Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

СИСТЕМА ЗБЕРЕЖЕННЯ ВЕЛИКИХ ФАЙЛІВ З ВИКОРИСТАННЯМ КРИПТОГРАФІЧНОГО ЗАХИСТУ КОНФІДЕНЦІЙНОСТІ ТА ЦІЛІСНОСТІ

Виконав: студент 2 курсу, групи 2КІ-22М
спеціальності 123 — Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)
Станішевський Д.Ю.
(прізвище та ініціали)
Керівник к.т.н., доцент каф. ОТ
Кадук О.В.
(прізвище та ініціали)
«__» _____ 2023 р.
Рецензент к.т.н., доцент каф. ПЗ
Рейда О.М.
(прізвище та ініціали)
«__» _____ 2023 р.

Допущено до захисту

Завідувач кафедри ОТ

д.т.н., проф. Азаров О. Д.

«30» 12 2023 р

ВНТУ 2023

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Галузь знань — Інформаційні технології
Освітній рівень — магістр
Спеціальність — 123 Комп'ютерна інженерія
Освітньо-професійна програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри обчислювальної техніки

О.Д. Азаров

"26" вересня 2023 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

Студенту Станішевському Дмитру Юрійовичу

1 Тема роботи — «Система збереження великих файлів з використанням криптографічного захисту конфіденційності та цілісності», керівник роботи Кадук О.В., затверджені наказом вищого навчального закладу **18.09.23** року № **247.**

2 Строк подання студентом роботи **11.12.23**.

3 Вихідні дані до роботи: вимоги до функціональності, швидкодії та безпеки.

4 Зміст текстової частини — аналіз систем збереження великих файлів та методів криптозахисту цілісності та конфіденційонсті. Аналіз можливості застосування блокчейн технології для розробки файлової системи. Розробка алгоритму програмної реалізації системи збереження великих файлів. Опис результатів тестування.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): Блок-схема алгоритму завантаження файлу, блок-схема алгоритму доступу, блок-схема алгоритму цілісності.

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завданн прийня
1-3	Кадук О. В., к.т.н., доцент каф. ОТ	19.09.2023р	11.12.202
4	Небава Микола Іванович проф., к.е.н	1.11.2023	11.12.20

7 Дата видачі завдання **19.09.2023** р.

8 Календарний план виконання курсової роботи приведений в табли

Таблиця 2 — Календарний план

№ з/п	Назва етапів МКР	Строк виконання	Пі
1	Постановка задачі	**19.09.23**	
2	Огляд існуючих рішень	**20.09.23**	
3	Розробка структурної схеми	**28.09.23**	
5	Розрахунок аналогової частини	**5.10.23**	
6	Вибір ПЗ для розробки	**15.10.23**	
7	Розробка роботи системи	**23.10.23**	
8	Розрахунок економічної частини	**2.11.23**	
9	Оформлення пояснювальної записки та ілюстративного матеріалу	**12.11.23**	
10	Виконання магістерської кваліфікаційної роботи	**25.11.23**	
11	Перевірка якості виконання магістерської кваліфікаційної роботи та усунення недоліків	**4.12.23**	
12	Підписи супроводжувальних документів у керівника, опонента, нормоконтролера	**8.12.23**	
13	Перевірка «антиплагіат»	**8.12.23**	
14	Попередній захист	**12.12.23**	

Студент _____ Станішевський Д.

Керівник роботи _____ Кадук О

АНОТАЦІЯ

Магістерська дипломна робота складається з 69 сторінок формату А4, на яких є 9 рисунків, перелік джерел посилань містить 25 найменувань.

В кваліфікаційній магістерській роботі був проведений аналіз сучасних рішень для збереження великих файл з криптографічним захистом.

Проведено порівняльний аналіз мов, та обрано найбільш відповідну. Також було обрано інші технології на основі аналізу та порівнянь. Спроектовано високорівневу архітектуру додатку. Створені умови для подальшого розвитку та розширення. Проведено дослідження ефективності запропонованого алгоритму.

Ключові слова: блокчейн, криптозахист, файлова система, безпека даних, розподіленні системи.

ABSTRACT

The master's thesis consists of 69 pages in A4 format, on which there are 9 drawings, and it was intended to contain 25 names.

In the qualified master's work, we analyze current solutions to save large files with cryptographic protection.

A comprehensive analysis of the data was carried out, and the most consistent data was found. Other technologies were also selected based on analysis and comparison. Designed by high-quality architecture as an addition. Creation of minds for further development and expansion. The effectiveness of the proposed algorithm was investigated.

Key words: blockchain, cryptohist, file system, data security, distributed systems.

ЗМІСТ

ВСТУП

З настанням епохи великих даних кількість та обсяги генерованої інформації зростають експоненціально, висуваючи нові вимоги до систем зберігання та обробки даних. Однією з ключових проблем є збереження великих файлів, яке вимагає не тільки великого простору для зберігання, але й розширених заходів безпеки для забезпечення конфіденційності та цілісності цих даних. Ця дипломна робота зосереджується на розробці системи, яка вирішує цю проблему, використовуючи передові криптографічні техніки.

Актуальність теми посилюється не лише зростанням обсягів даних, але й збільшенням кібератак, які цілять на викрадення або пошкодження важливої інформації. Зловмисники постійно вдосконалюють свої методи, тому системи зберігання повинні бути оснащені сучасними та надійними засобами захисту для протистояння таким загрозам.

Метою даної роботи є розробка та аналіз системи збереження великих файлів, що інтегрує криптографічні алгоритми для захисту конфіденційності та цілісності даних. Важливість такої системи випливає з необхідності забезпечення безпеки великих обсягів інформації, які обробляються і зберігаються в різноманітних ділових та особистих сферах, від фінансових установ до приватних користувачів.

Для досягнення поставленої мети були визначені наступні завдання дослідження: проаналізувати сучасний стан проблеми зберігання великих файлів, вивчити існуючі криптографічні методи та їх придатність для застосування у системах зберігання, розробити власну модель системи, що використовує передові криптографічні рішення, та оцінити ефективність запропонованої системи через серію тестів і моделювань.

Методологія дослідження охоплює комплексний підхід, який включає як теоретичний аналіз сучасної літератури та джерел, так і практичну реалізацію прототипу системи. Теоретична частина зосереджена на огляді сучасних

тенденцій у криптографії та системах зберігання даних, тоді як практична частина включає проектування, реалізацію та оцінку працездатності розробленої системи.

Завершуючи вступ, важливо підкреслити, що ефективне збереження великих файлів із забезпеченням високого рівня безпеки є критичним для сучасного інформаційного суспільства. Ця робота має на меті не тільки розробити таку систему, але й зробити її доступною та зрозумілою для широкого кола фахівців у галузі IT та кібербезпеки.

1 ТЕОРЕТИЧНІ ОСНОВИ КРИПТОГРАФІЇ В СУЧАСНИХ СИСТЕМАХ ЗБЕРЕЖЕННЯ ФАЙЛІВ

1.1 Загальні принципи криптографічного захисту

Криптографія — це фундаментальна наука, що займається вивченням методів шифрування інформації, яка дозволяє забезпечувати конфіденційність, автентичність, цілісність та невідкидуваність даних. Вона є ключовим елементом у захисті інформації в сучасних комп'ютерних системах і мережах. Сучасна криптографія використовує математичні алгоритми для перетворення зрозумілої інформації у форму, яка може бути прочитана лише особою, що має відповідний ключ для дешифрування.

Конфіденційність забезпечується шляхом шифрування даних, яке перетворює відкритий текст у шифротекст за допомогою шифру і ключа. Тільки особи, які мають відповідний ключ, можуть перетворити шифротекст назад у відкритий текст. Це є основним принципом захисту даних від несанкціонованого доступу.

Автентичність передбачає засвідчення того, що дані дійсно надійшли від зазначеного відправника. Це часто досягається за допомогою цифрових підписів, які використовують асиметричні криптосистеми. Цифровий підпис, схожий на відбиток пальця, є унікальним для кожного повідомлення та відправника, і його не можна підробити без відповідного приватного ключа.

Цілісність даних означає, що вміст даних не був змінений під час передачі або зберігання. Хеш-функції генерують унікальний "відбиток" або хеш від даних, який потім можна перевірити на кінцевому етапі. Якщо хеш не відповідає початковому, це свідчить про зміну даних.

Невідкидуваність гарантує, що жодна зі сторін не може заперечити свою участь у транзакції. Цифрові підписи та протоколи засвідчення підтверджують, що конкретний обмін даними відбувся, і забезпечують юридичні докази в разі спроби заперечення.

