм Dependency Injection, вбудований у ASP.NET Core, що дозволяє централізовано керувати створенням об’єктів, підвищує гнучкість архітектури та знижує зв’язність між компонентами (рис. 3.2, 3.3).

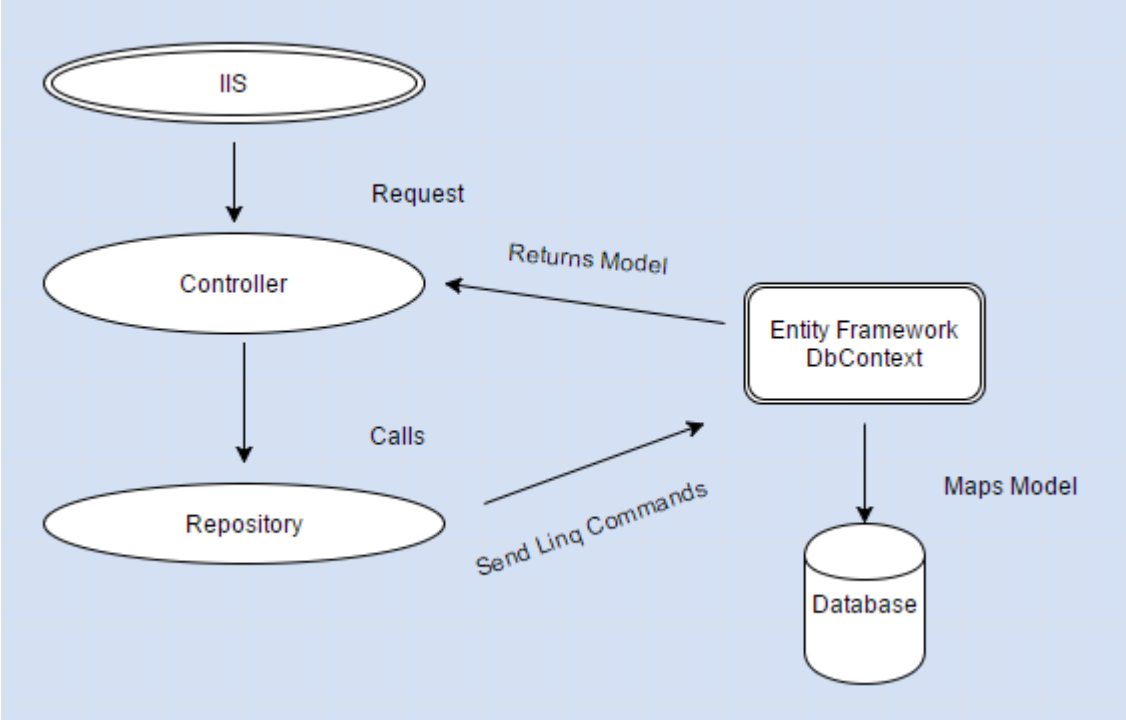


Рис 3.2 – реалізація патерну Repository

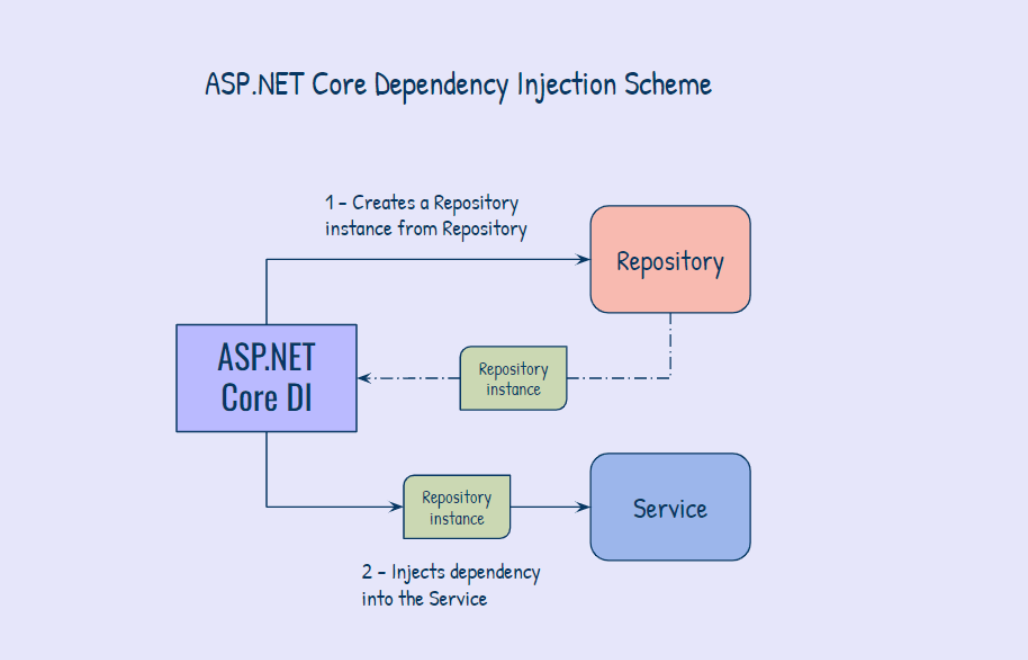


Рис 3.3 – реалізація Dependency Injection

Контролери системи побудовані у вигляді REST API і мають атрибут [Authorize], що забезпечує захист усіх маршрутів від неавторизованого доступу. Для автентифікації та авторизації використовується механізм JWT (JSON Web Token), який дозволяє передавати і перевіряти ідентифікаційні дані користувача у вигляді компактного підписаного токена. Після проходження автентифікації користувач отримує токен, який клієнт зобов'язаний надсилати у кожному подальшому запиті. Сервер, отримуючи токен, виконує його перевірку за допомогою вбудованих механізмів валідації підпису, терміну дії та вмісту. З токена витягується інформація про користувача, його роль та ID, що дозволяє не лише визначити, чи може користувач звертатись до певної кінцевої точки, а й застосовувати рольові обмеження на рівні бізнес-логіки (рис 3.4).

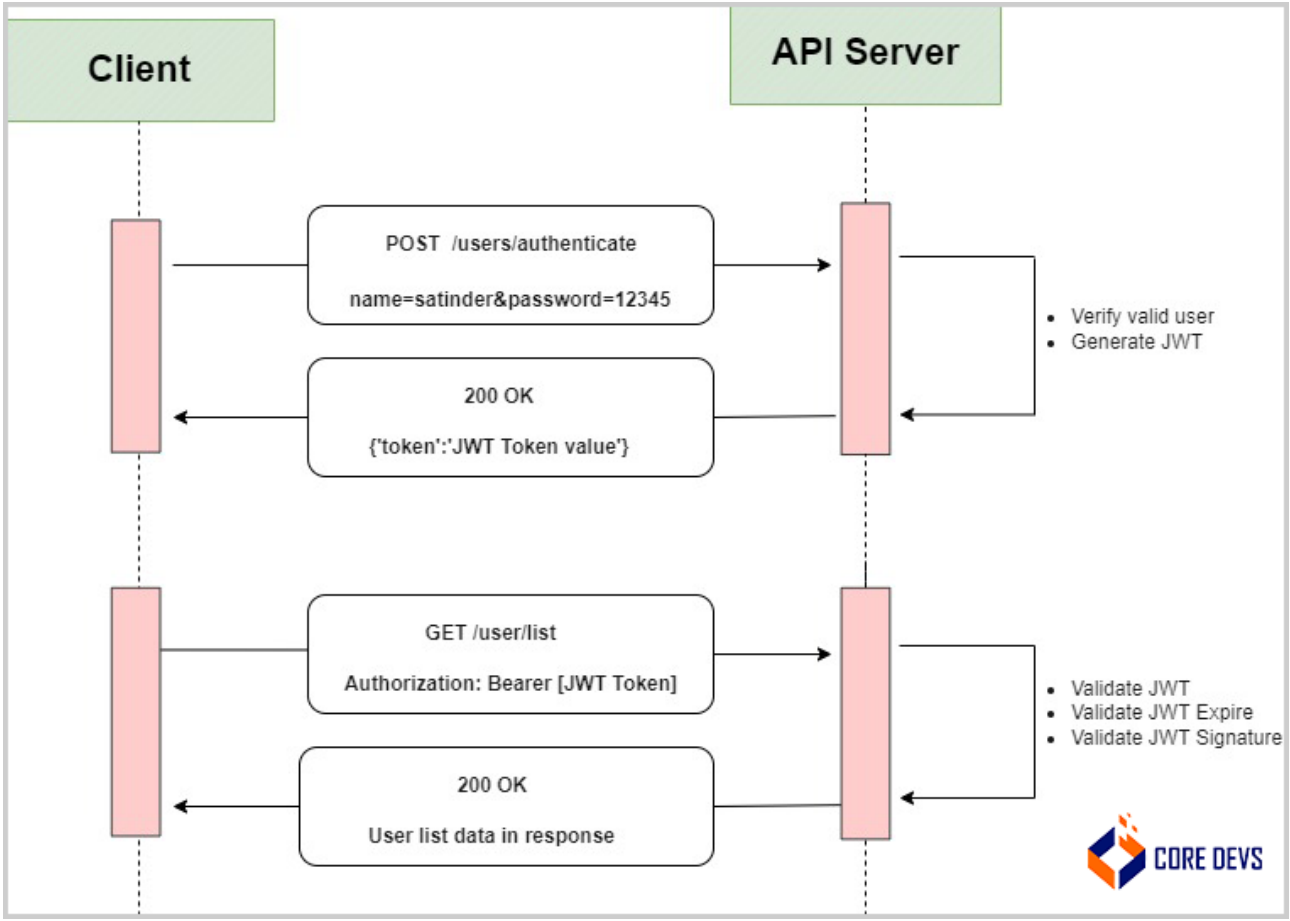


Рис 3.4 – реалізація механізму авторизації JWT

Особливу увагу в проєкті приділено безпеці збережених даних, зокрема персональної інформації та вмісту файлів. Усі паролі перед збереженням до бази проходять хешування за допомогою алгоритму bcrypt, що є сучасним стандартом захисту облікових даних. Хешування виконується не на основному сервері, а на окремому мікросервісі, відповідальному за автентифікацію, що дозволяє зменшити ризики компрометації. Bcrypt автоматично додає унікальну сіль до кожного пароля та виконує обчислення з визначеною складністю, що забезпечує стійкість до атак перебору, навіть у разі витоку гешів.

Що стосується самих файлів, то з міркувань безпеки та ефективності вони не зберігаються в базі даних. Перед передачею на сервер файли шифруються за допомогою алгоритму AES-256 — це відбувається на стороні клієнта або мікросервісу перед завантаженням. Таким чином, сервер приймає вже зашифровані файли і зберігає їх у файловій системі за шляхом, що вказується у полі RelativePath. Цей підхід дозволяє зменшити навантаження на базу, забезпечити швидкий доступ до даних при відновленні та підвищити загальну безпеку — навіть при фізичному доступі до серверного сховища розшифрувати вміст файлу без ключа неможливо (рис 3.5).

```
using var aes = Aes.Create()!;
aes.Key = _encryptionKey;
aes.GenerateIV();

using var fsOut = new FileStream(finalPath, FileMode.Create, FileAccess.Write, FileShare.None);

await fsOut.WriteAsync(aes.IV, 0, aes.IV.Length);

using var crypto = new CryptoStream(
    fsOut,
    aes.CreateEncryptor(aes.Key, aes.IV),
    CryptoStreamMode.Write
);

await file.CopyToAsync(crypto);
await crypto.FlushFinalBlockAsync();
```

Рис 3.5 – програмна реалізація шифрування AES-256

Уся логіка, пов’язана зі створенням резервних копій, прив’язкою файлів, перевіркою гешів, обробкою запитів на оновлення чи видалення, а також облік користувачів і їх ролей, реалізована централізовано на сервері. Клієнтська частина не має прямого доступу до бази даних або файлової системи — вся взаємодія з інформацією відбувається через контролери, які делегують запити відповідним сервісам і репозиторіям. Такий підхід забезпечує чіткий розподіл відповідальностей, дозволяє відслідковувати всі дії користувача, логувати операції, перевіряти права доступу і реагувати на можливі порушення безпеки.

У межах загального підходу до масштабованості, вся серверна логіка побудована з урахуванням можливості розширення — як у плані функціоналу, так і в плані навантаження. Завдяки структурі на базі репозиторіїв та інжекції залежностей нові модулі можна додавати без ризику порушення існуючої логіки. Крім того, централізований доступ до бази через EF Core дозволяє легко реалізовувати нові запити, інтегрувати сторонні сервіси (наприклад, для аналітики або резервного архівування) та контролювати цілісність даних за допомогою транзакцій.

У результаті серверна частина виступає не лише як посередник між клієнтом і базою даних, а як повноцінний центр логіки, відповідальний за автентифікацію, авторизацію, обробку запитів, захист даних і цілісність системи загалом. Така реалізація базується на сучасних підходах до побудови безпечних і масштабованих застосунків та відповідає найкращим практикам архітектури програмного забезпечення.