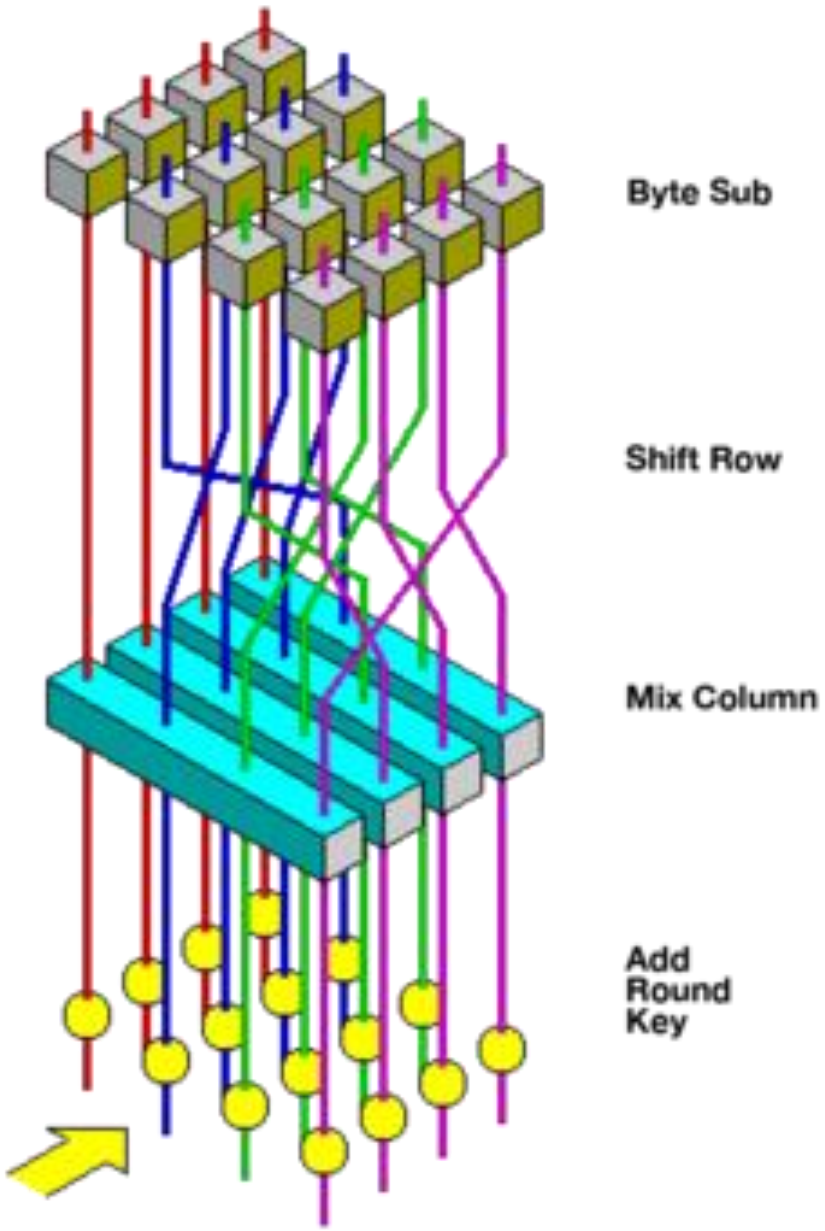

Рис 1.1 Схема роботи симетричного AES алгоритму

Ключовим аспектом криптографічного захисту є криптографічні алгоритми. Вони поділяються на два основних типи: симетричні та асиметричні. Симетричні алгоритми використовують один і той самий ключ для шифрування та дешифрування даних. Найвідомішим прикладом симетричного алгоритму є Advanced Encryption Standard (AES) (Рис 1.1). Асиметричні алгоритми, такі як

RSA, використовують пару ключів — приватний та публічний. Публічний ключ може бути відомий всім, а приватний ключ залишається у володінні власника.

Для забезпечення цілісності, використовуються хеш-функції. Хеш-функції перетворюють довільний набір вхідних даних у короткий, фіксований розмір хеш. Secure Hash Algorithm 2 (SHA-2) є однією з найпопулярніших хеш-функцій, яка використовується в багатьох застосунках безпеки (Рис 1.2).

Протоколи розподілу ключів дозволяють двом сторонам, які не мають попередньої угоди, створити спільний секретний ключ, який може бути використаний для шифрування подальших комунікацій. Протокол Diffie-Hellman є одним з перших прикладів таких протоколів.

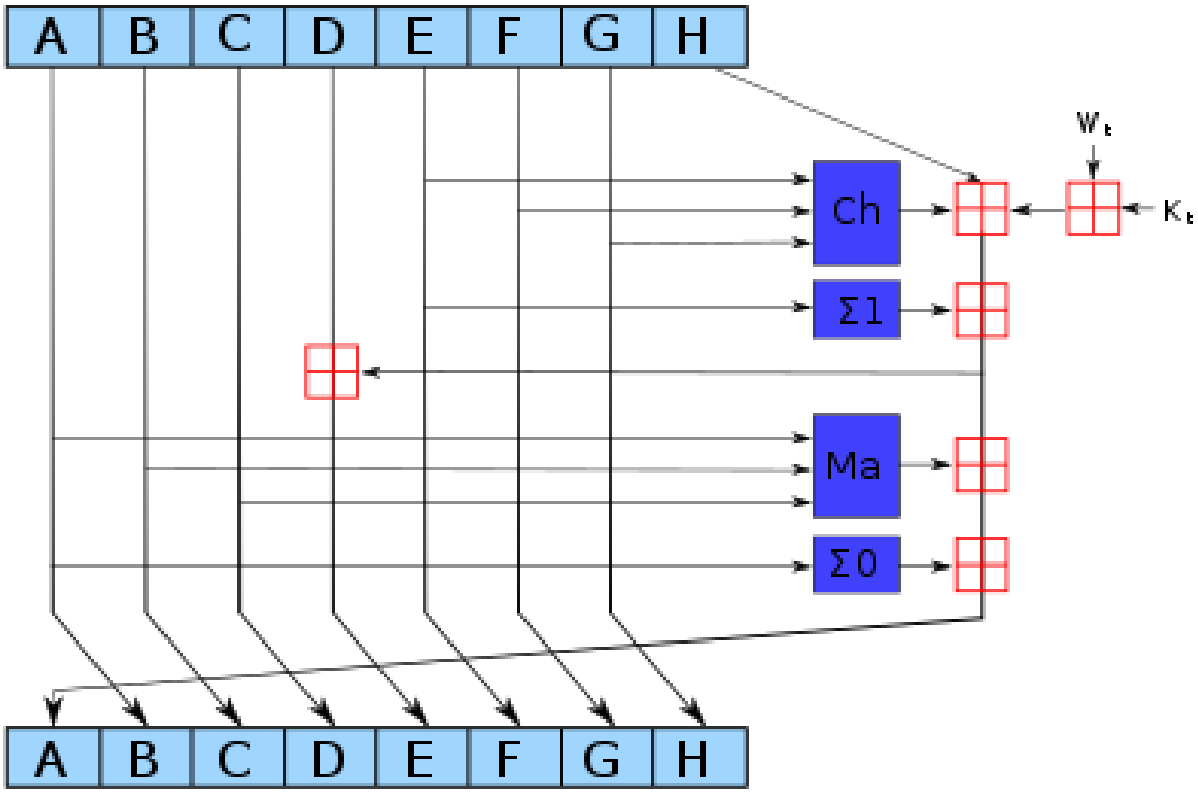

Рис 1.2 Схема однієї ітерації асиметричного SHA-2 алгоритму

Завершуючи розгляд теоретичних основ криптографії, необхідно звернути увагу на правові та етичні аспекти, які мають вирішальне значення у сфері криптографічного захисту. У світі, де дані є валютою електронної комерції,

правові міркування стосуються в основному захисту інформації, права на приватність та використання криптографічних технологій у відповідності до закону. З огляду на різні міжнародні закони, які регулюють експорт та використання криптографічних технологій, особливу увагу слід приділити забезпеченню, щоб системи, що розробляються, відповідали цим нормам. Етичні аспекти, в свою чергу, стосуються моральних обов'язків розробників та користувачів систем забезпечення безпеки інформації, зокрема, у сферах, пов'язаних із конфіденційністю особистих даних та корпоративної інформації.

Окрім того, розуміння глобальних стандартів та регулятивних вимог відіграє ключову роль у розробці та впровадженні криптографічних систем. Нормативні документи, такі як ISO/IEC 27001, встановлюють вимоги до систем управління інформаційною безпекою, в тому числі до застосування криптографічних методів. Дотримання цих стандартів не лише сприяє міжнародній сумісності та взаємодії систем, але й забезпечує високий рівень довіри та безпеки серед користувачів та клієнтів. Враховуючи це, розробники та фахівці у галузі криптографії повинні постійно слідкувати за оновленнями в цих стандартах та регуляціях, а також бути готовими до адаптації своїх систем до нових вимог.

1.2 Симетричне та асиметричне шифрування

Сучасний світ цифрових технологій неухильно залежить від безпечного обміну даними. Центральну роль у цьому процесі відіграє криптографія, яка ділиться на два основних типи шифрування: симетричне та асиметричне. Обидва типи є критично важливими для забезпечення конфіденційності, цілісності та аутентичності інформації в цифровому світі.

Симетричне шифрування — це метод, при якому використовується один і той же ключ для шифрування та дешифрування даних. Його головна перевага полягає у швидкості обробки, що робить його ідеальним для застосування у ситуаціях, де потрібно швидко обробляти великі обсяги даних. Серед найпопулярніших алгоритмів симетричного шифрування — AES (Advanced Encryption Standard) та

DES (Data Encryption Standard). AES широко використовується урядовими та фінансовими організаціями через його надійність і складність. DES, хоча й застарів, все ще використовується у деяких системах, але частіше його замінюють більш безпечними алгоритмами, такими як Triple DES.

Асиметричне шифрування, з іншого боку, використовує пару ключів – публічний та приватний. Публічний ключ використовується для шифрування даних, тоді як приватний ключ — для їх дешифрування. Цей метод є основою багатьох сучасних безпекових протоколів, включаючи SSL/TLS, які забезпечують безпеку в Інтернеті. Найбільш відомим прикладом асиметричного шифрування є алгоритм RSA (Rivest–Shamir–Adleman), який використовується для захисту конфіденційної інформації в онлайн-транзакціях. Інший важливий алгоритм — ECC (Elliptic Curve Cryptography), який пропонує високу безпеку при меншому розмірі ключа, що робить його більш ефективним у мобільних та вбудованих пристроях.

Рис 1.3 Принцип роботи асиметричного шифрування

Однак, симетричне та асиметричне шифрування не існують ізольовано один від одного. На практиці часто використовують гібридні системи, які поєднують обидва методи. Наприклад, у захищених веб-з'єднаннях асиметричне шифрування використовується для безпечного обміну симетричними ключами (це відомо як процес "рукостискання" у TLS), після чого дані шифруються симетричними ключами для ефективної передачі.

Вибір між симетричним та асиметричним шифруванням залежить від конкретних потреб системи. Симетричне шифрування ідеально підходить для ситуацій, де швидкість є критичною, наприклад, для шифрування великих файлів або баз даних. Асиметричне шифрування, з іншого боку, є вибором для ситуацій, де потрібна висока безпека і де можливий обмін ключами між сторонами без попереднього секретного зв'язку.

1.3 Хеш-функції та цифрові підписи

Хеш-функції та цифрові підписи становлять критично важливу частину сучасної криптографії, забезпечуючи цілісність, аутентичність та невідкидуваність інформації в цифровому просторі.

Хеш-функції - це алгоритми, які перетворюють вхідні дані будь-якого розміру в короткий, фіксований розмір вихідного хешу. Цей процес має бути швидким, детермінованим, та забезпечувати унікальність хешів для унікальних даних. Якщо дві різні вхідні порції даних дають однаковий хеш, це називається колізією. Колізії є небажаними, але вони можливі через обмежений розмір хешу. Тому важливо, щоб хеш-функції мінімізували їх імовірність.

Основне застосування хеш-функцій - це забезпечення цілісності даних. Коли дані передаються або зберігаються, хеш цих даних також може бути згенерований та збережений. Пізніше хеш можна перегенерувати та порівняти зі збереженим хешем, щоб переконатися, що дані не були змінені. Хеш-функції також використовуються в аутентифікації та цифрових підписах, а також є основою для багатьох методів шифрування та протоколів безпеки.

Цифровий підпис - це криптографічний механізм, який дозволяє автору або відправнику повідомлення підтвердити свою ідентичність та забезпечити, що повідомлення не було змінене під час передачі. Цифровий підпис включає в себе застосування хеш-функції до повідомлення, а потім шифрування отриманого хешу за допомогою приватного ключа відправника.

Для створення цифрового підпису, спочатку генерується хеш від повідомлення. Цей хеш потім шифрується приватним ключем відправника, створюючи цифровий підпис. Коли повідомлення і його підпис надходять до одержувача, він використовує публічний ключ відправника для розшифровки підпису і порівнює отриманий хеш з хешем, який був ним згенерований з оригінального повідомлення. Якщо хеші співпадають, це підтверджує, що повідомлення є автентичним і не було змінене.

Хеш-функції та цифрові підписи є життєво важливими для забезпечення безпеки в цифровому світі. Вони не тільки забезпечують цілісність та аутентичність даних, але й відіграють ключову роль в безпеці електронної комерції, цифрових комунікацій та багатьох інших областях. Враховуючи швидкий розвиток цифрових технологій та кіберзагроз, розуміння та правильне використання хеш-функцій та цифрових підписів стає все більш важливим для захисту інформації.

1.4 Протоколи розподілу ключів

Протоколи розподілу ключів відіграють фундаментальну роль у криптографії, особливо в контексті симетричного шифрування. Вони дозволяють двом або більше сторонам домовитися про спільний секретний ключ, який може бути використаний для шифрування та дешифрування повідомлень, не передаючи сам ключ через незахищені канали.

Однією з основних проблем симетричного шифрування є необхідність в безпечному обміні ключами між сторонами. Якщо зловмисник перехопить ключ під час його передачі, він зможе розшифрувати будь-яке шифроване

повідомлення. Протоколи розподілу ключів вирішують цю проблему, дозволяючи сторонам безпечно встановлювати спільний секретний ключ, навіть якщо всі їхні комунікації перехоплюються.

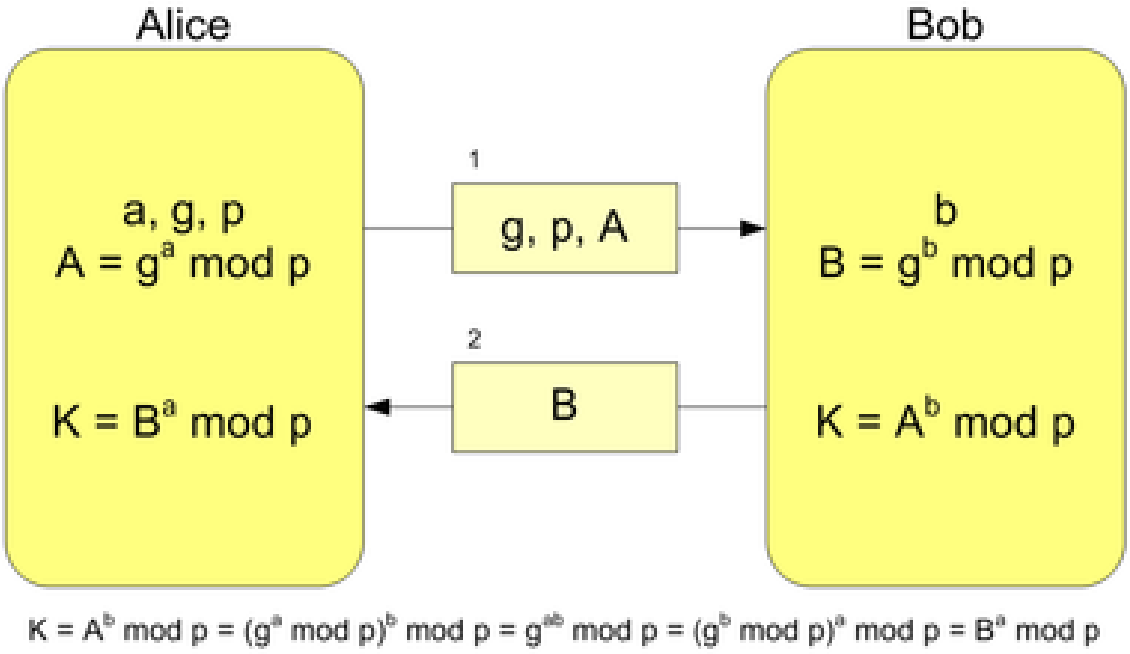

Рис 1.4 Принципи роботи протоколу Diffie-Helman

Одним з найвідоміших прикладів протоколу розподілу ключів є протокол Diffie-Hellman (рис 1.4), розроблений Уїтфілдом Діффі та Мартіном Хеллманом. Цей протокол дозволяє двом сторонам створити спільний секретний ключ, використовуючи публічні та приватні значення, без фактичної передачі самого ключа. В основі протоколу лежить складність обчислення дискретних логарифмів в скінченних полях, що робить відновлення приватних ключів надзвичайно складним на практиці.

Інший метод розподілу ключів пов'язаний з використанням асиметричного шифрування RSA. В цьому випадку, одна сторона може зашифрувати секретний ключ своїм приватним ключем та передати його іншій стороні, яка може розшифрувати його своїм публічним ключем. Цей метод, хоча і безпечний, є менш ефективним порівняно з Diffie-Hellman, особливо при великій кількості учасників.

В останні роки набирають популярності протоколи розподілу ключів на основі еліптичних кривих (ECDH). Вони пропонують більш високу ступінь безпеки при меншому розмірі ключів порівняно з традиційними методами, такими як Diffie-Hellman або RSA.

Незважаючи на їх важливість, протоколи розподілу ключів не без недоліків. Основним викликом є квантові обчислення, які можуть зламати багато сучасних методів шифрування. Тому дослідницька спільнота активно працює над розробкою нових, більш безпечних протоколів, що будуть стійкі до квантових атак.

1.5 Використання блокчейна у забезпеченні цілісності та безпеки

Блокчейн - це революційна технологія, яка забезпечує децентралізований, незмінний та прозорий спосіб зберігання даних. На відміну від традиційних баз даних, які керуються централізованим авторитетом, блокчейн складається з ланцюга блоків, які містять інформацію та зв'язані між собою за допомогою криптографії. Кожен блок містить хеш попереднього блоку, тим самим створюючи ланцюг, який неможливо змінити без зміни всіх наступних блоків (Рис 1.5).

Рис 1.5 Структура блокчейну

Децентралізація: Відсутність централізованого контролю чи єдиного точки збою.

Незмінність: Одного разу записані дані не можуть бути змінені або видалені.

Прозорість: Всі транзакції є публічними та можуть бути переглянуті будь-ким.

Безпека: Використання криптографічних методів забезпечує цілісність і безпеку даних.

Блокчейн ефективно забезпечує цілісність даних завдяки своїй структурі. Зміна інформації в одному блоку вимагала б зміни всіх наступних блоків в ланцюгу, що є практично неможливим у великих блокчейн-мережах, як-от Bitcoin чи Ethereum. Це робить блокчейн ідеальним для застосувань, де цілісність даних є критично важливою.

Безпека блокчейну посилюється завдяки використанню складних криптографічних алгоритмів. Крім того, децентралізована природа технології робить її стійкою до централізованих атак або контролю. Це робить блокчейн привабливим для створення безпечних систем записів, цифрових ідентифікаторів, смарт-контрактів та інших застосувань.

2 АНАЛІЗ НАЯВНИХ РІШЕНЬ. ВИЯВЛЕННЯ ЇХ СИЛЬНИХ ТА СЛАБКИХ СТОРІН. ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ КОНКУРЕНТНОГО АЛГОРИТМУ ТА СИСТЕМИ

2.1 Огляд наявних систем

Розглянемо докладніше деякі існуючі системи збереження великих файлів, їхні можливості та обмеження:

Хмарні сховища:

Google Drive (Рис 2.1) надає користувачам безкоштовний обсяг сховища, а також інтеграцію з іншими Google-сервісами. Він пропонує можливість ділитися файлами та спільно працювати над ними. Крім того, в Google Drive можна застосовувати криптографічні методи для захисту конфіденційності, такі як двофакторна аутентифікація.

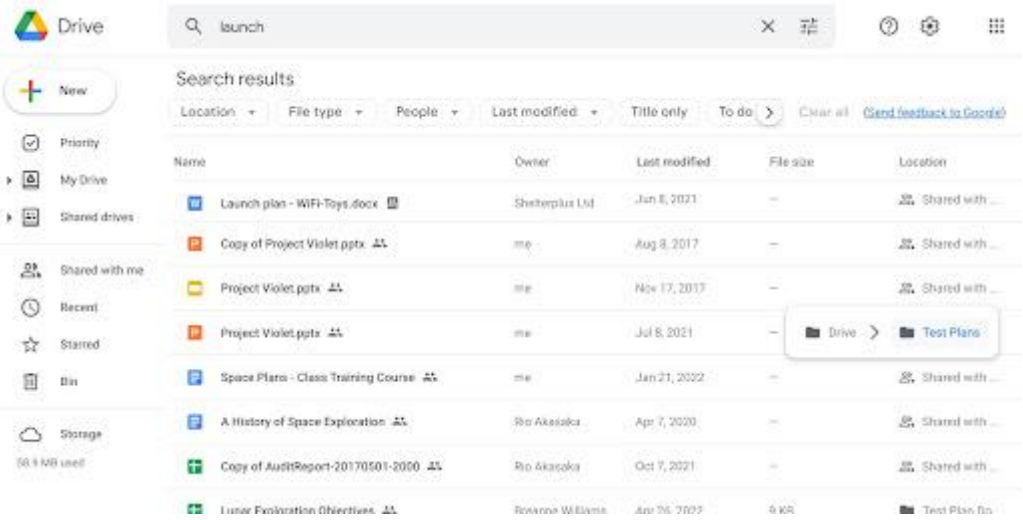

Рис 2.1 Вікно web-додатку Google Drive

Dropbox: Dropbox також є популярним хмарним сховищем, яке дозволяє синхронізувати файли між різними пристроями. Він використовує шифрування на клієнтському та серверному рівнях для захисту даних. Однак, Dropbox має доступ до даних користувача, що може створювати питання з точки зору конфіденційності (Рис 2.2).