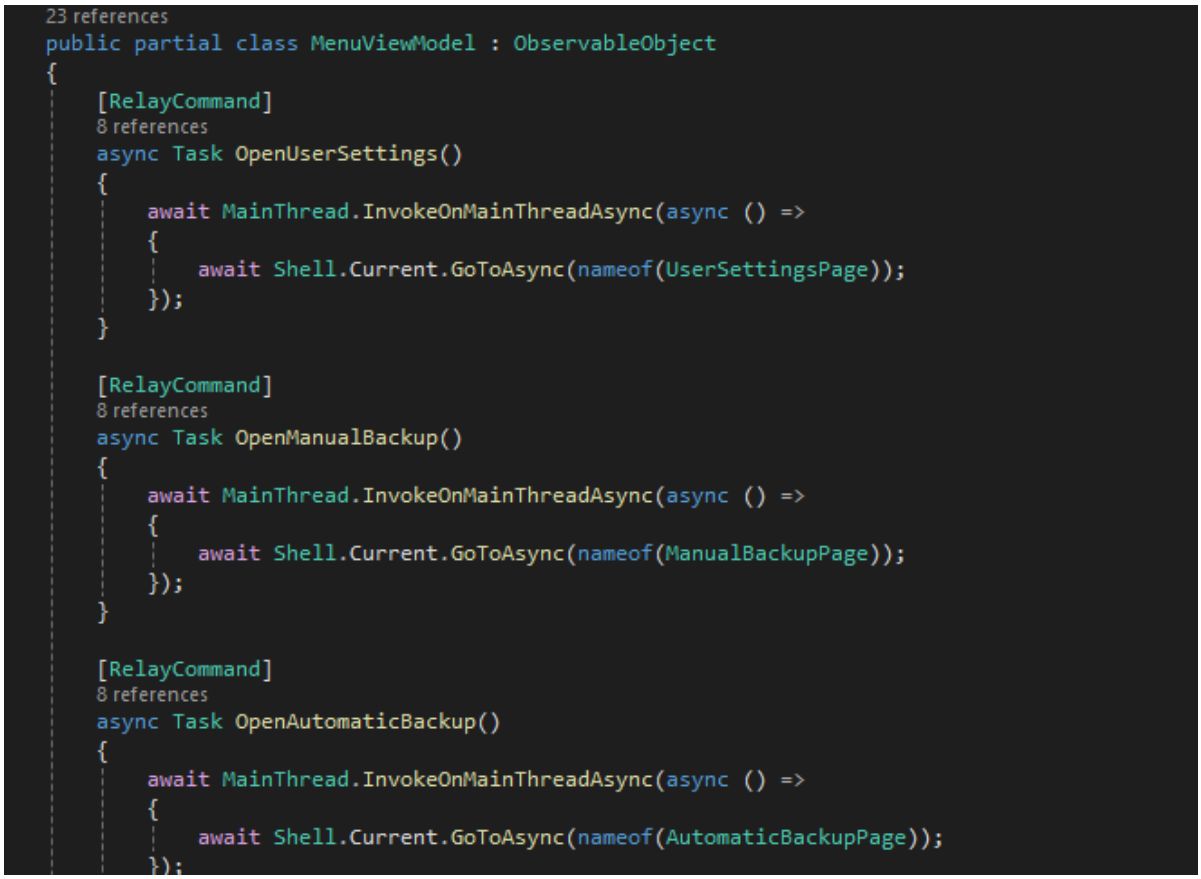
3.3 Розроблення клієнтського застосунку

Клієнтська частина клієнт-серверної системи резервного копіювання розроблена з використанням .NET MAUI — сучасного кросплатформеного фреймворку, що дозволяє створювати застосунки під різні платформи з єдиною кодовою базою. У цьому проєкті клієнт орієнтований виключно на настільні операційні системи — Windows та macOS, що дозволяє враховувати

специфіку взаємодії саме для користувачів персональних комп’ютерів. Завдяки MAUI вдалося реалізувати стабільний, адаптивний інтерфейс із сучасним зовнішнім виглядом, який добре масштабується під різні розміри екранів.

Архітектурною основою застосунку став патерн MVVM (Model-View-ViewModel), який забезпечив повне розділення відповідальностей між шаром UI, логікою взаємодії та моделями даних. Вся бізнес-логіка зосереджена у ViewModel-класах, які не мають жодної залежності від елементів інтерфейсу. Це дозволяє легко масштабувати програму, змінювати інтерфейс без змін у логіці, а також проводити модульне тестування основних функцій. Для кожного екрану реалізовано відповідну ViewModel, яка містить стан, команди, асинхронні дії, а також працює з сервісами (наприклад, API-запити, локальні налаштування, авторизація) (рис. 3.6, 3.7).

Рис. 3.6 – патерн MVVM



```
23 references
public partial class MenuViewModel : ObservableObject
{
    [RelayCommand]
    8 references
    async Task OpenUserSettings()
    {
        await MainThread.InvokeOnMainThreadAsync(async () =>
        {
            await Shell.Current.GoToAsync(nameof(UserSettingsPage));
        });
    }

    [RelayCommand]
    8 references
    async Task OpenManualBackup()
    {
        await MainThread.InvokeOnMainThreadAsync(async () =>
        {
            await Shell.Current.GoToAsync(nameof(ManualBackupPage));
        });
    }

    [RelayCommand]
    8 references
    async Task OpenAutomaticBackup()
    {
        await MainThread.InvokeOnMainThreadAsync(async () =>
        {
            await Shell.Current.GoToAsync(nameof(AutomaticBackupPage));
        });
```

Рис 3.7 – реалізація патерну за допомогою Toolkit

У проєкті активно використовувалася бібліотека CommunityToolkit.Mvvm, яка спрощує реалізацію MVVM за рахунок автоматичної генерації INotifyPropertyChanged, RelayCommand та зменшення кількості шаблонного коду. Наприклад, для реалізації змінної стану достатньо вказати атрибут [ObservableProperty], а для асинхронної команди — [RelayCommand(IncludeCancelCommand = true)], що дозволяє швидко організувати повноцінну реактивну логіку.

Оскільки застосунок є асинхронним і багато операцій (запити до API, шифрування, обчислення, оновлення стану) виконуються у фонових потоках, для оновлення інтерфейсу використовувалися виклики до MainThread.InvokeOnMainThreadAsync. Це дозволило, зокрема, асинхронно оновлювати прогресбар під час завантаження або створення резервної копії, що критично важливо для інформування користувача про поточний стан процесу. Крім того, навігація між сторінками також виконується асинхронно і

потребує доступу до головного потоку, щоб уникнути конфліктів із UI, що в MAUI дозволяє виконувати безпечно саме через MainThread.

Для зберігання локальної інформації застосунок використовує SecureStorage — захищене середовище для зберігання конфіденційних даних, яке надається платформою MAUI. У межах цього проєкту SecureStorage застосовується для збереження токена авторизації, щоб уникнути повторної аутентифікації при кожному запуску, а також для збереження налаштувань автоматичного резервного копіювання (наприклад, інтервал, обсяг файлів, чи активна функція). Завдяки цьому користувач отримує персоналізований досвід і не потребує повторного налаштування при кожному запуску програми.

Комунікація із сервером реалізована через HttpClient з використанням асинхронних запитів. Усі запити супроводжуються заголовком Authorization: Bearer, якщо користувач автентифікований. У випадках, коли сесія недійсна або токен прострочено, застосунок обробляє ці ситуації та виконує перенаправлення на екран входу, видаляючи збережені дані та пропонуючи авторизацію повторно. Для кожного типу запиту реалізовано обробку помилок, інформування користувача через повідомлення, а також відображення спінера або прогресбару.

Інтерфейс програми складається з головного меню, яке організоване за допомогою AppShell. Меню надає користувачу доступ до таких розділів: історія резервних копій, ручне створення бекапу, конфігурація автоматичного бекапу, відновлення файлів, редагування профілю користувача, а також розділ для оформлення або продовження преміум-підписки. Залежно від ролі користувача (звичайний або преміум), відображаються додаткові функції або обмеження, реалізовані через внутрішню логіку ViewModel без дублювання інтерфейсу.

Таким чином, клієнтська частина забезпечує зручний і стабільний спосіб роботи з системою резервного копіювання, поєднуючи сучасні підходи до архітектури (MVVM), безпечне зберігання конфіденційної інформації (SecureStorage), правильне асинхронне оновлення інтерфейсу (MainThread),

ефективну роботу з сервером (HttpClient) і інтуїтивно зрозумілий інтерфейс, який адаптований до вимог користувачів на Windows та macOS (рис 3.8).

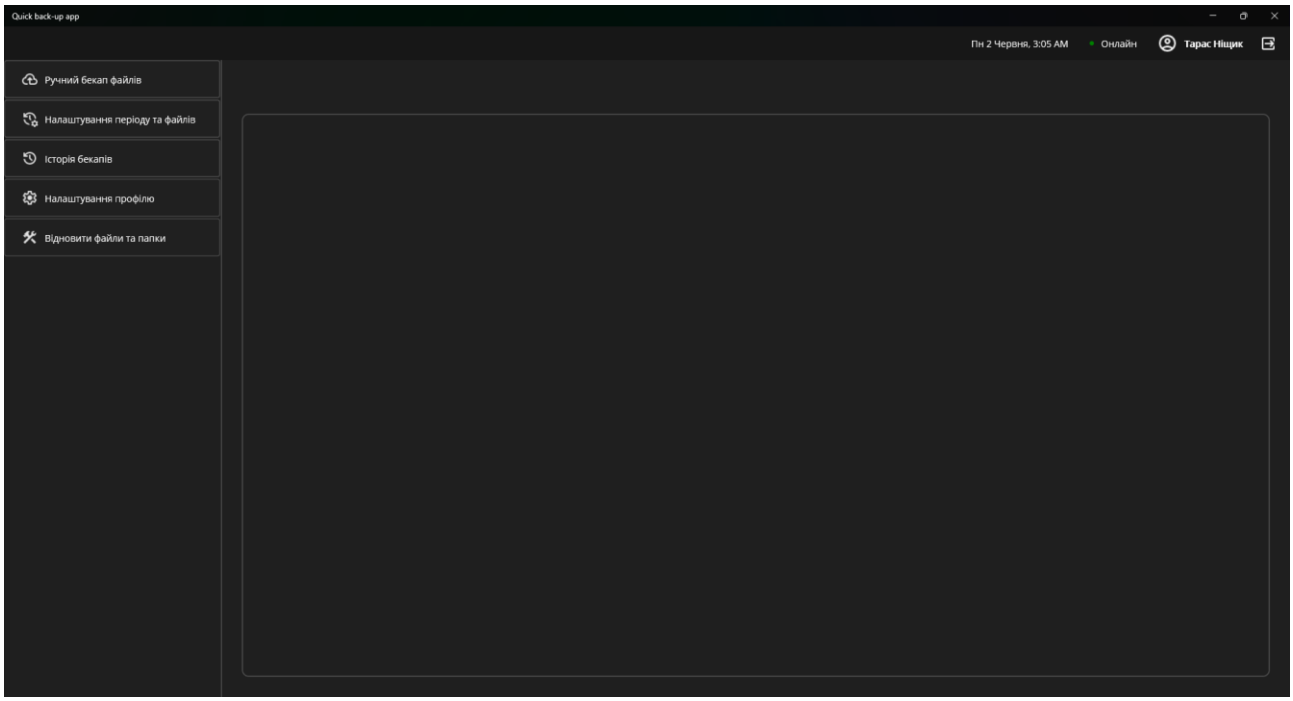


Рис 3.8 – інтерфейс головної сторінки

4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Тестування ручного резервного копіювання

Першим етапом тестування клієнт-серверної системи резервного копіювання стане перевірка функціональності ручного створення резервної копії. У цьому режимі користувач самостійно ініціює процес бекапу, обираючи файли та папки для збереження. Такий підхід дозволяє оцінити базову працездатність системи: правильність формування запиту, шифрування даних, передавання на сервер, збереження метаданих у базі даних та наявність зашифрованих файлів у файловому сховищі. У процесі тестування буде використано набір різних типів файлів і вкладених папок, що дозволить перевірити, як система обробляє структуру каталогів, імена файлів, великі обсяги даних та взаємодію між клієнтом і сервером при різному навантаженні.

Спочатку перейдемо за допомогою меню на відповідний компонент, а саме: «Ручний бекап файлів», та оберемо папку та декілька файлів (рис. 4.1, 4.2).

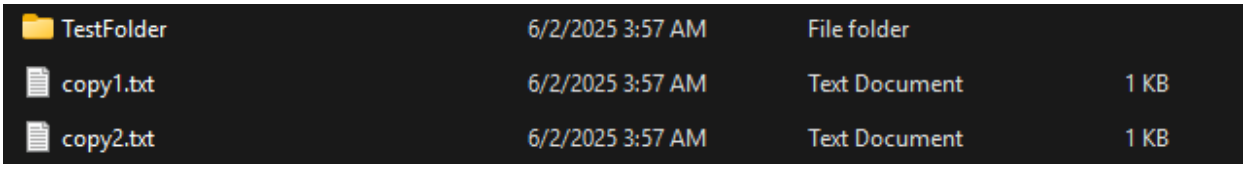


Рис 4.1 – обранні системні файли та папки

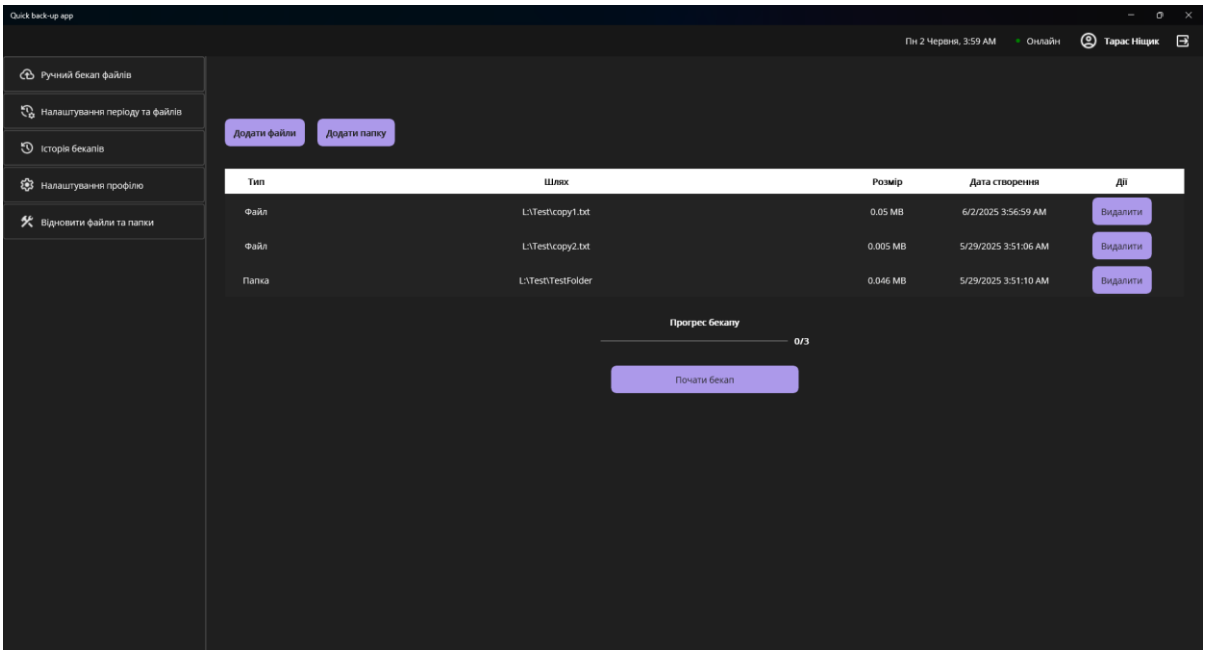


Рис 4.2 – додані файли та папки у середовище

Наступним кроком це є саме резервне копіювання, запустимо його та подивимось чи появилась копія в історії (рис. 4.3, 4.4).