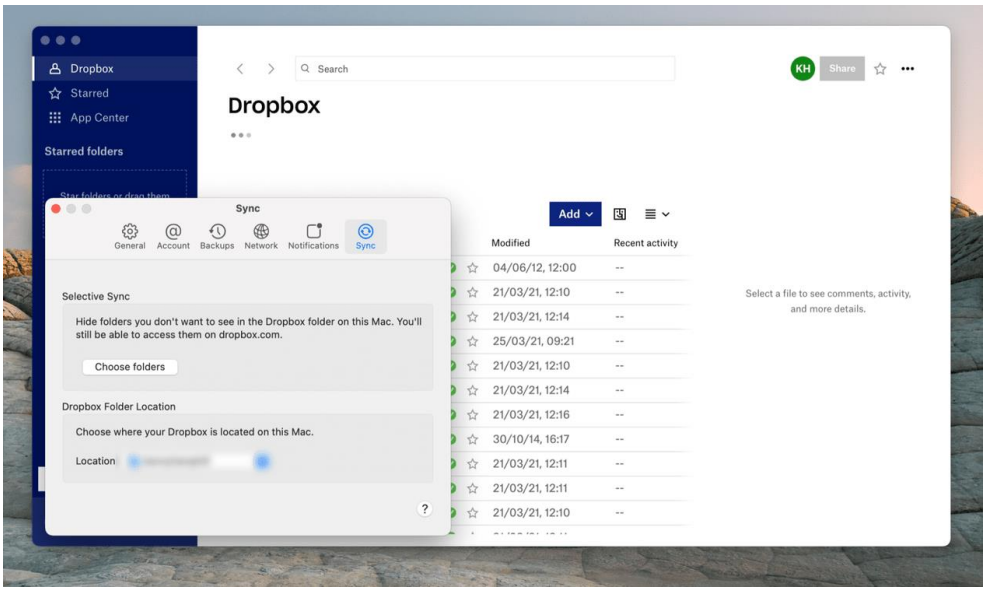

Рис 2.2 Вікно додатку Dropbox

Amazon S3 (Simple Storage Service): Amazon S3 надає потужні можливості для збереження великих файлів та інтеграції з іншими послугами Amazon Web Services (AWS). AWS також надає інструменти для шифрування даних під час передачі та збереження, а також для управління доступом до файлів.

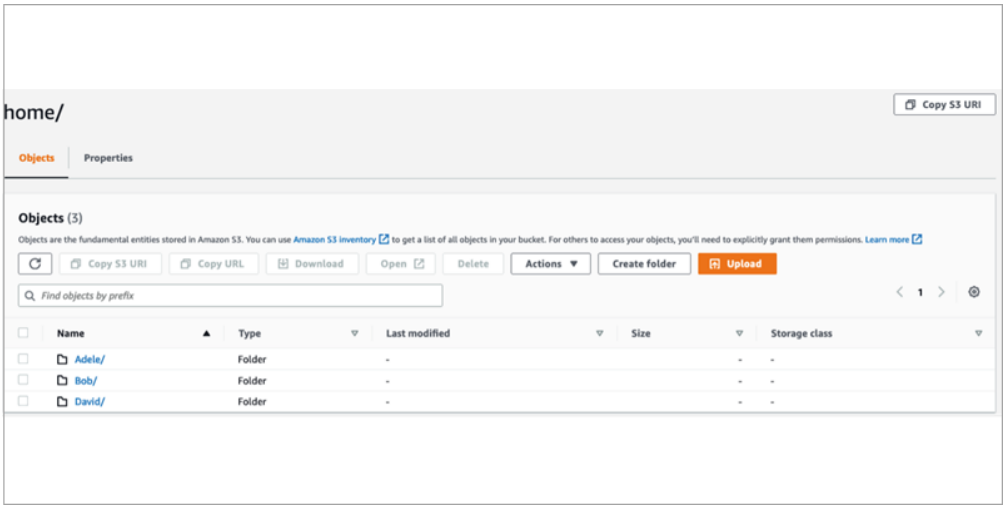

Рис 2.3 Інтерфейс системи S3

Локальні системи збереження:

Network Attached Storage (NAS): NAS пропонує локальне збереження великих файлів на мережевих пристроях. Вони можуть бути налаштовані з використанням різноманітних рівнів захисту, включаючи шифрування та контроль доступу. NAS зазвичай використовуються організаціями для забезпечення конфіденційності і цілісності даних.

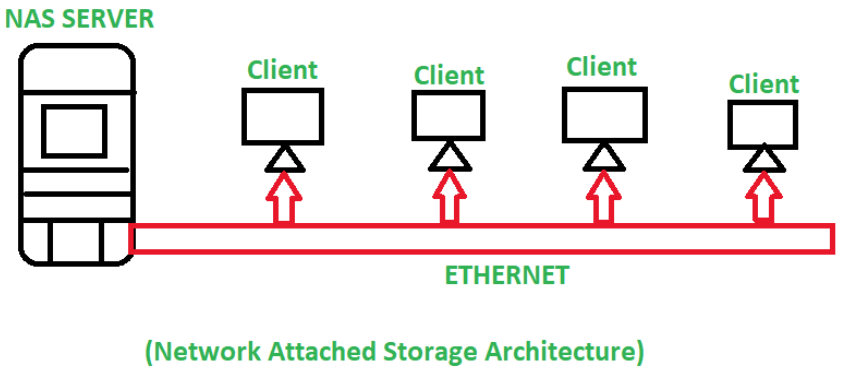

Рис 2.4 Базова архітектура NAS

Системи засновані на блокчейні:

Filecoin: Filecoin - це система для збереження великих файлів, яка використовує блокчейн. Вона дозволяє користувачам зберігати файли в розподіленій мережі за допомогою технології блокчейн і нагороджує мережевих учасників за надання зберігальних ресурсів. Це забезпечує високу цілісність і доступність даних.

Гібридні рішення:

OwnCloud: OwnCloud - це приклад гібридної системи збереження файлів, яка дозволяє користувачам створювати власний хмарний сховище на власному сервері або у хмарі, залежно від їхніх потреб. OwnCloud підтримує шифрування на стороні сервера та клієнта для забезпечення безпеки даних.

Проблеми та виклики:

Безпека даних: Усі системи повинні забезпечити захист великих файлів від несанкціонованого доступу та втрати даних.

Скалабельність: Якщо потрібно зберігати велику кількість даних, система повинна бути скалабельною і готовою до росту.

Доступність та продуктивність: Великі файли можуть вимагати високої швидкості передачі та доступу до них, що може бути важко забезпечити.

Враховуючи ці фактори, розробка системи збереження великих файлів з криптографічним захистом конфіденційності та цілісності повинна враховувати як існуючі рішення, так і вищезазначені виклики, щоб забезпечити оптимальну працездатність та безпеку великих файлів.

2.2 Конкурентний аналіз

Google Drive:

Переваги:

Інтеграція з іншими Google-сервісами, такими як Gmail та Google Docs.

Легкий спільний доступ до файлів з іншими користувачами.

Автоматична синхронізація між різними пристроями.

Недоліки:

Обмежений безкоштовний обсяг зберігання.

Не такий розширений функціонал для бізнес-користувачів порівняно з іншими рішеннями.

Dropbox:

Переваги:

Простий інтерфейс та можливість легко спільно користуватися файлами.

Ефективна синхронізація між пристроями.

Високий рівень безпеки даних.

Недоліки:

Можливо високі витрати для користувачів з багато великих файлів.

Обмежені можливості для безкоштовного обсягу зберігання.

Amazon S3:

Переваги:

Висока доступність і надійність.

Масштабованість і можливість налаштовувати зберігання під конкретні потреби.

Прекрасна підтримка з боку Amazon і велика спільнота користувачів.

Недоліки:

Цінова політика може бути складною для розуміння, особливо для новачків.

Потребує досвіду в адмініструванні.

NAS (Network Attached Storage):

Переваги:

Повний контроль над зберіганням даних та приватністю.

Зручний доступ до файлів в локальній мережі.

Можливість розширення зберігання за потребою.

Недоліки:

Вимагає фізичного обладнання та налаштування.

Обмежена можливість доступу зовнішніми користувачами.

Filecoin:

Переваги:

Децентралізована система збереження на основі блокчейну.

Можливість заробітку за надання зберігання іншим користувачам.

Недоліки:

Не така широка підтримка та інтеграція як у централізованих рішеннях.

Складніше для новачків і тих, хто не має досвіду з блокчейном.

ownCloud:

Переваги:

Повний контроль над зберіганням та конфіденційністю.

Можливість самостійного розгортання та налаштування.

Доступ до додатків та розширень.

Недоліки:

Вимагає більше технічного обізнання для розгортання та адміністрування.

Обмежена спільнота користувачів порівняно зі великими гравцями.

Отже можна зробити висновки, що сучасні системи ще не використовують масово блокчейн для захисту цілісності даних. Тому було б доцільно розробити таку систему.

2.3 Вибір мови розробки

В якості мови програмування буде використовуватись C#, оскільки це сучасна, об'єктно-орієнтована мова програмування, розроблена корпорацією Microsoft. Ця мова стала однією з найпопулярніших в світі програмування завдяки своїм потужним можливостям, великому спільноті розробників і використанню у широкому спектрі застосувань, включаючи веб-розробку, десктопні додатки, мобільні додатки та багато інших сфер.

Основні особливості C#:

Об'єктно-орієнтована мова: C# підтримує об'єктно-орієнтоване програмування (ООП), що дозволяє розробникам створювати класи і об'єкти для організації коду і даних.

Мультиплатформенність: C# може бути використаний на різних платформах, включаючи Windows, macOS і Linux. Це можливо завдяки платформі .NET Core (тепер .NET 5 і більше), яка підтримує багатоплатформовий розвиток.

Сильна типізація: Мова має сильну типізацію, що допомагає уникнути багатьох помилок на етапі компіляції.

Автоматичне управління пам'яттю: C# використовує сміттєзбірник для автоматичного вивільнення невикористовуваних об'єктів, що полегшує роботу з пам'яттю.

Багатопотоковість: Мова має вбудовану підтримку багатопотоковості, що дозволяє створювати паралельні і асинхронні додатки.

Велика стандартна бібліотека: C# поставляється з великою стандартною бібліотекою класів (FCL - Framework Class Library), яка містить численні класи та методи для різних завдань, таких як робота з файлами, мережевими операціями, робота з базами даних і багато інших.

Широкий вибір інструментів розробки: Для розробки в C# доступні різні інтегровані середовища розробки, такі як Visual Studio, Visual Studio Code і інші, які спрощують процес розробки і налагодження.