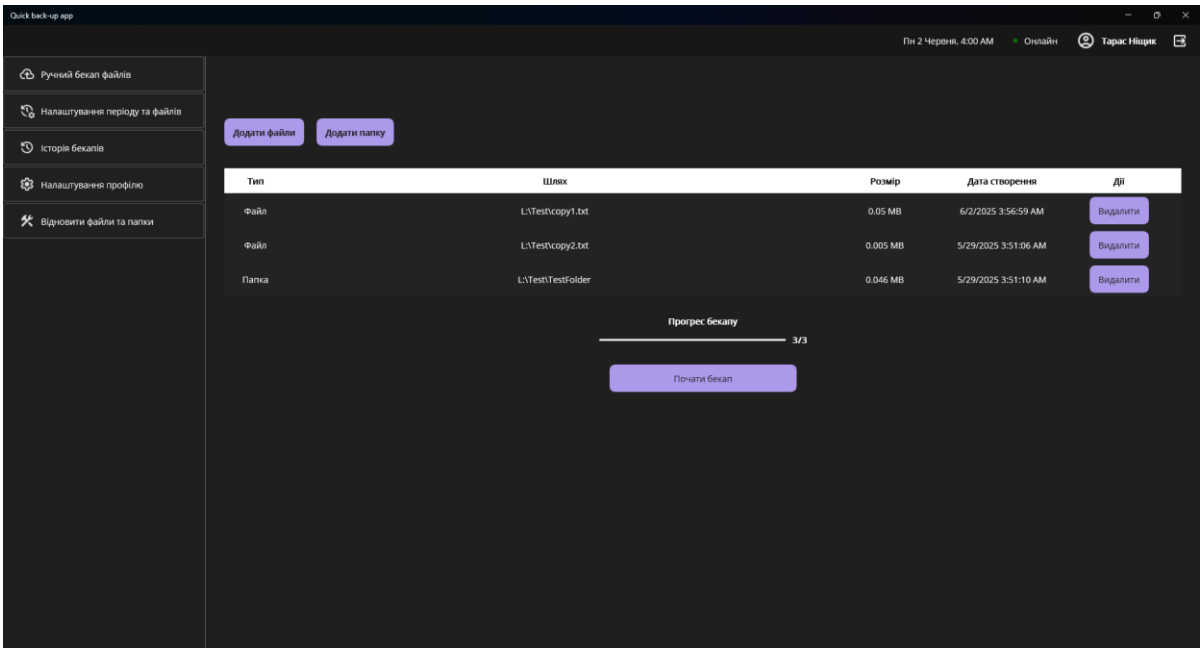


Рис 4.3 – завершення процесу резервного копіювання

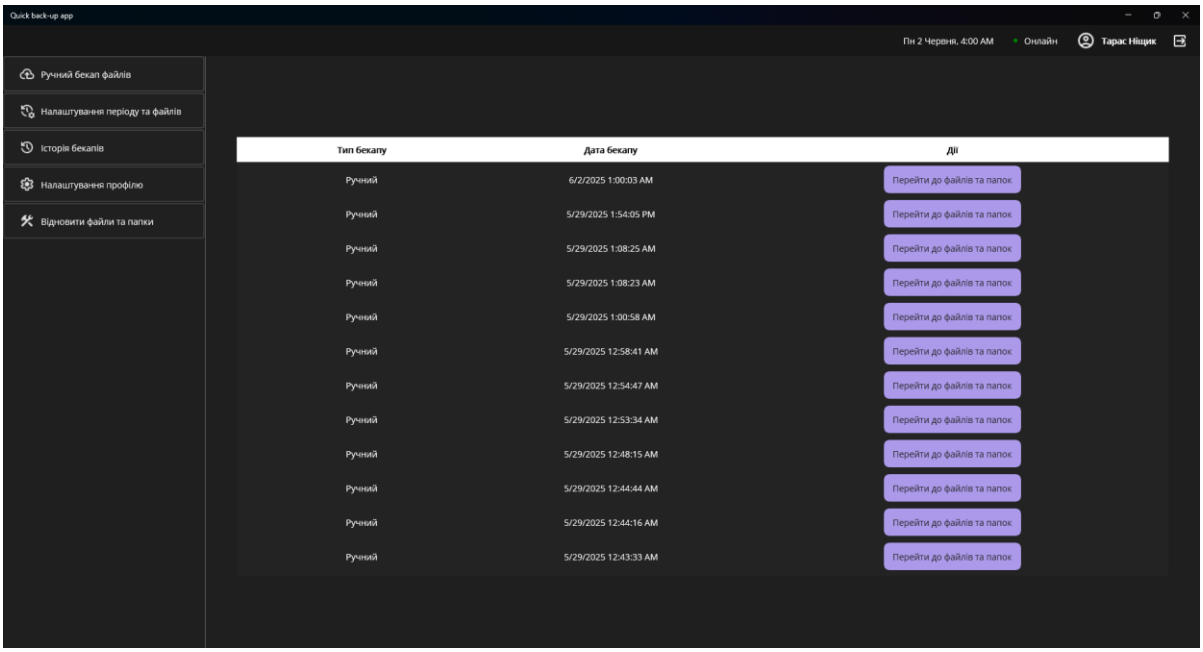


Рис 4.4 – історія копіювань поточного користувача

Бачимо, що бажана резервна копія дійсно є в історії, що доказує роботу цього сервісу.

Заради перевірки цілісності файлу, а також його шифрування перейдемо до розташування файлів серверу та подивимось його вміст (рис. 4.5, 4.6).

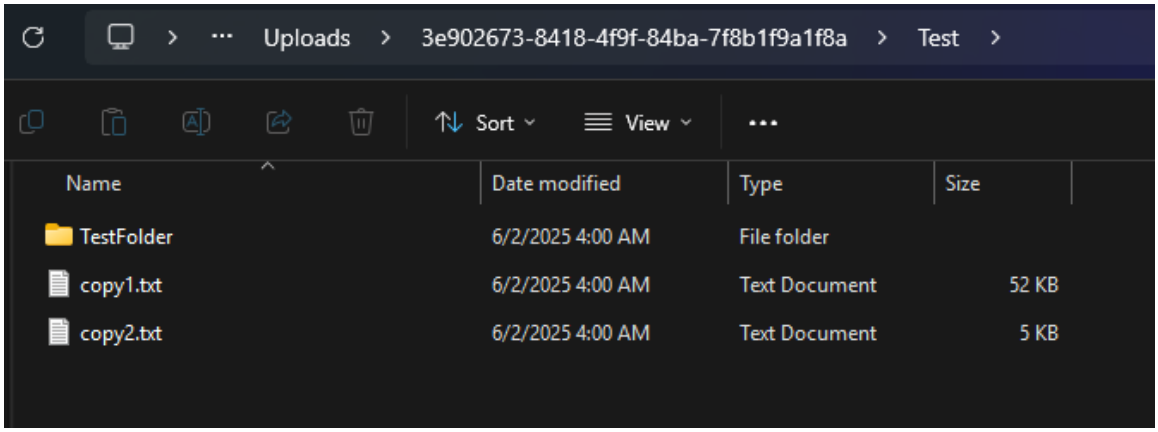


Рис 4.5 – вміст серверної частини

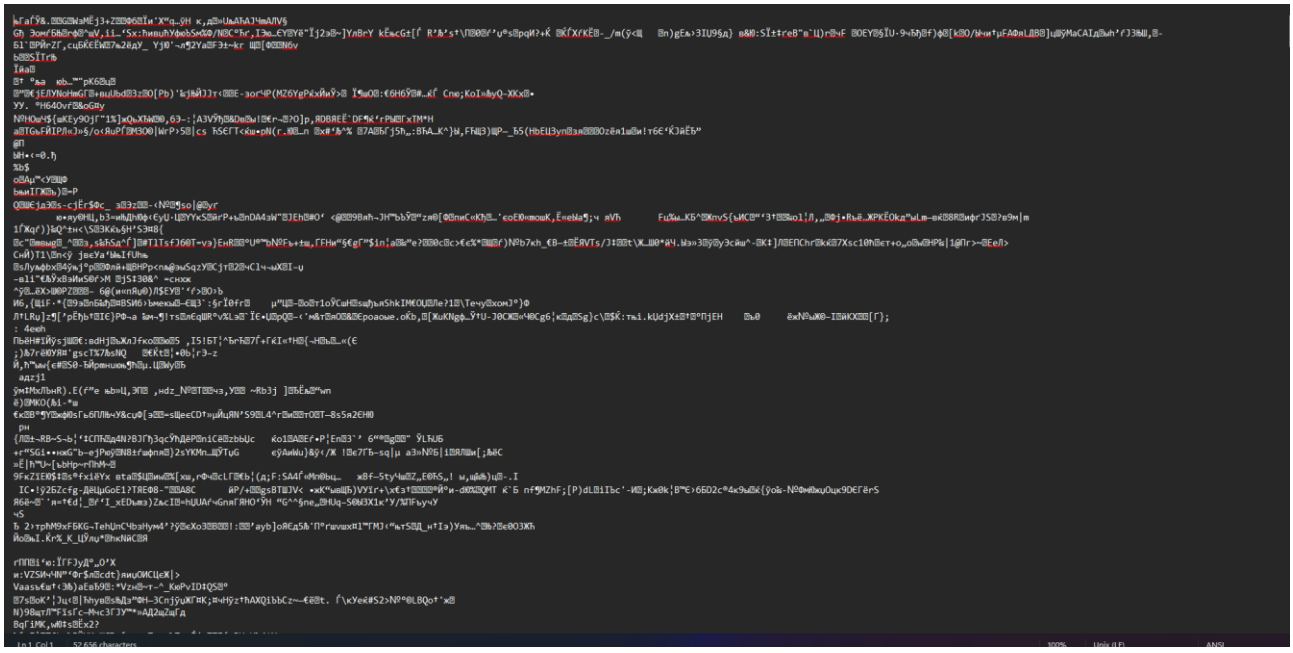


Рис 4.6 – зашифрований вміст текстового файлу

Отже, з цього приводу можна точно сказати, що сервіс не лише працює, а й повністю правильно виконує свої функції.

4.2 Тестування відновлення файлів та папок

З минулого тесту у нас залишилася збережена резервна копія наших файлів. Тепер потрібно протестувати їх відновлення у разі необхідності користувачем. Перейдемо у меню, оберемо відповідну дату резервної копії, оберемо папку, куди завантажити вміст, та запустимо процес (рис 4.7, 4.8).

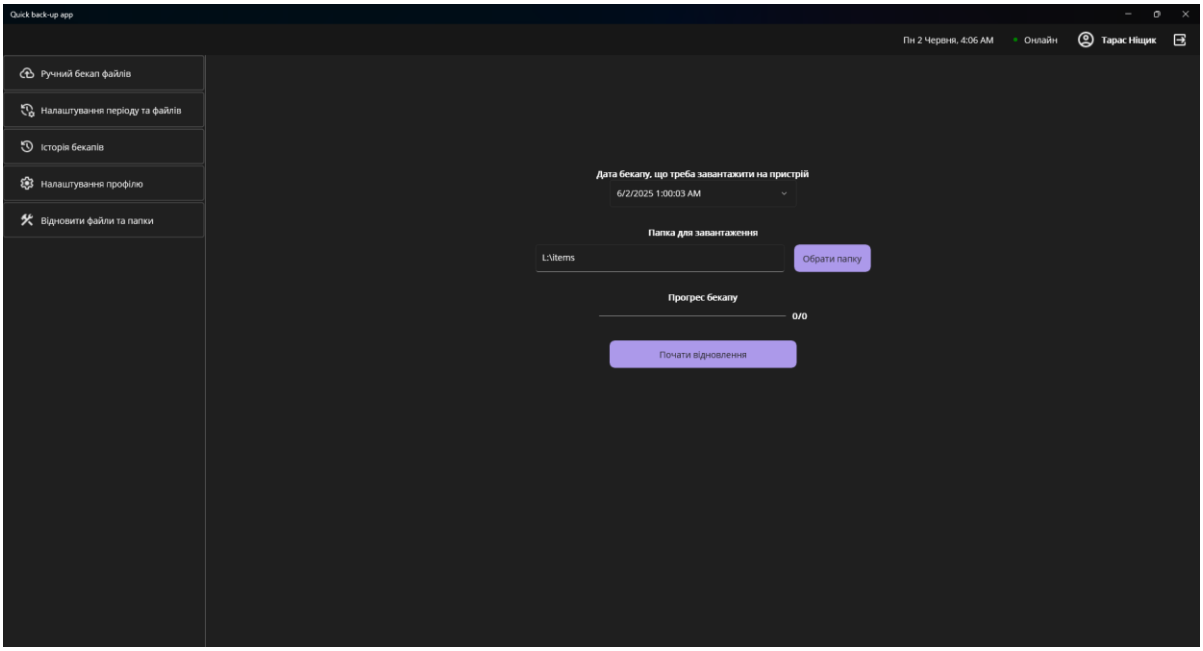


Рис 4.7 – заповнення усіх полей для початку відновлення

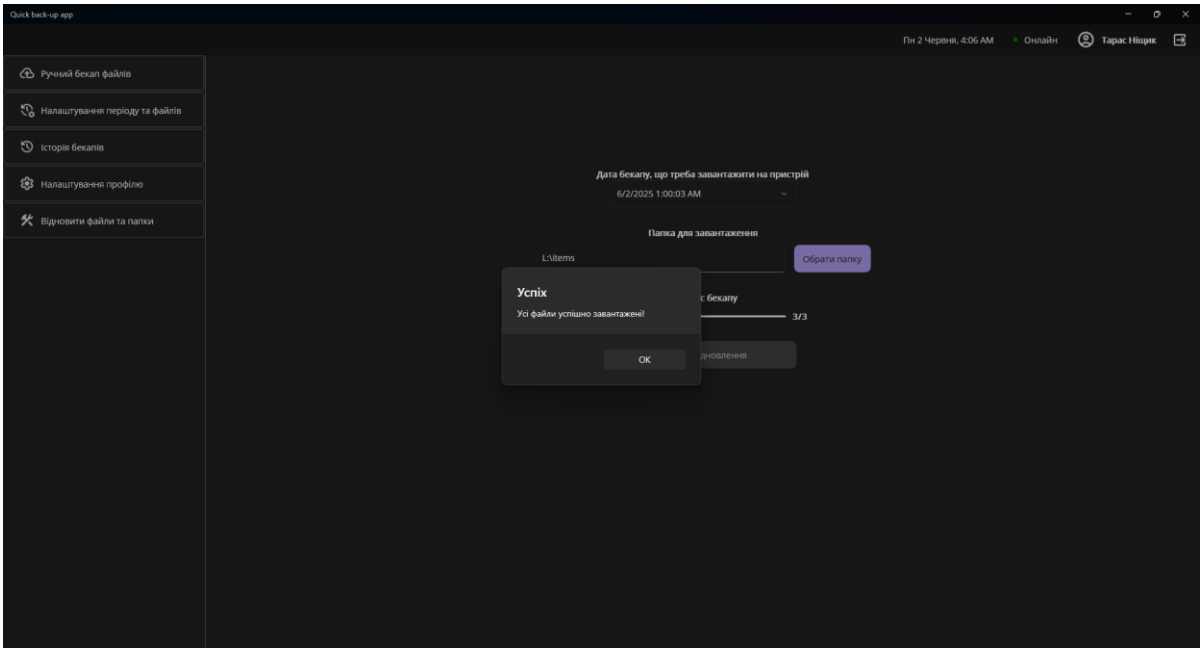


Рис 4.8 – повідомлення про успішне відновлення усіх файлів та папок

Із застосунку прийшло сповіщення, що все відновлено, тепер потрібно це перевірити також і на пристрої користувача (рис. 4.9, 4.10).

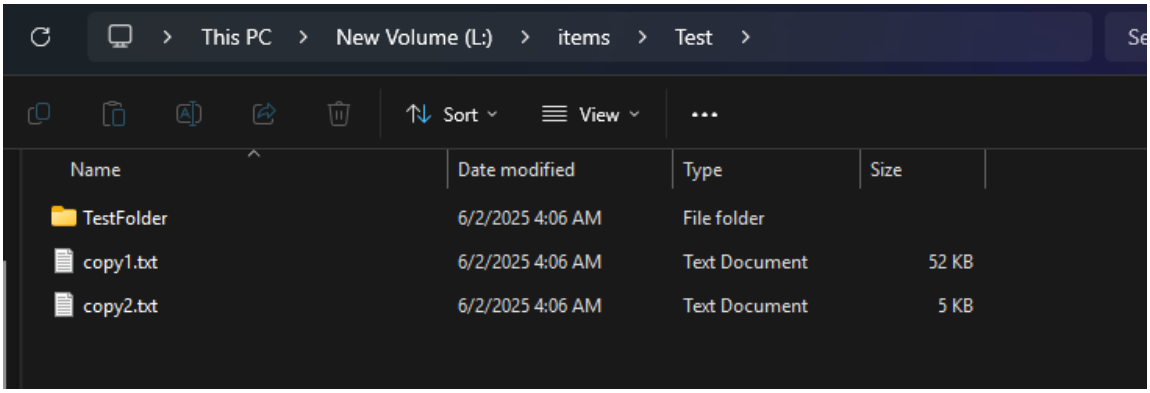


Рис 4.9 – результат відновлення файлів та папок

Рис 4.10 – розшифрований вміст файлу

Отже, з рисунків бачимо, що відновлення файлів та папок дійсно пройшло успішно, вміст не змінився, він зберігся, що свідчить про правильність роботи алгоритму шифрування AES-256, а також загально, що алгоритм відновлення працює правильно і без похибок.

4.3 Тестування автоматичного резервного копіювання

Для тестування автоматизованого резервного копіювання нам потрібно налаштувати це у відповідному пункті меню «Налаштування періоду та файлів».

Після чого кожного дня у цей період відбуватиметься процес копіювання заданих файлів та папок на сервер (рис 4.11, 4.12).

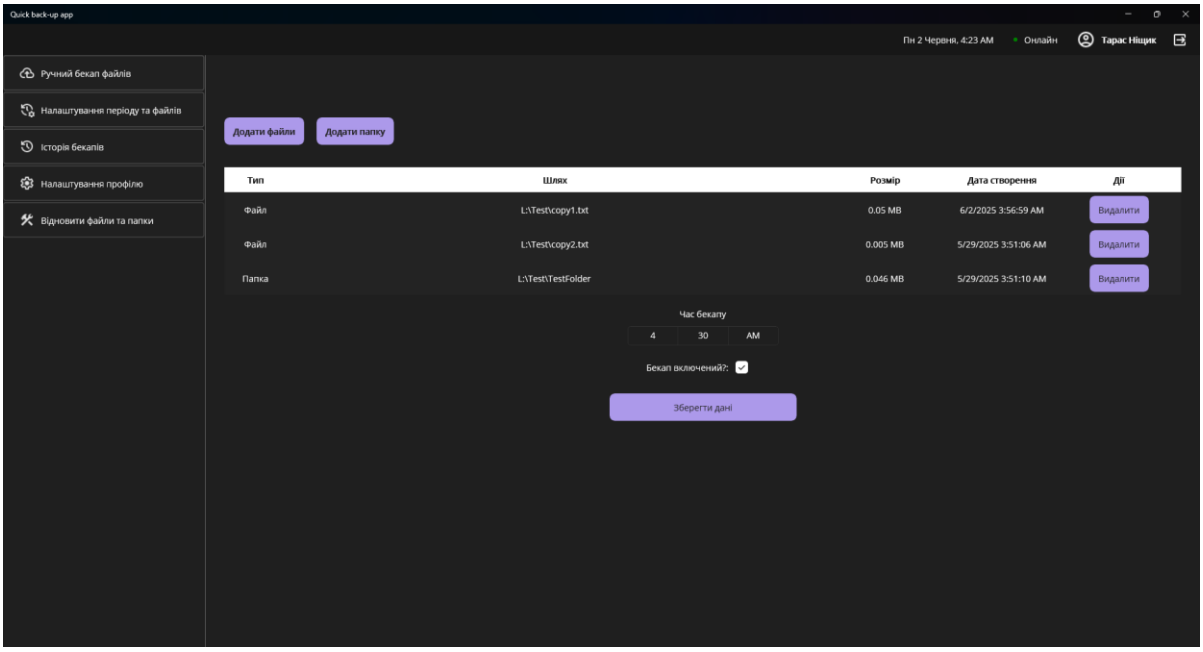


Рис 4.11 – налаштування автоматичного резервного копіювання

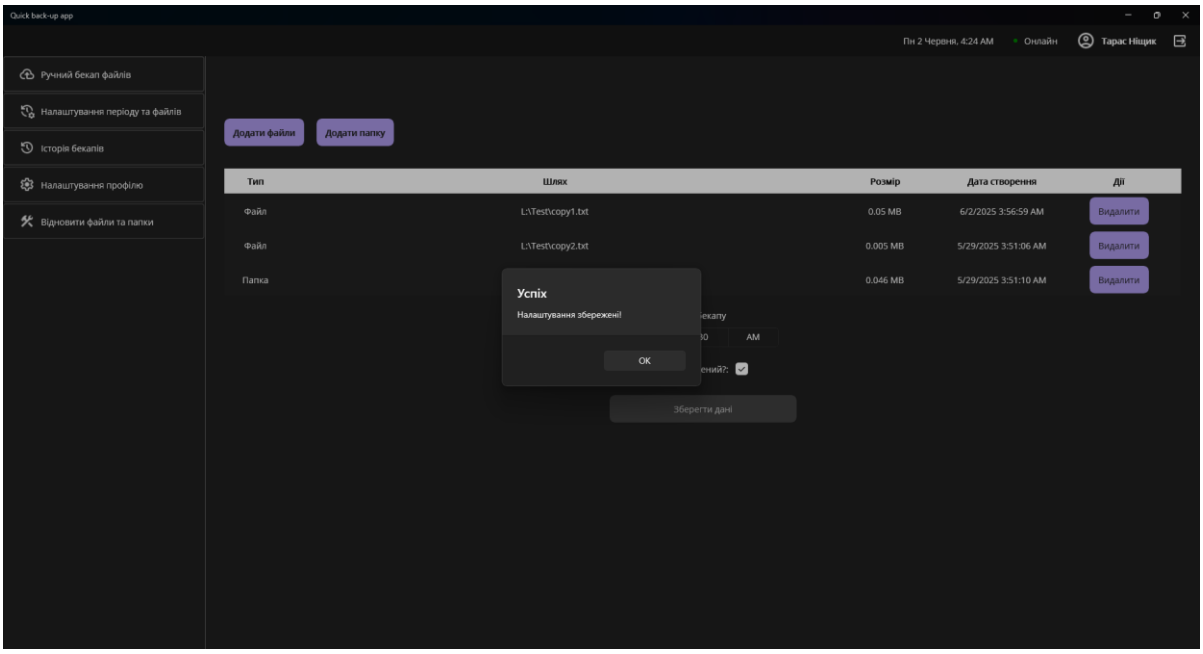


Рис 4.12 – сповіщення про успішне налаштування цього процесу

Коли здійсниться автоматичне копіювання, в історії має з’явитися новий запис, з якого ми можемо побачити не лише саму дату створення копії, але й її вміст – файли та папки (рис. 4.13, 4.14).