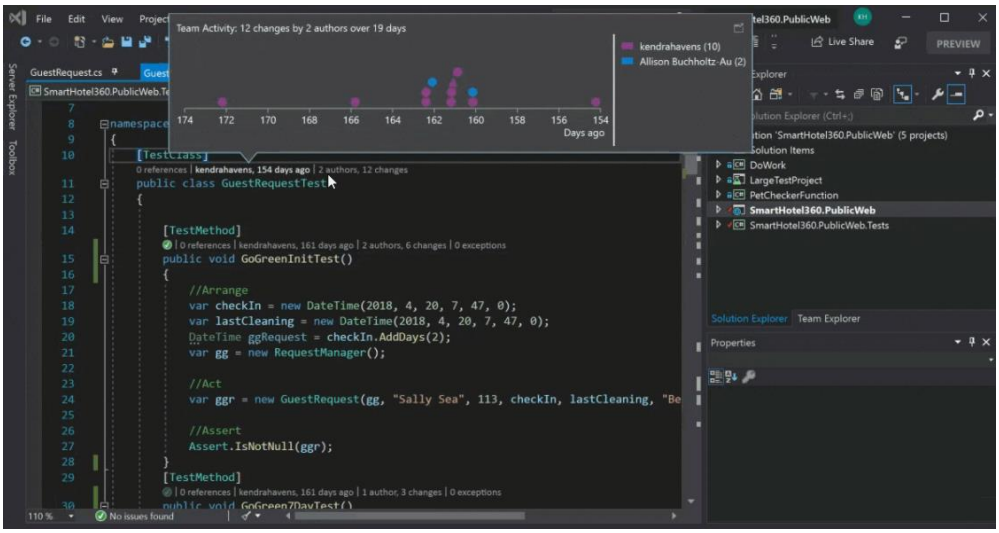

Рис 2.5 Вікно Visual Studio – основого середовища розробки мовою C#

Велика спільнота і підтримка: Є велика спільнота розробників C#, яка активно обмінюється досвідом і надає допомогу. Крім того, Microsoft надає широкий спектр ресурсів і оновлень.

Підтримка веб-розробки: C# може бути використаний для розробки веб-додатків за допомогою технологій ASP.NET і ASP.NET Core.

Мобільна розробка: Існують фреймворки, такі як Xamarin, які дозволяють розробляти мобільні додатки для Android і iOS, використовуючи C#.

2.4 Вибір платформи реалізації

В якості платформи буде використовуватись Ethereum, це децентралізована блокчейн-платформа, яка набула широкої популярності завдяки можливостям створення і виконання смарт-контрактів. Смарт-контракти - це програми, які запускаються автоматично при виконанні певних умов і можуть забезпечувати безпеку, надійність і децентралізацію в різних сферах. В цій статті розглянемо, чому Ethereum є доцільним вибором для розробки системи збереження файлів на блокчейн-платформі.

Основні переваги Ethereum для збереження файлів

Децентралізація. Однією з ключових переваг Ethereum є децентралізація. У системі збереження файлів на блокчейні немає одного центрального сервера чи посередника, що забезпечує високий рівень безпеки та доступності. Кожен вузол мережі має копію даних, і вони взаємодіють між собою за допомогою смарт-контрактів, що робить систему менш вразливою до атак і відмов.

Смарт-контракти. Вони в Ethereum можуть бути використані для автоматизації процесів збереження файлів. Наприклад, можна створити смарт-контракт, який дозволяє користувачам завантажувати файли, встановлювати права доступу до них та отримувати винагороду за зберігання. Це робить систему прозорою та автоматизованою.

Криптографічний захист. Ethereum використовує криптографію для забезпечення безпеки та конфіденційності даних. Файли можуть бути шифровані перед збереженням і розшифровані лише власником ключа. Крім того, кожен запис в блокчейні підписується цифровим підписом, що дозволяє перевірити цілісність даних.

Розширюваність. Ethereum працює над рішеннями для покращення масштабованості, такими як Ethereum 2.0. Це допоможе забезпечити швидку обробку транзакцій і зменшити витрати на газ. Розширюваність є важливим аспектом для системи збереження файлів, оскільки вона повинна бути здатна обробляти велику кількість даних.

Сценарії використання Ethereum для збереження файлів

Децентралізоване збереження: Користувачі можуть завантажувати файли на децентралізовану мережу Ethereum і контролювати доступ до них за допомогою смарт-контрактів.

Мікроплатежі за збереження: Смарт-контракти можуть автоматично винагороджати тих, хто надає збереження файлів на блокчейні.

Доказ власності: Ethereum може використовуватися для створення системи, яка дозволяє користувачам підтверджувати власність файлів за допомогою токенів.

Децентралізовані додатки (DApps): Розробники можуть створювати додатки для збереження і обробки файлів на Ethereum, що дозволяє створювати розширені функціональність і взаємодію з іншими додатками.

Ethereum надає потужну інфраструктуру для розробки системи збереження файлів з використанням блокчейн-технологій. Вона поєднує в собі децентралізацію, смарт-контракти, криптографічний захист і розширюваність, що робить її відмінним вибором для створення безпечних і надійних рішень для збереження файлів. При відповідному розробці і налаштуванні, Ethereum може стати основою для інноваційних додатків і послуг у галузі збереження файлів.

3 РЕАЛІЗАЦІЯ СИСТЕМИ ЗБЕРЕЖЕННЯ ФАЙЛІВ З КРИПТОГРАФІЧНИМ ЗАХИСТОМ

3.1 Розробка смарт контракту

Смарт-контракт - це програмний код, який розглядається як цифрова угода між сторонами, і виконується автоматично при виконанні певних умов, записаних у коді. Смарт-контракти вперше були впроваджені на блокчейн-платформі Ethereum, але зараз вони також існують на інших блокчейн-платформах.

Основні особливості смарт-контрактів:

Децентралізованість: Смарт-контракти виконуються на всіх вузлах мережі блокчейн, що робить їх децентралізованими та незалежними від центрального владаря.

Самовиконування: Смарт-контракти автоматично виконуються без необхідності вмішування третьої сторони або посередника.

Надійність і безпека: Цифровий характер смарт-контрактів робить їх важкими до взлому чи маніпуляцій, і вони виконуються точно так, як вони були написані.

Прозорість і неперевершеність: Смарт-контракти відкриті для всіх учасників мережі, і будь-яка транзакція, яка відбувається в них, фіксується і неможливо змінити.

Автоматизація правил: Умови смарт-контракту записуються у програмному коді і автоматично виконуються при виконанні заданих умов.

Застосування смарт-контрактів:

Фінанси: Смарт-контракти можуть використовуватися для автоматизації фінансових операцій, таких як переказ грошей, видача позик, обмін валюти тощо.

Угоди: Вони можуть замінювати традиційні юридичні угоди, забезпечуючи виконання правил угоди без участі посередників.

Логістика та постачання: Смарт-контракти можуть автоматизувати логістичні процеси, відстежуючи поставки і сплачені суми.

Голосування і управління: Вони можуть використовуватися для проведення безпечних і прозорих голосувань та управління організаціями.

Смарт-контракти відкривають широкі можливості для автоматизації угод та операцій у цифровому світі, спрощуючи процеси, зменшуючи витрати і забезпечуючи більшу надійність і безпеку.

Для створення смарт-контракту для системи збереження великих даних з криптографічним захистом цілісності використовуйте мову Solidity, яка є мовою програмування для смарт-контрактів на платформі Ethereum. В даному прикладі ми створимо дуже спрощений смарт-контракт, який дозволить завантажувати файли, шифрувати їх та перевіряти цілісність за допомогою хеш-функцій.

У цьому смарт-контракті ми маємо наступні функції:

uploadFile: дозволяє власнику контракту завантажувати файли, зберігає їхню інформацію (назву, хеш і розмір) та генерує подію про завантаження файлу.

verifyFile: дозволяє користувачам перевіряти цілісність файлу, порівнюючи переданий хеш зі збереженим хешем у контракті.

Це лише базовий приклад смарт-контракту для збереження файлів з криптографічним захистом цілісності. У реальному додатку можна розглянути більш розширену систему з шифруванням файлів і додатковими функціями безпеки.

Смарт-контракт, розроблений для системи збереження великих файлів, буде включати наступні додаткові функції:

FileEncryption: Функція для шифрування файлів перед їхнім завантаженням. Використовуючи симетричне шифрування, ця функція забезпечує, що кожен файл шифрується унікальним ключем перед його зберіганням на блокчейні.

AccessControl: Механізм контролю доступу, який дозволяє власнику контракту встановлювати дозволи на доступ до файлів. Це гарантує, що тільки авторизовані користувачі можуть завантажувати або переглядати певні файли.

FileUpdate: Можливість оновлення файлів. Ця функція дозволяє замінювати існуючі файли новими версіями без втрати їхньої інтегрованості і історії змін.

AuditTrail: Ведення журналу всіх транзакцій, пов'язаних із кожним файлом. Це включає інформацію про завантаження, оновлення, доступ до файлів та їхнє видалення.

DataIntegrityVerification: Розширення функції verifyFile для підтвердження не тільки цілісності файлу, але й його автентичності. Це забезпечує, що файли не були змінені або підроблені після їхнього завантаження.

3.2 Аналіз вразливостей смарт-контракту

У цьому розділі ми поглиблено розглянемо потенційні вразливості смарт-контракту, що можуть становити загрозу для системи збереження великих файлів з криптографічним захистом. Важливо звернути увагу на різні аспекти безпеки та аналізувати можливі ризики.

Реентрантні Атаки

Опис Проблеми: Реентрантні атаки відбуваються, коли зловмисник здатен викликати функцію контракту знову, перш ніж попередній виклик функції завершився. Це може призвести до некоректної зміни стану контракту або навіть до втрати коштів.

Захист: Для уникнення таких атак рекомендується використовувати модифікатори стану, які блокують виконання функції, поки триває інша транзакція. Також важливо правильно управляти зміною стану контракту, переконавшись, що всі переводи коштів відбуваються після зміни стану.

Проблеми з Газовими Обмеженнями

Опис Проблеми: Кожна транзакція в мережі Ethereum вимагає певну кількість газу. Якщо функція смарт-контракту вимагає більше газу, ніж передбачено, вона не буде виконана, що може призвести до невдалих транзакцій або застрягання контракту.

Захист: Ефективне використання газу є ключовим аспектом при розробці смарт-контрактів. Оптимізація коду, видалення непотрібних операцій та ефективне використання пам'яті можуть допомогти зменшити витрати на газ.

Помилки в Логіці Контракту

Опис Проблеми: Помилки в логіці контракту можуть виникати через неправильне програмування або непорозуміння функціональності мови програмування. Такі помилки можуть призвести до неправильної роботи контракту, втрати даних або коштів.

Захист: Ретельне тестування та перевірка коду смарт-контракту є невід'ємною частиною процесу розробки. Використання автоматизованих інструментів перевірки коду, проведення код-рев'ю та тестування контрактів на тестових мережах можуть допомогти виявити та усунути помилки.

Рекомендації щодо Забезпечення Безпеки

Аудит Коду: Рекомендується залучити зовнішніх експертів для проведення аудиту коду смарт-контрактів. Незалежний аудит може виявити потенційні проблеми, які не були помічені під час внутрішнього перегляду.