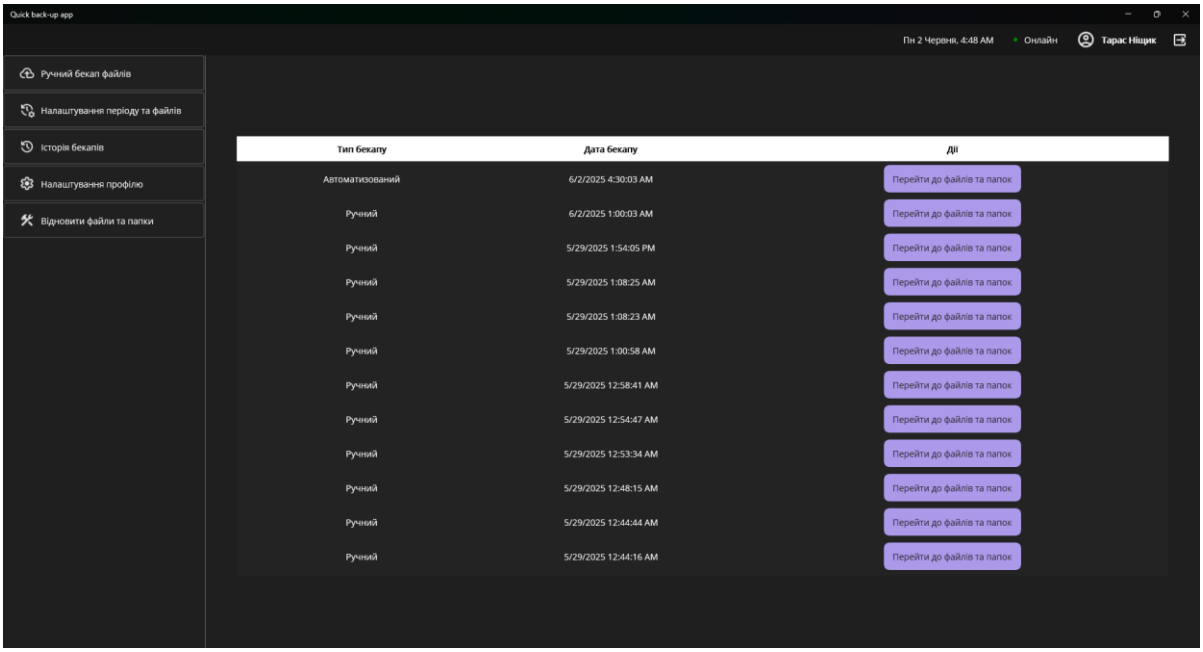


Рис 4.13 – історія резервних копіювань користувача

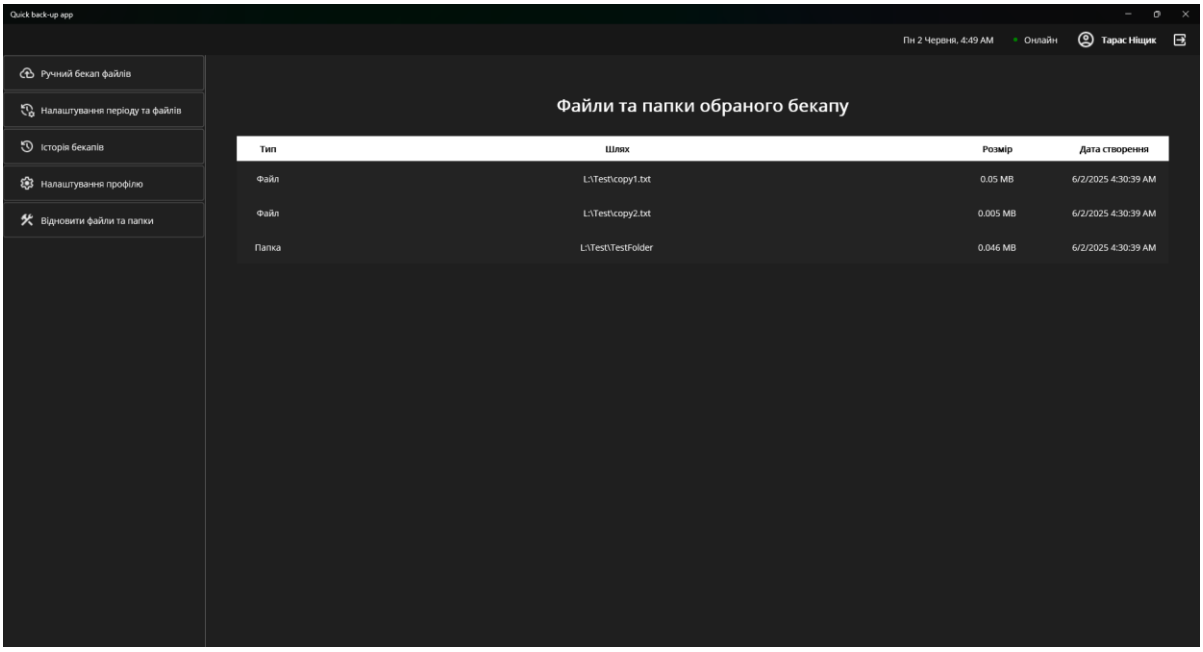


Рис 4.14 – файл та папки, що були відправлені на сервер

Отже, можна стверджувати про те, що автоматичне резервне копіювання також працює.

4.4 Тестування зміни налаштувань профілю

Для цього тесту нам потрібно перейти у пункт меню «Налаштування профілю», змінити параметри профілю та побачити зміни у інфобарі зверху (рис. 4.15, 4.16, 4.17).

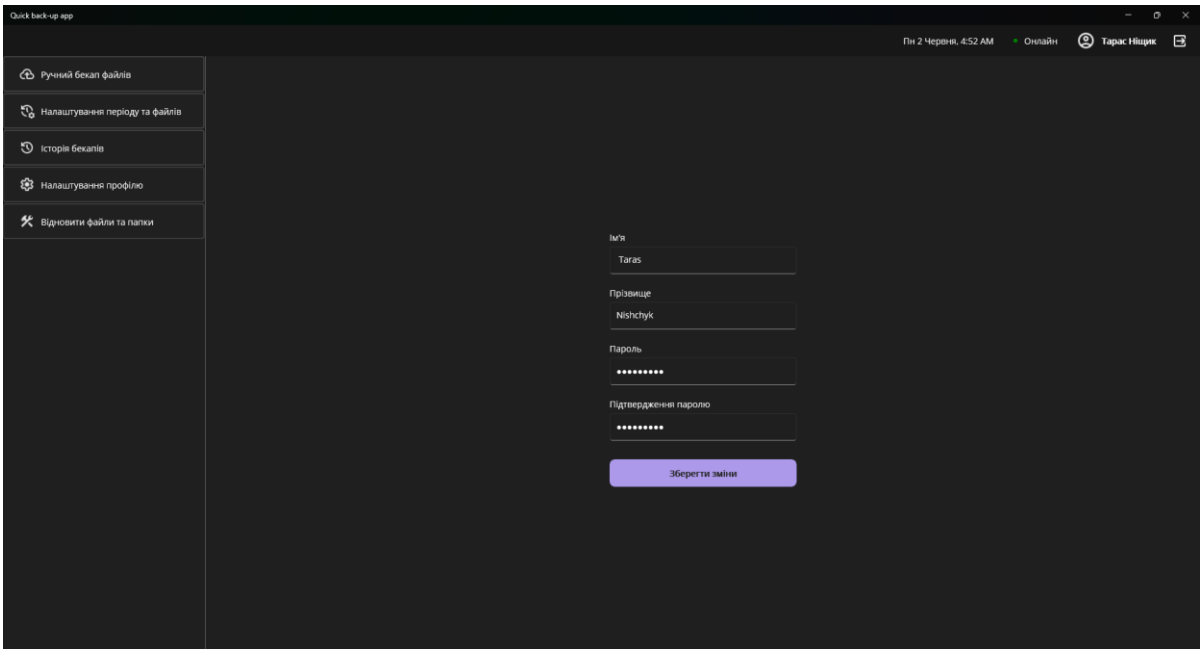


Рис 4.15 – заповнені дані для зміни профілю

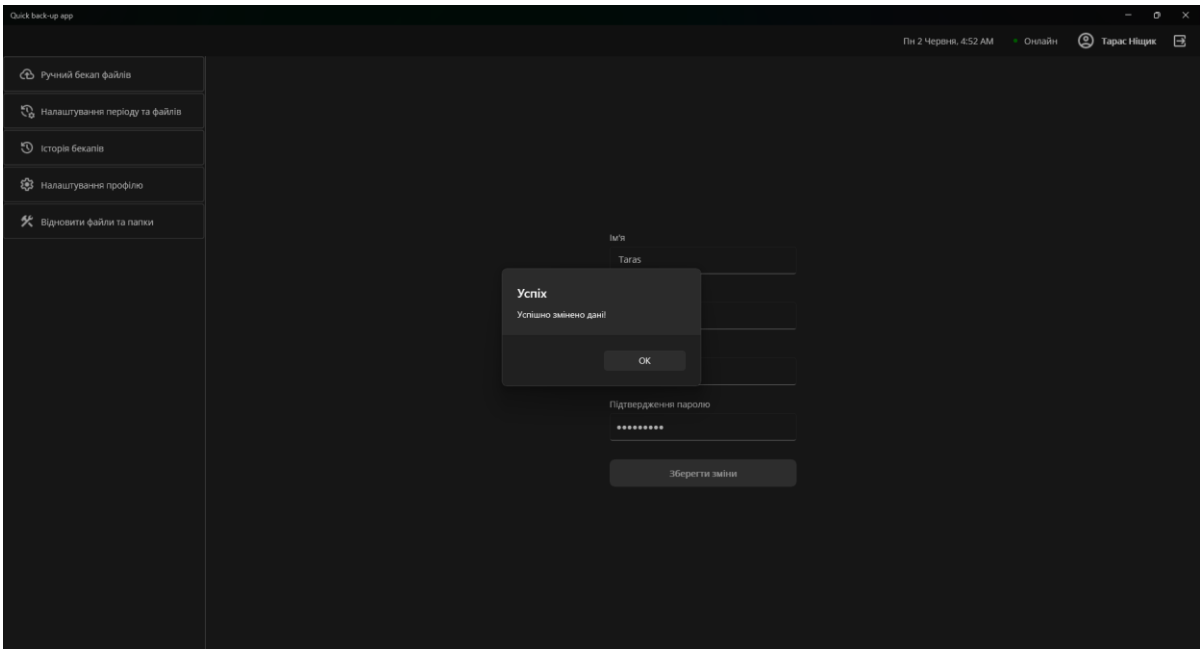


Рис 4.16 – сповіщення про успішне збереження даних

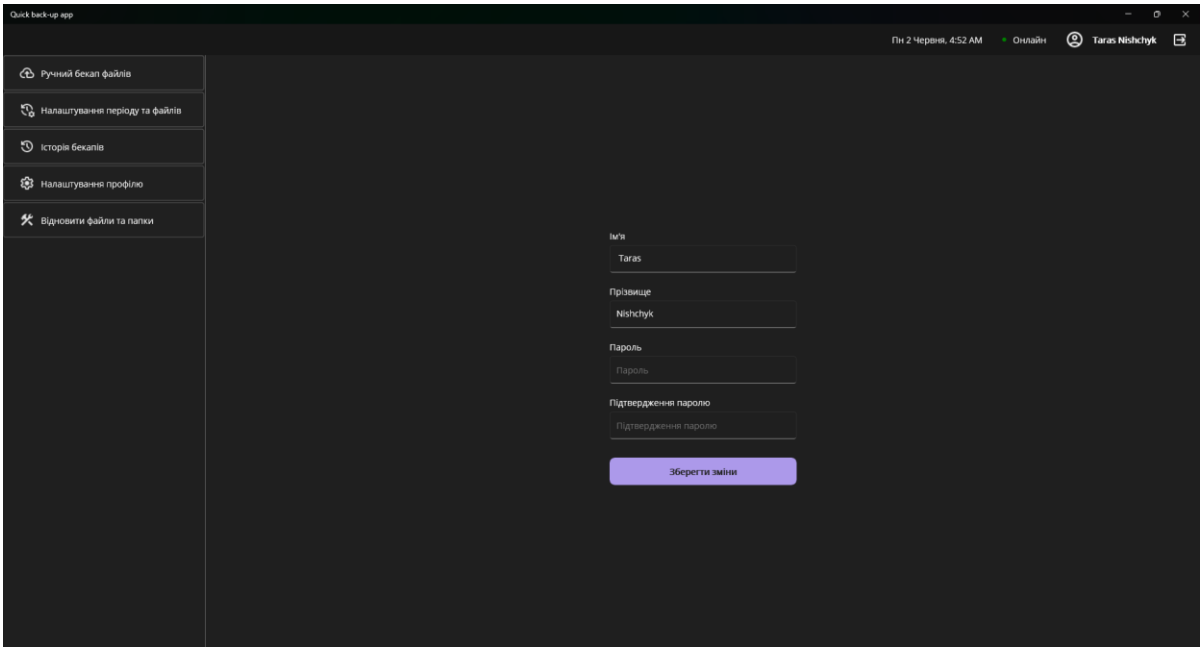


Рис 4.17 – оновлення даних після сповіщення

Отже, можна стверджувати про те, що й ця функція застосунку також працює правильно.

ВИСНОВКИ

У ході виконання дипломного проєкту було проаналізовано сучасні системи резервного копіювання, зокрема такі як Veeam, Duplicati та Microsoft OneDrive. На основі цього аналізу були сформульовані основні вимоги до створюваної системи, з урахуванням сильних сторін існуючих рішень та недоліків, яких варто було уникнути. Результатом роботи стала розробка власної клієнт-серверної системи резервного копіювання, орієнтованої на збереження зашифрованих файлів, захист персональних даних користувача та забезпечення зручного функціоналу як для базових, так і для преміум-користувачів.

Під час розроблення було використано сучасні технології: серверна частина реалізована на базі ASP.NET Core з використанням патерна репозиторій, EF Core, автентифікації через JWT-токени, гешування паролів за алгоритмом bcrypt, а також шифрування файлів за допомогою AES-256 на стороні мікросервісу. Клієнтська частина створена на платформі .NET MAUI з використанням архітектури MVVM, CommunityToolkit, навігації через AppShell, збереження чутливих даних у SecureStorage та асинхронною взаємодією із сервером через HttpClient. Було реалізовано всі необхідні функції: ручне резервне копіювання, автоматичне бекапування, історія збережених копій, відновлення файлів, управління профілем користувача та інтеграція ролей доступу.

На етапі тестування були перевірені основні сценарії використання системи — створення та збереження резервних копій, шифрування і відновлення файлів, робота з налаштуваннями та правами доступу, стабільність роботи при різному навантаженні. Результати тестування підтвердили правильність реалованої логіки, безпеку даних та зручність користування застосунком.

Отже, реалізована система може бути визнана ефективним рішенням для резервного копіювання особистих файлів, що поєднує простоту використання,

сучасну архітектуру, високий рівень безпеки та можливість масштабування. Це робить її придатною як для звичайних користувачів, так і для невеликих організацій, яким важливо зберігати дані в захищеному вигляді з можливістю їх швидкого відновлення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Alvaka Networks. (2025). Exploring Encrypted Backup Services for Robust Data Security: https://www.alvaka.net/exploring-encrypted-backup-services-for-robust-data-security/
2. Bacula Systems. (2025). Backup Encryption 101: Guidelines & Best Practices: https://www.baculasystems.com/blog/backup-encryption-101/
3. CrashPlan. (2024). 3 Reasons to Encrypt Business Data Before Uploading to the Cloud: https://www.crashplan.com/blog/three-reasons-to-encrypt-your-business-data-before-uploading-to-the-cloud/
4. Proton. (2023). What is the Best Encryption for Cloud Storage: https://proton.me/blog/encryption-for-cloud-storage
5. Dropbox. (2025). Encrypted Cloud Storage Guide: How to Secure Data: https://www.dropbox.com/resources/encrypted-cloud-storage
6. Google Cloud. (2025). Data Encryption Options | Cloud Storage: https://cloud.google.com/storage/docs/encryption
7. Red Hat. (2025). Backup and Restore | Red Hat Advanced Cluster Security for Kubernetes: https://docs.redhat.com/en/documentation/red_hat_advanced_cluster_security_for_kubernetes/3.71/html-single/backup_and_restore/index
8. Red Hat. (2025). Key Considerations for Ensuring Data Security: https://learn.redhat.com/t5/General/Key-considerations-for-ensuring-data-security/td-p/43564
9. Red Hat. (2025). When Backups Fail: A Cautionary Sysadmin Tale: https://www.redhat.com/en/blog/backups-cautionary-tale
10. Code Maze. (2024). How to Secure Passwords with BCrypt.NET: https://code-maze.com/dotnet-secure-passwords-bcrypt/
11. Claudio Bernasconi. (2023). How to Hash Passwords with BCrypt in C#: https://claudiobernasconi.ch/blog/how-to-hash-passwords-with-bcrypt-in-csharp/

12. Jason Watmore. (2022). .NET 6.0 - Hash and Verify Passwords with BCrypt: https://jasonwatmore.com/post/2022/01/16/net-6-hash-and-verify-passwords-with-bcrypt
13. CodeBob75. (2023). Repository Pattern C# Ultimate Guide: Entity Framework Core, Clean Architecture, DTOs, Dependency Injection: https://medium.com/@codebob75/repository-pattern-c-ultimate-guide-entity-framework-core-clean-architecture-dtos-dependency-6a8d8b444dcb
14. Code with Mukesh. (2020). Repository Pattern in ASP.NET Core - Ultimate Guide: https://codewithmukesh.com/blog/repository-pattern-in-aspnet-core/
15. Stack Overflow. (2020). Implementing the Repository Pattern Correctly with EF Core: https://stackoverflow.com/questions/64957036/implementing-the-repository-pattern-correctly-with-ef-core
16. Microsoft Learn. (2023). Dependency Injection in .NET MAUI: https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/dependency-injection?view=net-maui-9.0

ДОДАТОК А

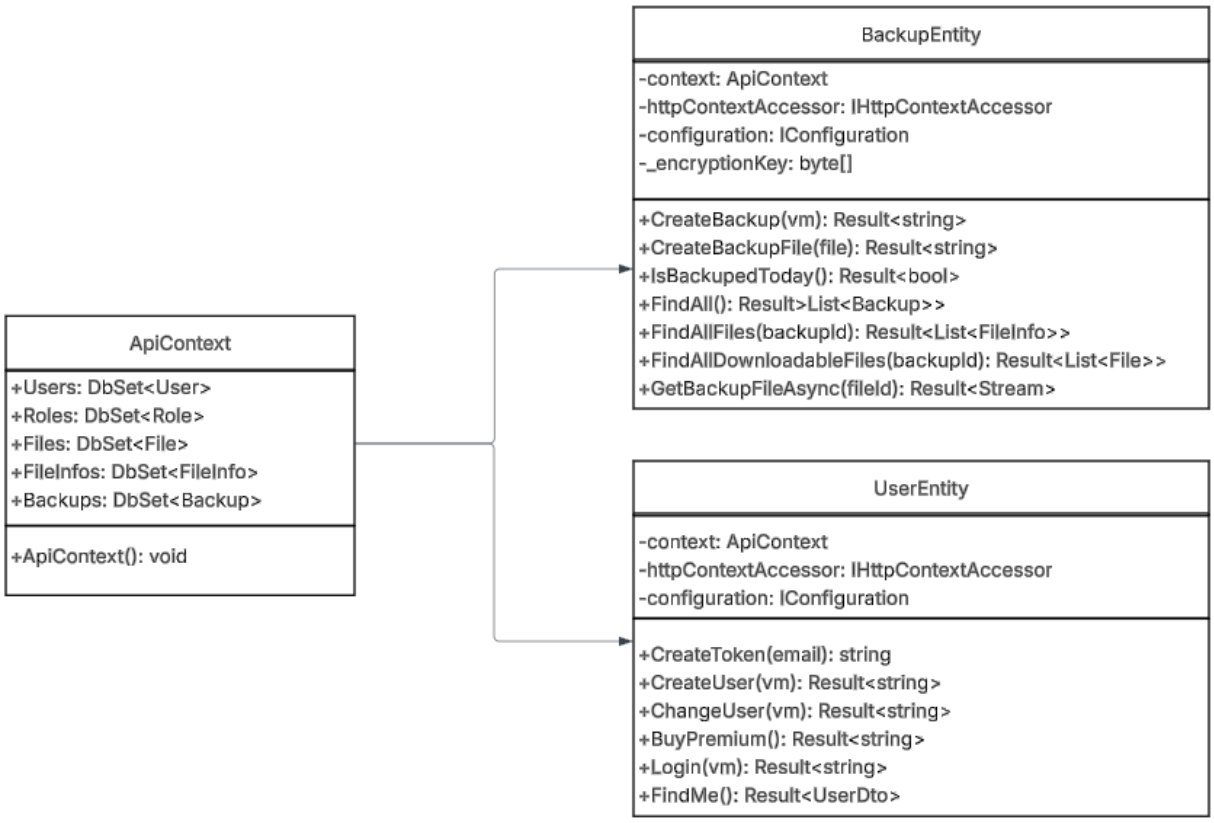
Клієнт-серверна резервного копіювання зашифрованих файлів

Схема взаємодії модулів програми

ІАЛЦ.045440.005 Д1

Аркушів: 1

Київ – 2025



					ІАЛЦ.045440.005 Д1
Змін.	Арк.	№ докум.	Підпис	Дата	
Розробив		Ніщик Т. А.			Клієнт-серверна система резервного копіювання зашифрованих файлів
Перевірив		Сулема О. К.			
Консульт.					**Схема взаємодії модулів програми**
Н. контроль		Клятченко Я. М.			
Зав. каф.		Романкевич В.О.			

Літ.	Аркуш	Аркушів
	1	1

КПІ
Ім. Ігоря Сікорського,
ФПМ КВ-11

ДОДАТОК Б

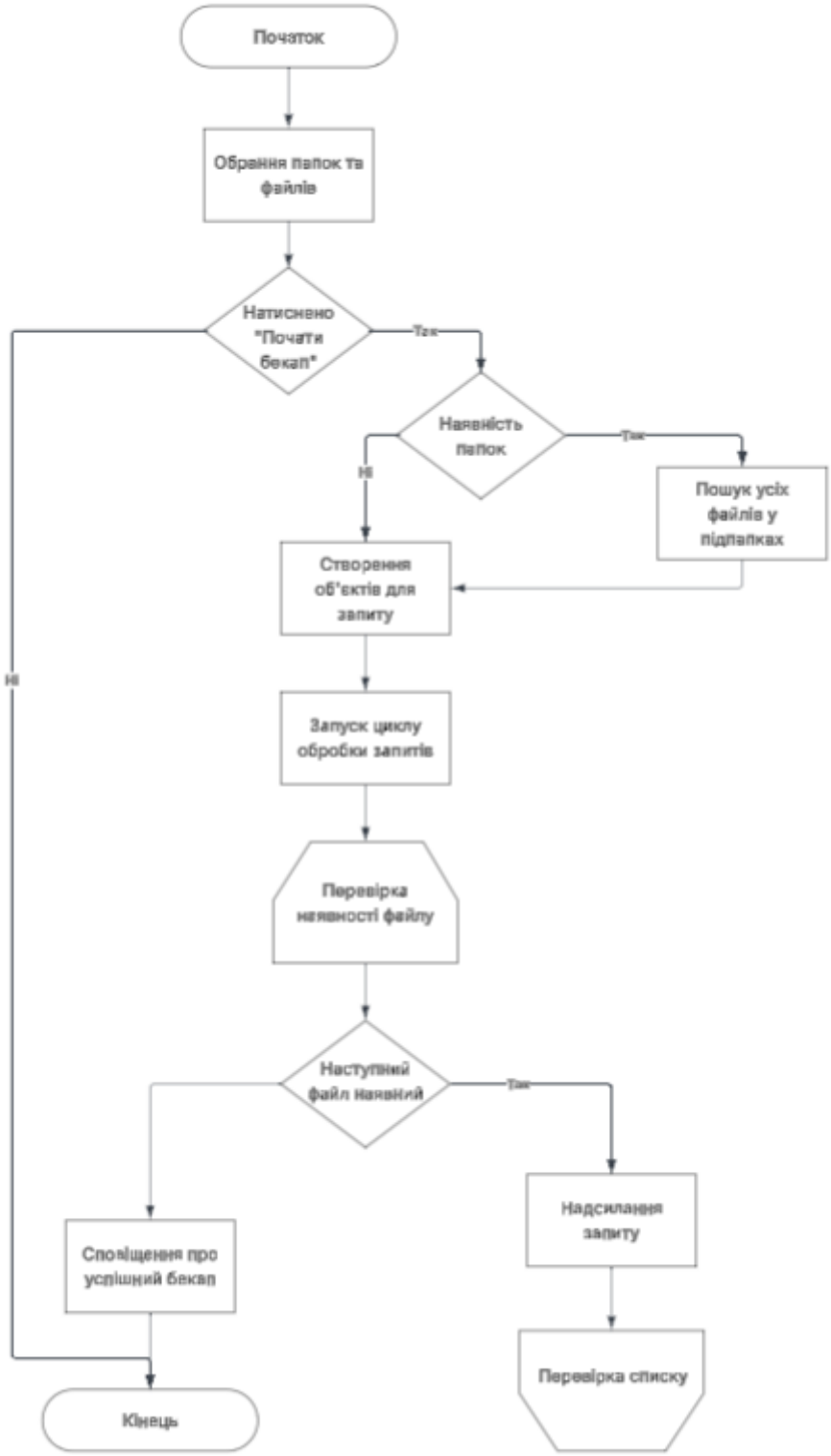
Клієнт-серверна система резервного копіювання зашифрованих файлів

Алгоритм обробки створення резервної копії

ІАЛЦ.045440.006 Д2

Аркушів: 1

Київ – 2025



					ІАЛЦ.045440.006 Д2		
Змін.	Арк.	№ докум.	Підпис	Дата			
Розробив		Ніщик Т. А.			Клієнт-серверна система резервного копіювання зашифрованих файлів	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Сулема О. К.					
Консульт.						КПІ Ім. Ігоря Сікорського, ФПМ КВ-11	
Н. контроль		Клятченко Я. М.			**Алгоритм обробки створення резервної копії**		
Зав. каф.		Романкевич В.О.					

ДОДАТОК В

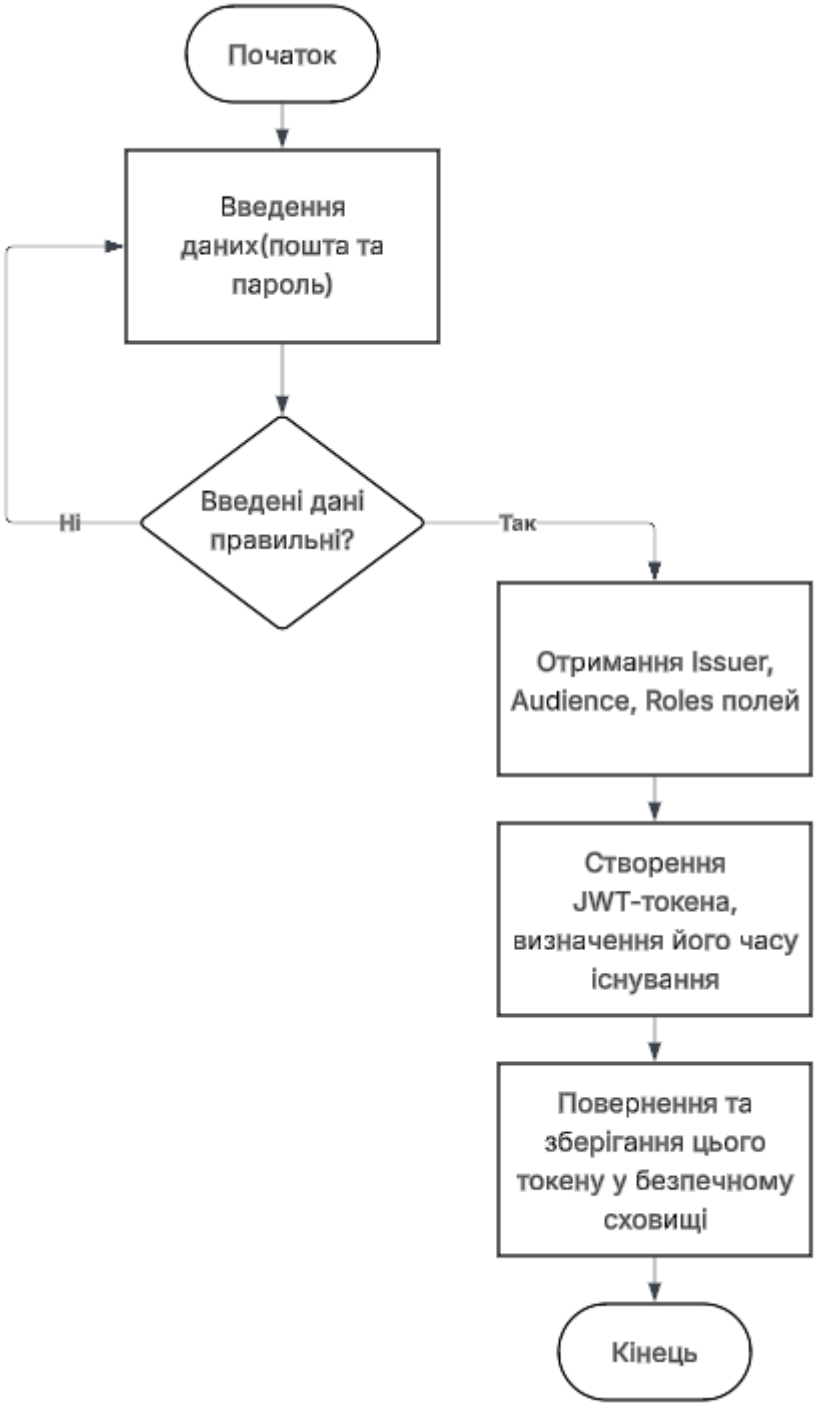
Клієнт-серверна система резервного копіювання зашифрованих файлів

Алгоритм створення токену для авторизації

ІАЛЦ.045440.007 ДЗ

Аркушів: 1

Київ – 2025



					ІАЛЦ.045440.007 ДЗ		
Змін.	Арк.	№ докум.	Підпис	Дата			
Розробив		Ніщик Т. А.			Клієнт-серверна система резервного копіювання зашифрованих файлів	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Сулема О. К.					
Консульт.						КПІ Ім. Ігоря Сікорського, ФПМ КВ-11	
Н. контроль		Клятченко Я. М.			**Алгоритм створення токену для авторизації**		
Зав. каф.		Романкевич В.О.					