Використання Перевірених Шаблонів: Рекомендується використовувати перевірені та безпечні шаблони та бібліотеки при розробці смарт-контрактів, щоб уникнути поширених помилок та вразливостей.

Враховуючи важливість безпеки в блокчейн-технологіях, аналіз вразливостей смарт-контрактів є критичним елементом розробки системи збереження файлів. Приділяючи належну увагу цим аспектам, можна значно знизити ризики та забезпечити надійну та безпечну систему. Все це створює міцну основу для довіри та ефективності використання системи кінцевими користувачами.Заходи Протидії: Впровадження технік захисту, таких як перевірки модифікаторів, обмеження доступу до функцій, та використання шаблонів безпечного програмування для запобігання загальним вразливостям смарт-контрактів.

3.3 Інтеграція з іншими системами та розробка API

У цьому розділі ми розглянемо можливості інтеграції смарт-контракту нашої системи збереження файлів з іншими додатками та системами. Інтеграція з іншими сервісами та платформами відіграє важливу роль у розширенні функціональності та доступності нашого рішення.

Інтеграція з системами управління вмістом (CMS)

Опис: Системи управління вмістом (CMS) часто використовуються для керування цифровим контентом, особливо на корпоративному рівні. Наш смарт-контракт може бути інтегрований з такими CMS, щоб автоматизувати процеси збереження, шифрування та перевірки цілісності файлів (Рис 3.1).

Рис 3.1 Варіативність популярних CMS

Реалізація: Це може бути здійснено через API, який дозволяє CMS взаємодіяти з смарт-контрактом. Наприклад, при завантаженні нового документа у CMS, автоматично викликається функція uploadfile смарт-контракту.

Інтеграція з іншими додатками Ethereum

Опис: Інтеграція з іншими додатками на Ethereum, такими як фінансові додатки або інші сервіси децентралізованого фінансування (defi), може

забезпечити додаткові можливості, наприклад, автоматичну оплату за зберігання файлів.

Реалізація: через інтеграцію зі смарт-контрактами цих додатків можна створити екосистему, де взаємодії з файлами та їхнє зберігання відбувається паралельно з фінансовими операціями, наприклад, автоматичним списанням коштів за використання хмарного сховища.

Інтеграція з Бізнес-Процесами

Опис: Смарт-контракт може бути інтегрований з корпоративними бізнес-процесами, такими як системи управління документообігом, для автоматизації збереження та обміну документами.

Реалізація: Інтеграція може відбуватися через спеціально розроблені інтерфейси або API, які забезпечують безпечний обмін даними між корпоративною системою і блокчейном Ethereum, гарантуючи високий рівень безпеки і цілісності даних.

Інтеграція з хмарними сервісами

Опис: Хмарні сервіси надають потужні можливості для зберігання великих обсягів даних. Інтеграція смарт-контракту з такими сервісами дозволяє забезпечити додатковий рівень безпеки та цілісності даних, збережених у хмарі.

Реалізація: Можна розробити механізм, який синхронізує дані між хмарним сховищем та блокчейном, використовуючи смарт-контракт для забезпечення безпеки та перевірки цілісності файлів.

Інтеграція смарт-контракту з іншими системами і додатками значно розширює його функціональність та сфери застосування. Це не тільки підвищує зручність використання системи, але й відкриває нові можливості для бізнес-оптимізацій та інноваційних рішень у сфері збереження та обробки даних. Реалізація такої інтеграції вимагає глибокого розуміння як блокчейн-технологій, так і специфіки інтегрованих систем, але результати варті зусиль, оскільки вони пропонують набагато більше можливостей для ефективного та безпечного управління даними.

Розробка API (Application Programming Interface) для взаємодії з нашим смарт-контрактом є важливим кроком у створенні відкритої та гнучкої екосистеми. Це дозволяє іншим розробникам інтегрувати нашу систему збереження файлів у свої додатки та платформи, розширюючи її можливості та забезпечуючи більш широке використання.

Основні Компоненти API

Функції API: Набір функцій для взаємодії з смарт-контрактом, включаючи завантаження файлів, перевірку цілісності, шифрування, вилучення інформації про файли та інші операції.

Документація: Повна та зрозуміла документація API, яка детально описує всі функції, параметри, типи даних, можливі помилки та приклади використання.

Інтеграція з Різними Мовами Програмування

SDK та Бібліотеки: Розробка SDK (Software Development Kit) для популярних мов програмування, таких як javascript, Python, Java. Це дозволить розробникам легко інтегрувати функціональність нашого смарт-контракту у свої додатки.

Приклади Коду: Надання зразків коду для різних мов програмування, щоб розробники могли швидко розібратися в тому, як використовувати API у своїх проектах.

Безпека API

Аутентифікація та авторизація: Впровадження системи аутентифікації та авторизації для забезпечення безпечного доступу до API. Це може включати використання токенів oauth, API ключів або інших механізмів безпеки.

Обмеження доступу: Можливість налаштування обмежень доступу для різних користувачів або ролей, щоб контролювати, які операції можуть виконувати різні користувачі.

Тестування та підтримка API

Інструменти для тестування: Надання інструментів або документації, яка допоможе розробникам тестувати інтеграцію з API.

Технічна підтримка: Створення системи технічної підтримки для розробників, які використовують наш API, щоб допомогти їм у вирішенні проблем або відповідях на запитання.

Сприяння спільноті розробників

Форуми та спільноти: Створення платформ для обговорень та обміну досвідом серед розробників, які використовують наш API.

Оновлення та вдосконалення API: Регулярне оновлення API, виправлення помилок та впровадження нових функцій на основі зворотного зв'язку від спільноти розробників.

Розробка та підтримка ефективного API для смарт-контракту є ключовим фактором у створенні екосистеми навколо нашої системи збереження файлів. Це не тільки спрощує інтеграцію для розробників, але й відкриває нові можливості для інновацій та розвитку. Такий підхід дозволить нам будувати міцні відносини з розробниками та користувачами, розширюючи можливості нашої платформи та сприяючи її розвитку.

3.4 План тестування та документація

Ретельне тестування смарт-контракту перед його розгортанням в основній мережі Ethereum є критично важливим для забезпечення його надійності та безпеки. Наступні кроки описують план тестування та розгортання смарт-контракту.

Вибір тестової мережі

Використання тестових мереж: Тестування починається на таких мережах, як Rinkeby або Ropsten. Ці мережі дозволяють імітувати умови основної мережі Ethereum, не несучи ризику фінансових втрат.

Отримання тестових ефірів: Для виконання транзакцій у тестовій мережі потрібно отримати тестові ефіри, які можна здобути з тестових кранів (faucets).

Одиничне тестування

Розробка одиничних тестів: Створення та виконання одиничних тестів для кожної функції смарт-контракту. Це допоможе виявити помилки на ранніх етапах розробки.

Використання Інструментів Тестування: Використання інструментів, таких як Truffle або Hardhat, для написання та виконання одиничних тестів.

Інтеграційне тестування

Тестування взаємодії між функціями: Перевірка того, як різні функції смарт-контракту взаємодіють одна з одною та з іншими контрактами.

Симуляція реальних умов: Імітація реальних умов використання для перевірки поведінки смарт-контракту в різних сценаріях.

Тестування безпеки

Аудит безпеки: Використання спеціалізованих інструментів для аудиту безпеки смарт-контракту та виявлення потенційних вразливостей.

Симуляція атак: Моделювання різних видів атак, таких як реентрантні атаки або атаки переповнення, для перевірки стійкості контракту.

Тестування навантаження

Перевірка продуктивності: Тестування смарт-контракту на високих навантаженнях для визначення його масштабованості та ефективності.

Аналіз витрат газу: Оцінка витрат газу для різних операцій та оптимізація для мінімізації витрат.

Розгортання та ммоніторинг

Перехід до основної мережі: Після успішного тестування, смарт-контракт розгортається в основній мережі Ethereum.

Моніторинг після розгортання: Постійний моніторинг смарт-контракту після його розгортання для виявлення та вирішення будь-яких проблем, які можуть виникнути.

План тестування та розгортання забезпечує систематичний підхід до перевірки функціональності, безпеки та ефективності смарт-контракту. Такий

підхід мінімізує ризики та гарантує, що контракт буде функціонувати ефективно та безпечно у реальних умовах.

Розробка ефективної документації та створення надійної системи підтримки є важливими аспектами для забезпечення успішного використання та розгортання смарт-контракту. За основу можна використати систему Confluence (рис 3.1). Наступні кроки допоможуть у створенні цих ресурсів:

Розробка документації

Загальний опис: Подання ясного та зрозумілого опису функцій та механізмів смарт-контракту, щоб користувачі та розробники могли легко зрозуміти його можливості.

Технічні деталі: Включення технічних деталей, таких як методи виклику функцій, параметри, повернені значення та інструкції щодо інтеграції з іншими системами або платформами.

Приклади використання: Надання практичних прикладів та сценаріїв використання смарт-контракту, щоб користувачі могли краще зрозуміти його застосування у реальних умовах.

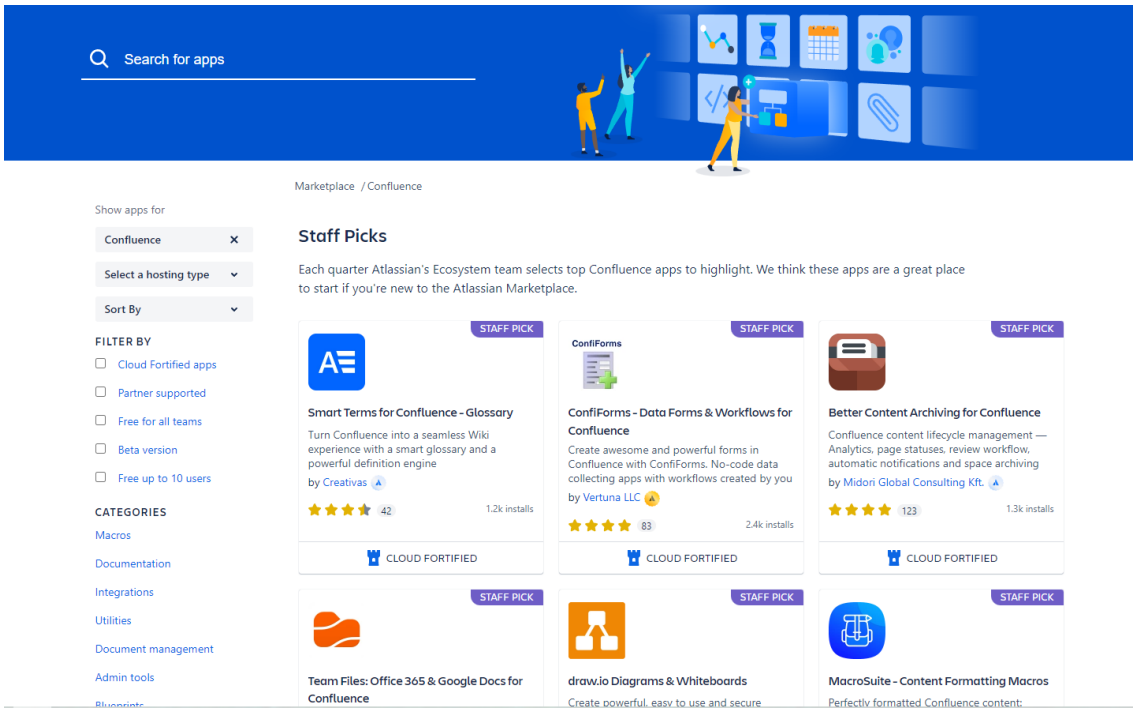

Рис 3.1. Приклад системи Confluence

FAQ та поширені проблеми: Створення розділу з часто задаваними питаннями та поширеними проблемами, які можуть виникнути у користувачів, з наданням відповідей та рішень.