ДОДАТОК Г

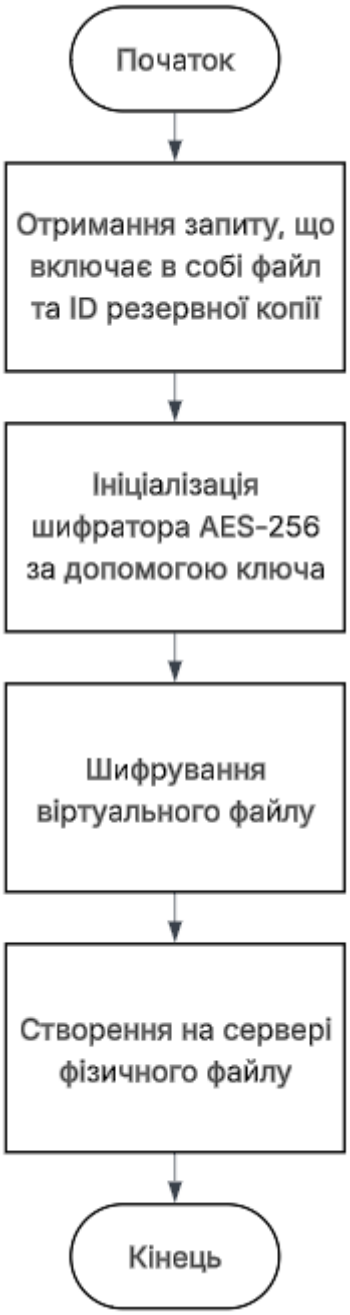
Клієнт-серверна система резервного копіювання зашифрованих файлів

Схема збереження файлу на сервері

ІАЛЦ.045440.008 Д4

Аркушів: 1

Київ – 2025



					ІАЛЦ.045440.008 Д4		
Змін.	Арк.	№ докум.	Підпис	Дата			
Розробив		Ніщик Т. А.			Клієнт-серверна система резервного копіювання зашифрованих файлів	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Сулема О. К.					
Консульт.					**Схема збереження файлу на сервері**	КПІ Ім. Ігоря Сікорського, ФПМ КВ-11	
Н. контроль		Клятченко Я. М.					
Зав. каф.		Романкевич В.О.					

ДОДАТОК Д

Клієнт-серверна система резервного копіювання зашифрованих файлів

Презентація

Аркушів: 9

Київ – 2025

КЛІЄНТ-СЕРВЕРНА СИСТЕМА РЕЗЕРВНОГО КОПІЮВАННЯ ЗАШИФРОВАНИХ ФАЙЛІВ

Студент:
Ніщик Тарас Андрійович
Група: КВ-11

Дипломний керівник:
доцент каф. СПСКС, д-р філ.
Сулема Ольга Костянтинівна

АКТУАЛЬНІСТЬ РОБОТИ

У сучасному цифровому середовищі безпека та збереження даних є критично важливими аспектами функціонування як для окремих користувачів, так і для організацій. Щодня генеруються великі обсяги інформації, частина з якої є конфіденційною або має особливу цінність.

Втрата таких даних унаслідок збоїв системи, вірусних атак, людських помилок або фізичного пошкодження обладнання може призвести до серйозних наслідків — як фінансових, так і репутаційних.

ОГЛЯД НАЯВНИХ РІШЕНЬ

Veeam Backup

Microsoft OneDrive

ПОРІВНЯННЯ НАЯВНИХ РІШЕНЬ

Додаток	Переваги	Недоліки
Veeam Backup	1. Потужна система резервного копіювання та відновлення 2. Підтримка образів системи, віртуальних машин (VM), серверів 3. Автоматизація, розклад, інкрементальні копії 4. Шифрування, контроль доступу, аудит	1. Складніше налаштування для новачків 2. Необхідні ресурси сервера або хосту 3. Платна ліцензія для повного функціоналу 4. Не зберігає файли в хмарі за замовчуванням
Microsoft OneDrive	1. Інтеграція з Windows та Microsoft 365 2. Просте хмарне зберігання для файлів 3. Автоматична синхронізація між пристроями 4. Можливість відновлення попередніх версій файлів	1. Обмежений контроль над резервними копіями 2. Немає повного образу системи або розкладу 3. Залежність від інтернету 4. Обмежений простір без підписки (5 ГБ)

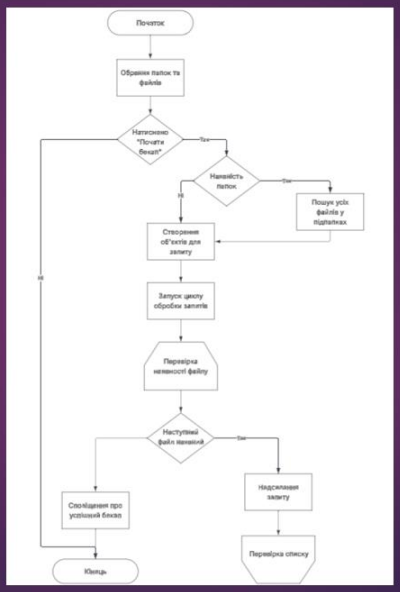
МЕТА ПРОЄКТУ

Метою цього проєкту є покращення процесу створення резервних копій шляхом створення легкої у використанні та освоєнні клієнт-серверної системи для створень резервних копій.

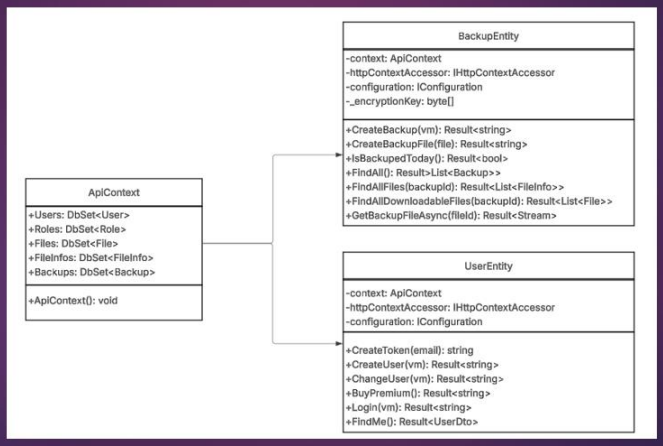
Вимоги до програмного продукту, що розробляється:

- Сумісність з будь якою операційною системою Windows, Mac;
- Можливість автоматичного та ручного резервного копіювання;
- Легкість у використанні та освоєнні програми.

АЛГОРИТМ РОБОТИ ЗАСТОСУНКУ (РЕЗЕРВНОЇ КОПІЇ)



АЛГОРИТМ ВЗАЄМОДІЇ МОДУЛІВ СИСТЕМИ



АЛГОРИТМ СТВОРЕННЯ ТОКЕНУ ДЛЯ АВТОРИЗАЦІЇ

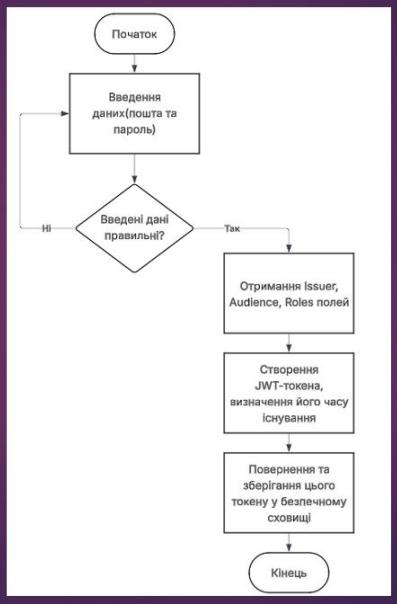
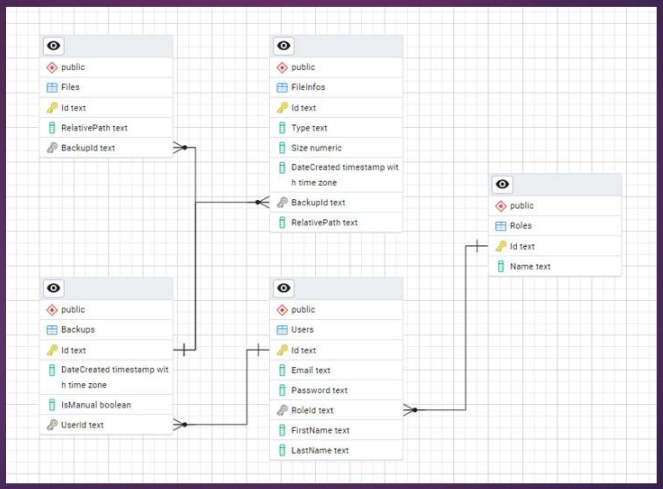
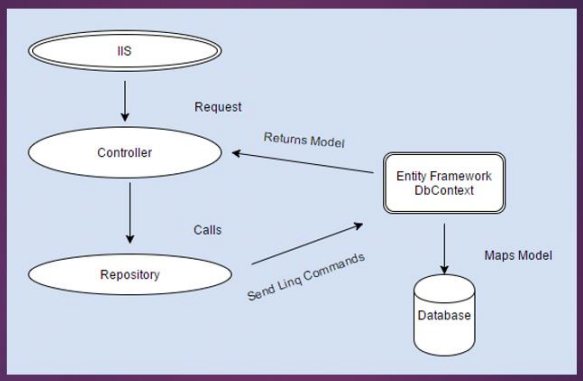


СХЕМА ЗБЕРЕЖЕННЯ ФАЙЛУ НА СЕРВЕРІ

СТРУКТУРА БАЗИ ДАНИХ



АРХІТЕКТУРА СЕРВЕРНОГО ЗАСТОСУНКУ

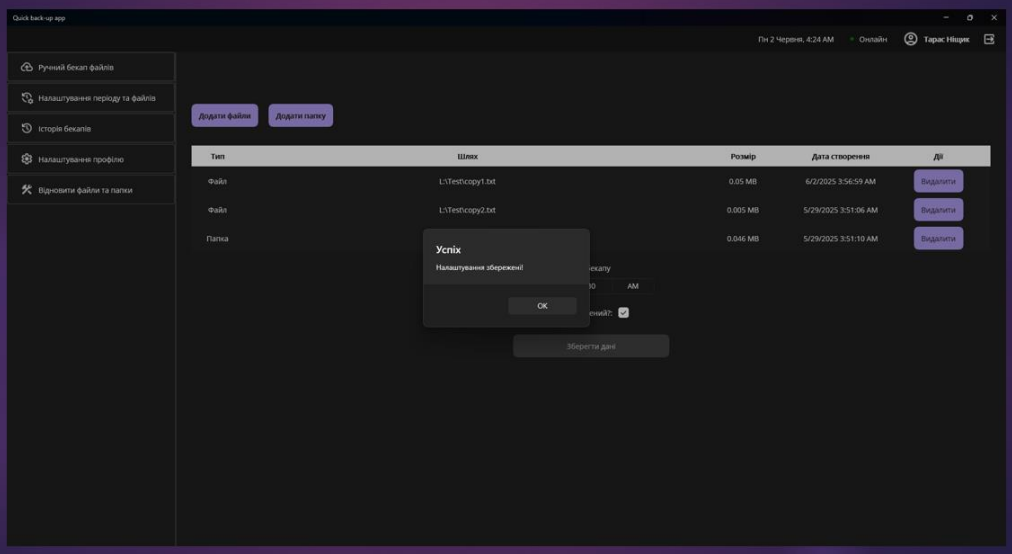


АРХІТЕКТУРА КЛІЄНТНОГО ЗАСТОСУНКУ

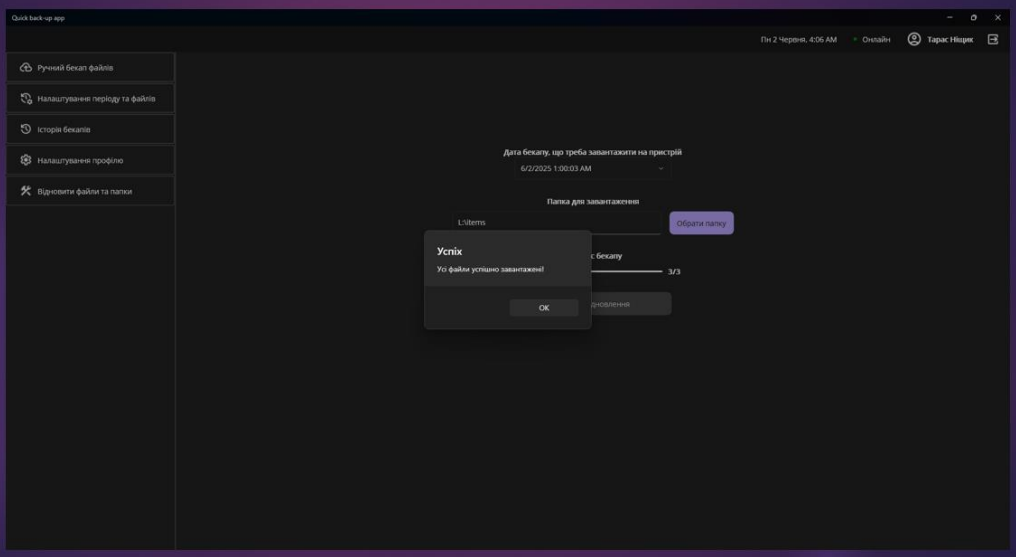
ОБРАНІ ТЕХНОЛОГІЇ

- Реалізацію клієнт-серверної системи для створення резервних копій побудовано за допомогою .NET Core, а саме: .NET MAUI, ASP.NET Core, Entity Framework Core. Це високопродуктивні фреймворки для роботи з API, графічними застосунками та базами даних.
- База даних була створена за допомогою PostgreSQL, швидкою, надійною та зручною у користуванні.

ТЕСТУВАННЯ (1)



ТЕСТУВАННЯ (2)



ВИСНОВКИ (1)

- Проведено аналіз сучасних систем резервного копіювання (Veeam, Duplicati, Microsoft OneDrive) та сформульовано основні вимоги до розроблюваного рішення.
- Розроблено клієнт-серверну систему резервного копіювання з підтримкою шифрування файлів, захистом персональних даних та зручним функціоналом для різних категорій користувачів.
- Реалізовано серверну частину на основі ASP.NET Core з використанням патерна репозиторій, EF Core, автентифікації через JWT, хешування паролів за алгоритмом bcrypt та шифрування файлів за допомогою AES-256.

ВИСНОВКИ (2)

- Створено клієнтську частину на платформі .NET MAUI з використанням архітектури MVVM, CommunityToolkit, AppShell, SecureStorage та асинхронною взаємодією з сервером через HttpClient.
- Реалізовано повний набір функцій: ручне та автоматичне резервне копіювання, історія збережених копій, відновлення файлів, управління профілем користувача, підтримка ролей доступу.
- Проведено тестування основних сценаріїв використання системи, включаючи створення, збереження, шифрування та відновлення резервних копій, а також роботу з налаштуваннями й навантаженнями.

Дякую за увагу!

ДОДАТОК Е

Клієнт-серверна система резервного копіювання зашифрованих файлів

Фрагмент коду

Аркушів: 6

Київ – 2025

BackupEntity.cs

```
using DiplomaProjectAPI.Interfaces;
using DiplomaProjectAPI.Models;
using DiplomaProjectAPI.ViewModels;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Options;
using System.Security.Cryptography;
using File = DiplomaProjectAPI.Models.File;
using FileInfo = DiplomaProjectAPI.Models.FileInfo;

namespace DiplomaProjectAPI.Repository
{
    public class BackupEntity : IBackup
    {
        private readonly ApiContext context;
        private readonly IHttpContextAccessor httpContextAccessor;
        private readonly IConfiguration configuration;
        private readonly byte[] _encryptionKey;

        public BackupEntity(ApiContext context, IConfiguration configuration,
            IHttpContextAccessor httpContextAccessor, IOptions<EncryptionOptions>
opts)
        {
            this.configuration = configuration;
            this.context = context;
            this.httpContextAccessor = httpContextAccessor;
            _encryptionKey = Convert.FromBase64String(opts.Value.Key);
        }

        public async Task<Result<string>> CreateBackup(BackupViewModel vm)
        {
            string? email = httpContextAccessor.HttpContext?.User?.Identity?.Name!;

            var user = await context.Users
                .FirstOrDefaultAsync(u => u.Email == email);

            if (user == null)
                return new Result<string> { Affected = 0, Message = "Користувача не
існує" };

            var backup = new Backup
            {
                DateCreated = DateTime.UtcNow,
                IsManual = vm.IsManual,
                UserId = user.Id
            };

            foreach(var file in vm.FileInfos)
            {
                var fileInfo = new Models.FileInfo
                {
                    DateCreated = DateTime.UtcNow,
                    Size = file.Size,
                    Type = file.Type,
                    RelativePath = file.RelativePath
                };
                backup.FileInfos.Add(fileInfo);
            }

            var model = await context.Backups.AddAsync(backup);
```

```
            await context.SaveChangesAsync();

            return new Result<string> { Affected = 1, Message = "Успішно створенно
бекап!", Model = model.Entity.Id };

        }

        public async Task<Result<string>> CreateBackupFile(string backupId, string
path, IFormFile file)
        {
            var folder = $"L://Uploads//{backupId}//";

            var filePath = path.Substring(3);
            var finalPath = Path.Combine(folder, filePath);

            var directory = Path.GetDirectoryName(finalPath)!;
            if (!Directory.Exists(directory))
                Directory.CreateDirectory(directory);

            using var aes = Aes.Create()!;
            aes.Key = _encryptionKey;
            aes.GenerateIV();

            using var fsOut = new FileStream(finalPath, FileMode.Create,
FileAccess.Write, FileShare.None);

            await fsOut.WriteAsync(aes.IV, 0, aes.IV.Length);

            using var crypto = new CryptoStream(
                fsOut,
                aes.CreateEncryptor(aes.Key, aes.IV),
                CryptoStreamMode.Write
            );

            await file.CopyToAsync(crypto);
            await crypto.FlushFinalBlockAsync();

            var model = new Models.File { BackupId = backupId, RelativePath =
finalPath };

            await context.Files.AddAsync(model);

            await context.SaveChangesAsync();

            return new Result<string> { Affected = 1, Message = "Успішно" };
        }

        public async Task<Result<bool>> IsBackupedToday()
        {
            string? email = httpContextAccessor.HttpContext?.User?.Identity?.Name!;
            var model = await context.Backups
                .Include(b => b.User)
                .AsNoTracking()
                .AnyAsync(b => b.User.Email == email && b.DateCreated.Date ==
DateTime.Now.Date);

            return new Result<bool> { Affected = 1, Model = model };
        }

        public async Task<Result<List<Backup>>> FindAll()
        {
            string? email = httpContextAccessor.HttpContext?.User?.Identity?.Name!;

            var user = await context.Users
                .Include(u => u.Role)
```

```
                .FirstOrDefaultAsync(u => u.Email == email);

            var models = context.Backups
                .Include(b => b.User)
                .Where(b => b.User.Email == email)
                .OrderByDescending(b => b.DateCreated)
                .AsNoTracking();

            if (user.Role.Name != "PREMIUM")
                models = models.Take(5);
            else
                models = models.Take(30);

                return new Result<List<Backup>> { Affected = 1, Model = await
models.ToListAsync() };
        }

        public async Task<Result<List<FileInfo>>> FindAllFiles(string backupId)
        {
            var models = await context.FileInfos
                .Where(fi => fi.BackupId == backupId)
                .ToListAsync();

            return new Result<List<FileInfo>> { Affected = 1, Model = models };
        }

        public async Task<Result<List<File>>> FindAllDownloadableFiles(string
backupId)
        {
            var models = await context.Files
                .Where(fi => fi.BackupId == backupId)
                .ToListAsync();

            return new Result<List<File>> { Affected = 1, Model = models };
        }

        public async Task<Result<Stream>> GetBackupFileAsync(string fileId)
        {
            var fileEntity = await context.Files.FindAsync(fileId);
            if (fileEntity == null)
                return new Result<Stream> { Affected = 0, Message = "Файл не знайдено"
};

            var encryptedPath = fileEntity.RelativePath;
            if (!System.IO.File.Exists(encryptedPath))
                return new Result<Stream> { Affected = 0, Message = "Файл відсутній на
диску" };

            var fs = new FileStream(encryptedPath, FileMode.Open, FileAccess.Read,
FileShare.Read);

            using var aes = Aes.Create()!;
            aes.Key = _encryptionKey;

            var iv = new byte[aes.BlockSize / 8];
            await fs.ReadAsync(iv, 0, iv.Length);
            aes.IV = iv;

            var decryptor = aes.CreateDecryptor(aes.Key, aes.IV);
            using var cryptoStream = new CryptoStream(fs, decryptor,
CryptoStreamMode.Read);

            var ms = new MemoryStream();
            await cryptoStream.CopyToAsync(ms);
            ms.Position = 0;
```

```
            return new Result<Stream> { Affected = 1, Message = "Успішно", Model = ms
};
        }
    }
}
```

UserEntity.cs

```
using BCrypt.Net;
using DiplomaProjectAPI.DTOs;
using DiplomaProjectAPI.Interfaces;
using DiplomaProjectAPI.Models;
using DiplomaProjectAPI.ViewModels;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace DiplomaProjectAPI.Repository
{
    public class UserEntity : IUser
    {
        private readonly ApiContext context;
        private readonly IHttpContextAccessor httpContextAccessor;
        private readonly IConfiguration configuration;

        public UserEntity(ApiContext context, IConfiguration configuration,
IHttpContextAccessor httpContextAccessor)
        {
            this.configuration = configuration;
            this.context = context;
            this.httpContextAccessor = httpContextAccessor;
        }

        public async Task<string> CreateToken(string email)
        {
            try
            {
                var user = await context.Users
                    .AsNoTracking()
                    .Include(u => u.Role)
                    .FirstOrDefaultAsync(u => u.Email == email);

                var claims = new List<Claim>() {
                    new Claim(ClaimTypes.Name, email),
                    new Claim(ClaimTypes.Role, user.Role.Name)
                };
                var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration.GetSection("AppSettings:Toke
n").Value!));
                var creds = new SigningCredentials(key,
SecurityAlgorithms.HmacSha256);
                var token = new JwtSecurityToken(
                    issuer: configuration.GetSection("AppSettings:Issuer").Value!,
                    audience: configuration.GetSection("AppSettings:Audience").Value!,
                    claims: claims,
```

```
                    expires: DateTime.Now.AddDays(7),
                    signingCredentials: creds
                    );
                var jwt = new JwtSecurityTokenHandler().WriteToken(token);
                return jwt;
            }
            catch
            {
                return string.Empty;
            }
        }

        public async Task<Result<string>> CreateUser(UserRegisterViewModel vm)
        {
            var user = await context.Users
                .AsNoTracking()
                .AnyAsync(u => u.Email == vm.Email);

            if (user)
                return new Result<string> { Affected = 0, Message = "Користувач вже
зареєстрований!" };

            var role = await context.Roles.FirstOrDefaultAsync(r => r.Name == "USER");

            var createdUser = new User
            {
                Email = vm.Email,
                FirstName = vm.FirstName,
                LastName = vm.LastName,
                Password = BCrypt.Net.BCrypt.HashPassword(vm.Password),
                RoleId = role.Id
            };

            var model = await context.Users.AddAsync(createdUser);

            await context.SaveChangesAsync();

            return new Result<string> { Affected = 1, Message = "Успішно
зареєстровано!" };
        }

        public async Task<Result<string>> ChangeUser(UserChangeViewModel vm)
        {
            string? email = httpContextAccessor.HttpContext?.User?.Identity?.Name!;

            var user = await context.Users
                .FirstOrDefaultAsync(u => u.Email == email);

            if (user == null)
                return new Result<string> { Affected = 0, Message = "Користувача не
існує" };

            user.FirstName = vm.FirstName;
            user.LastName = vm.LastName;
            if(!string.IsNullOrEmpty(vm.Password))
                user.Password = BCrypt.Net.BCrypt.HashPassword(vm.Password);

            await context.SaveChangesAsync();

            return new Result<string> { Affected = 1, Message = "Успішно змінено
дані!" };
        }

        public async Task<Result<string>> BuyPremium()
        {
```

```
            string? email = httpContextAccessor.HttpContext?.User?.Identity?.Name!;

            var user = await context.Users
                .FirstOrDefaultAsync(u => u.Email == email);

            if (user == null)
                return new Result<string> { Affected = 0, Message = "Користувача не
існує" };

            var role = await context.Roles.FirstOrDefaultAsync(r => r.Name ==
"PREMIUM");

            user.RoleId = role.Id;

            await context.SaveChangesAsync();

            return new Result<string> { Affected = 1, Message = "Успішно придбано
преміум для облікового запису!" };
        }

        public async Task<Result<string>> Login(UserLoginViewModel vm)
        {
            var user = await context.Users
                .AsNoTracking()
                .FirstOrDefaultAsync(u => u.Email == vm.Email);

            if (user == null)
                return new Result<string> { Affected = 0, Message = "Користувача не
існує" };

            var isValid = BCrypt.Net.BCrypt.Verify(vm.Password, user.Password);

            if(!isValid)
                return new Result<string> { Affected = 0, Message = "Введено
неправильний праоль" };

            var token = await CreateToken(user.Email);

            return new Result<string> { Affected = 1, Model = token };
        }

        public async Task<Result<UserDto>> FindMe()
        {
            string? email = httpContextAccessor.HttpContext?.User?.Identity?.Name!;

            var user = await context.Users
                .AsNoTracking()
                .Include(u => u.Role)
                .Where(u => u.Email == email)
                .Select(u => new UserDto
                {
                    FirstName = u.FirstName,
                    LastName = u.LastName,
                    Role = u.Role.Name
                })
                .FirstOrDefaultAsync();

            return new Result<UserDto> { Affected = 1, Model = user };
        }
    }
}
```