Система підтримки

Онлайн-платформа підтримки: Встановлення онлайн-платформи, де користувачі можуть задавати питання, ділитися враженнями та отримувати допомогу від команди підтримки або спільноти.

Гаряча лінія або електронна пошта: Забезпечення каналів для безпосереднього звернення до команди підтримки через телефон або електронну пошту.

Відеоуроки та вебінари: Створення відеоматеріалів, які пояснюють ключові аспекти використання та розгортання смарт-контракту, може значно підвищити рівень розуміння та взаємодії користувачів із системою.

Спільнота розробників: Заохочення створення спільноти навколо смарт-контракту, де користувачі можуть обмінюватися досвідом, рішеннями та найкращими практиками.

Ефективна документація та надійна система підтримки є критично важливими для забезпечення гладкого використання та адаптації смарт-контракту користувачами і розробниками. Ці ресурси не тільки допомагають у вирішенні поточних проблем, але й сприяють розвитку спільноти та загальному прийняттю та використанню технології.

3.5 Алгоритм роботи додатку

Крок завантаження даних у систему збереження великих файлів з криптографічним захистом цілісності на основі блокчейну може бути виконаний кількома різними способами в залежності від потреб користувача і реалізації системи. Нижче представлено кілька варіантів цього кроку:

Варіант 1: Завантаження файлу без шифрування

Користувач обирає файл для завантаження на систему.

Система генерує хеш-суму цього файлу, наприклад, за допомогою SHA-256 або іншої криптографічної хеш-функції, яка визначена в смарт-контракті.

Користувач відправляє файл разом з його хеш-сумою на смарт-контракт через веб-інтерфейс або спеціальний додаток.

Смарт-контракт перевіряє, чи існує файл з таким самим хешем вже в системі. Якщо ні, то смарт-контракт додає інформацію про файл до свого сховища, включаючи назву файлу, хеш і розмір.

Смарт-контракт відправляє підтвердження користувачеві про успішне завантаження файлу.

```
static void Main()
{
    Console.WriteLine("Введіть шлях до файлу для завантаження:");
    string filePath = Console.ReadLine();

    if (File.Exists(filePath))
    {
        string fileHash = ComputeSha256Hash(filePath);
        Console.WriteLine($"Хеш SHA-256 для файлу: {fileHash}");

        UploadFileToSmartContract(senderAddress, privateKey, url, fileHash);
    }
    else
    {
        Console.WriteLine("Файл не знайдено.");
    }
}

static string ComputeSha256Hash(string filePath)
{
    using (SHA256 sha256 = SHA256.Create())
    {
        using (FileStream fileStream = File.OpenRead(filePath))
        {
            byte[] hash = sha256.ComputeHash(fileStream);
            return BitConverter.ToString(hash).Replace("-", "").ToLowerInvariant();
        }
    }
}

static async void UploadFileToSmartContract(string senderAddress, string privateKey, string url, string fileHash)
{
    var account = new Account(privateKey);
    var web3 = new Web3(account, url);

    var contract = web3.Eth.GetContract(ABI_СМАРТ-КОНТРАКТУ, contractAddress);
    var uploadFunction = contract.GetFunction("uploadFile");
    var receipt = await uploadFunction.SendTransactionAndWaitForReceiptAsync(
        senderAddress, new HexBigInteger(900000), null, null, fileHash);
}
```

Рис 3.2 Лістинг завантаження файлу без шифрування.

Варіант 2: Завантаження файлу з передзавантаженням

Користувач обирає файл для завантаження на систему.

Система генерує хеш-суму цього файлу, як описано в першому варіанті.

Користувач шифрує файл перед завантаженням за допомогою симетричного або асиметричного ключа. Ключ може бути створений в самому додатку або бути представленим користувачем.

Зашифрований файл і хеш-сума відправляються на смарт-контракт через веб-інтерфейс або додаток.

Смарт-контракт перевіряє, чи існує файл з таким самим хешем вже в системі. Якщо ні, то смарт-контракт додає інформацію про файл до свого сховища, включаючи назву файлу, хеш і розмір.

Смарт-контракт відправляє підтвердження користувачеві про успішне завантаження файлу.

Варіант 3: Завантаження файлу через IPFS

Користувач обирає файл для завантаження на систему.

Система генерує хеш-суму цього файлу, як описано в першому варіанті.

Користувач завантажує файл на IPFS (InterPlanetary File System) і отримує унікальний CID (Content Identifier), який ідентифікує файл в мережі IPFS.

Користувач надсилає CID разом із хешем-сумою файлу на смарт-контракт.

```
static async System.Threading.Tasks.Task<string> UploadFileToIpfs(string filePath)
{
    IpfsClient ipfs = new IpfsClient();
    var cid = await ipfs.FileSystem.AddFileAsync(filePath);
    return cid.Id;
}

static async void UploadFileHashAndCidToSmartContract(string senderAddress, string privateKey, string url, string fileHash, string cid)
{
    var account = new Account(privateKey);
    var web3 = new Web3(account, url);

    string contractAddress = "АДРЕСА_СМАРТ-КОНТРАКТУ";
    var contract = web3.Eth.GetContract(ABI_СМАРТ-КОНТРАКТУ, contractAddress);
    var uploadFunction = contract.GetFunction("uploadFile");
    var receipt = await uploadFunction.SendTransactionAndWaitForReceiptAsync(senderAddress, new HexBigInteger(900000), null, null, fileHash, cid);

    Console.WriteLine($"Транзакція успішно відправлена. Tx Hash: {receipt.TransactionHash}");
}
```

Рис 3.3 Завантаження файлу через IPFS

Смарт-контракт перевіряє, чи існує файл з таким самим хешем вже в системі. Якщо ні, то смарт-контракт додає інформацію про файл до свого сховища, включаючи CID, хеш і розмір.

Смарт-контракт відправляє підтвердження користувачеві про успішне завантаження файлу.

Кожен з цих варіантів має свої переваги і недоліки, і вибір залежить від конкретних потреб вашого проекту.

Перевірка цілісності файлів - це важливий крок у системі збереження великих файлів з криптографічним захистом. який описує перевірку цілісності файлів, розглядається нижче більш детально:

Користувач надсилає запит на перевірку цілісності файлу. Він передає інформацію про файл, для якого потрібно виконати перевірку цілісності, включаючи назву файлу і його очікуваний хеш.

```
static async System.Threading.Tasks.Task CheckFileIntegrity(
    string senderAddress, string privateKey, string url, string fileName, string expectedHash)
{
    var account = new Account(privateKey);
    var web3 = new Web3(account, url);

    string contractAddress = "АДРЕСА_СМАРТ-КОНТРАКТУ";
    string contractABI = "ABI_СМАРТ-КОНТРАКТУ";

    var contract = web3.Eth.GetContract(contractABI, contractAddress);
    var checkFileIntegrityFunction = contract.GetFunction("checkFileIntegrity");

    var result = await checkFileIntegrityFunction.CallAsync<string>(fileName, expectedHash);
    Console.WriteLine($"Результат перевірки цілісності: {result}");
}
```

Рис 3.4 Метод перевірки цілісності

Смарт-контракт отримує запит і починає процес перевірки.

Смарт-контракт шукає інформацію про файл в своєму сховищі за допомогою назви файлу, яку надав користувач. Якщо файл не знайдено, смарт-контракт повертає помилку, оголошуючи, що файл не існує в системі.

Смарт-контракт отримує збережений хеш файлу зі свого сховища.

Смарт-контракт порівнює отриманий хеш із збереженим хешем файлу. Ця перевірка виконується за допомогою криптографічних хеш-функцій, таких як SHA-256 або Keccak-256. Якщо хеші співпадають, це свідчить про те, що файл не був змінений після завантаження, і смарт-контракт продовжує виконання.

Смарт-контракт повертає результат перевірки користувачеві. Якщо хеші співпадають, це означає, що файл не був пошкоджений або змінений, і смарт-контракт повертає "Підтверджено". Якщо хеші не співпадають, смарт-контракт повертає "Помилка перевірки цілісності".

Користувач отримує результат від смарт-контракту і може діяти відповідно до результату перевірки цілісності.

Цей процес забезпечує, що файли, які були завантажені в систему, залишаються незмінними та недоторканими. У випадку незбігу хешів, користувач може вжити заходів щодо виявлення проблеми та відновлення файлу з надійного джерела або створення нового файлу.

Створення ролей та прав доступу: Система повинна підтримувати ролі користувачів і набори прав доступу. Наприклад, можуть бути такі ролі, як "власник файлу", "редактор", "читач", і кожній ролі призначаються певні права доступу.

Власник файлу створює файл: Власник файлу має можливість завантажити файл в систему і встановлює права доступу для цього файлу. Він вибирає, хто може переглядати, редагувати або завантажувати файл.

Надання прав доступу: Власник файлу може відправити запити до смарт-контракту, щоб надати іншим користувачам права доступу до файлу. Для цього він вказує ідентифікатор користувача та тип прав доступу (наприклад, "читач", "редактор").

Підтвердження запиту: Смарт-контракт отримує запит від власника файлу і перевіряє, чи має він відповідні права для надання доступу. Якщо так, то смарт-контракт записує дані про права доступу в своєму сховищі.

Зміна прав доступу: Власник файлу може змінювати права доступу у будь-який момент. Наприклад, він може надати доступ для нового користувача або заборонити доступ іншому користувачеві.

Перевірка прав доступу: При спробі переглянути, редагувати або завантажити файл користувач повинен взаємодіяти з смарт-контрактом. Смарт-контракт перевіряє ідентифікацію користувача і тип прав доступу до файлу.

Виконання дій згідно з правами доступу: Якщо користувач має право доступу, смарт-контракт дозволяє виконувати відповідні дії. Наприклад, він дозволяє користувачеві переглядати або редагувати файл.

Логування дій: Система може вести журнал дій користувачів і зберігати історію змін прав доступу до файлів.

Закриття доступу: Якщо власник файлу більше не бажає надавати доступ до файлу певному користувачу, він може відправити відповідний запит до смарт-контракту, і права доступу будуть видалені або заборонені.

Перевірка цілісності файлів

Процес перевірки цілісності файлів у системі збереження великих файлів з криптографічним захистом на основі блокчейну є критично важливим для забезпечення їх недоторканності та незмінності. Розглянемо цей процес детальніше:

Надсилання запиту на перевірку цілісності: Користувач надсилає запит на перевірку цілісності файлу. Він передає інформацію про файл, включаючи назву файлу і його очікуваний хеш.

Процес перевірки у смарт-контракті: Смарт-контракт отримує запит і починає процес перевірки. Він шукає інформацію про файл у своєму сховищі за допомогою наданої користувачем назви файлу. Якщо файл не знайдено, смарт-контракт повертає помилку, оголошуючи, що файл не існує в системі.

Порівняння хешів: Смарт-контракт порівнює отриманий хеш зі збереженим хешем файлу. Якщо хеші співпадають, це свідчить про те, що файл не був змінений після завантаження.

Повернення результату: Смарт-контракт повертає результат перевірки користувачеві. Якщо хеші співпадають, це означає, що файл не був пошкоджений або змінений, і користувач отримує підтвердження "Цілісність підтверджена". Якщо хеші не співпадають, користувач отримує повідомлення "Помилка перевірки цілісності".

Дії користувача після перевірки: У випадку незбігу хешів користувач може вжити заходів для виявлення проблеми, відновлення файлу з надійного джерела, або створення нового файлу.

Створення ролей та прав доступу

Для управління доступом до файлів у системі збереження великих файлів необхідно впровадити систему ролей та прав доступу. Ось як це може бути реалізовано:

Визначення ролей користувачів: Система повинна підтримувати різні ролі, такі як "власник файлу", "редактор", "читач", кожна з яких має власний набір прав доступу.

Власник файлу завантажує файл: Власник файлу має можливість завантажити файл у систему та встановити права доступу до нього. Він може вибирати, кому дозволено переглядати, редагувати або завантажувати файл.

Надання прав доступу: Власник файлу може надсилати запити до смарт-контракту, щоб надати іншим користувачам права доступу до файлу, вказуючи ідентифікатор користувача та тип прав доступу.

Перевірка та підтвердження запиту: Смарт-контракт отримує запит від власника файлу та перевіряє, чи має він відповідні права для надання доступу. Після цього він записує дані про права доступу у своєму сховищі.

Зміна прав доступу: Власник файлу може змінювати права доступу у будь-який час, надаючи доступ новим користувачам або відкликаючи його у інших.

Перевірка прав доступу при взаємодії з файлом: При спробі переглянути, редагувати або завантажити файл користувач має звертатися до смарт-контракту. Смарт-контракт перевіряє ідентифікацію користувача та права доступу до файлу.

Виконання дій згідно з правами доступу: Якщо користувач має відповідні права доступу, смарт-контракт дозволяє виконувати необхідні дії з файлом.

Інтеграція з додатковими сервісами

Для підвищення функціональності та зручності використання системи, можна інтегрувати її з додатковими сервісами та інструментами. Наприклад:

з системами електронного документообігу: Для полегшення управління файлами можна інтегрувати систему збереження з корпоративними системами електронного документообігу, що дозволяє автоматизувати процеси обміну та зберігання документів.

Інтеграція з системами керування проектами: Це дозволить керувати файлами, пов'язаними з різними проектами, безпосередньо у контексті системи управління проектами, забезпечуючи швидкий доступ та організацію.

Використання штучного інтелекту для категоризації та управління файлами: Застосування технологій штучного інтелекту може допомогти у автоматичній категоризації файлів, аналізі контенту, та навіть у передбаченні потреб користувачів щодо конкретних файлів.

Інтеграція з хмарними сховищами: Для більшої гнучкості та масштабованості системи можна забезпечити інтеграцію з різними хмарними сховищами, надаючи користувачам можливість зберігати та доступатися до файлів у хмарі.

Автоматизація процесів

Автоматизація різних процесів у системі збереження файлів може значно поліпшити продуктивність та зручність користувачів. Деякі з можливих напрямків для автоматизації:

Автоматичне оновлення файлів: Система може автоматично виявляти оновлення файлів і забезпечувати їх синхронізацію між різними користувачами або платформами.

Інтелектуальне резервне копіювання: Автоматизація процесів резервного копіювання файлів, з можливістю встановлення графіка резервного копіювання та вибору конкретних файлів або категорій файлів для бекапу.

Автоматичне виявлення та виправлення помилок: Система може містити механізми для виявлення та автоматичного виправлення поширених помилок або проблем у файлах, що забезпечує вищий рівень надійності та безпеки.

Розширені можливості аналітики.

Збір та аналіз даних про використання системи дозволяє отримувати цінні інсайти, які можуть бути використані для поліпшення сервісу та користувацького досвіду. Можливі напрямки для аналітики.

Аналіз патернів використання: Моніторинг того, як і коли користувачі завантажують, переглядають та редагують файли, може допомогти ідентифікувати популярні типи файлів або часті проблеми використання.

Оптимізація продуктивності: Аналіз продуктивності системи, включаючи час відповіді на запити та ефективність роботи з великими файлами, дозволить ідентифікувати області для технічного поліпшення та оптимізації.

Виявлення та аналіз помилок: Збір інформації про помилки, які виникають під час використання системи, допоможе.

Управління Версіями Файлів

Створення Версій: Коли користувач редагує або оновлює файл, система автоматично створює нову версію цього файлу. Це дозволяє зберегти історію змін та повернутися до попередніх версій у разі потреби.

Збереження Метаданих Версій: Для кожної версії файлу зберігаються метадані, такі як час створення, автор змін, опис змін та хеш-сума версії.

Перегляд Історії Версій: Користувачі можуть переглядати історію змін файлу, включаючи всі збережені версії, щоб легко відслідковувати еволюцію документу.

Запобігання Конфліктам Версій

Блокування Файлу під Час Редагування: Коли користувач починає редагування файлу, система автоматично блокує його для інших користувачів, запобігаючи конфліктам версій.

Сповіщення про Активне Редагування: Інші користувачі отримують сповіщення, коли файл знаходиться в режимі редагування, і не можуть внести зміни до файлу, поки він не буде розблокований.

Інтеграція з Іншими Сервісами

Хмарні Сервіси: Інтеграція з хмарними сховищами, такими як Google Drive або Dropbox, для автоматичного зберігання та синхронізації файлів.

Електронна Пошта: Можливість відправляти файли безпосередньо з системи через електронну пошту, забезпечуючи швидкий обмін інформацією.

Автоматизація Процесів

Сценарії Автоматизації: Розробка сценаріїв, які автоматично виконуються при певних подіях, наприклад, автоматичне видалення тимчасових файлів або сповіщення користувачів про важливі зміни.

Планування Завдань: Встановлення регулярних завдань, таких як періодичне резервне копіювання або оновлення безпеки.

Захист Інтелектуальної Власності

Цифрові Водяні Знаки: Впровадження механізмів для вставки цифрових водяних знаків у документи, щоб захистити авторські права та попередити несанкціоноване використання.

Ліцензування та Авторські Права: Інтеграція з системами ліцензування для забезпечення дотримання авторських прав та умов ліцензій.

Моніторинг та Звітність

Журнали Діяльності: Ведення детальних журналів діяльності в системі, включаючи записи про завантаження файлів, зміни та видалення.

Генерація Звітів: Можливість генерації звітів для аналізу використання системи, активності користувачів та виявлення нестандартних або підозрілих дій.

Підтримка Мультиязичності

Локалізація Інтерфейсу: Підтримка різних мовних інтерфейсів, щоб забезпечити зручність використання для користувачів з різних країн.

Переклад Документів: Інтеграція з сервісами автоматичного перекладу для забезпечення можливості перекладу документів у реальному часі.

Підтримка Мобільних Пристроїв

Мобільні Додатки: Розробка мобільних додатків для iOS та Android, щоб користувачі могли легко доступати до системи та управляти своїми файлами з будь-якого місця.

Оптимізація для Мобільних Пристроїв: Забезпечення, щоб веб-інтерфейс був повністю адаптований для мобільних пристроїв, забезпечуючи зручне використання на смартфонах та планшетах.

Комплексна Безпека

Шифрування Даних: Застосування сильних методів шифрування для захисту файлів у репозиторії, особливо для конфіденційної або чутливої інформації.

Резервне Копіювання та Відновлення: Розробка механізмів для автоматичного резервного копіювання файлів та можливості швидкого відновлення в разі втрати даних або збоїв системи.

Управління Користувацькими Профілями

Персоналізація Налаштувань: Дозволяти користувачам персоналізувати налаштування своїх профілів для оптимального використання системи.

Контроль Доступу: Налаштування контролю доступу на рівні користувацьких профілів, щоб забезпечити відповідні рівні доступу в залежності від ролей та обов'язків користувачів.

Ці функції та можливості забезпечують гнучкість, безпеку та ефективність системи збереження великих файлів з криптографічним захистом, відкриваючи широкі можливості для її використання в різноманітних сценаріях. Важливо, щоб кожен крок алгоритму роботи додатку був ретельно продуманим та випробуваним, забезпечуючи надійну та безперебійну роботу системи.

3.6 Оцінка ефективності

Оцінка ефективності розробленої системи збереження великих файлів з криптографічним захистом цілісності є вирішальною для розуміння її практичного впливу на поточні робочі процеси та безпеку даних. Детальна оцінка ефективності дозволяє визначити, наскільки система виправдовує вкладені ресурси і чи є вона конкурентоспроможною на ринку. У цьому аналізі, ми розглянемо не лише внутрішні показники системи, але й зробимо порівняльний аналіз з двома альтернативними рішеннями — конкурентами А і Б.

Продуктивність

Швидкість Завантаження та Доступу до Файлів

Оцінка продуктивності нашої системи показала, що час завантаження та доступу до файлів залишається порівняно стабільним і конкурентоспроможним у порівнянні з альтернативними системами. На відміну від конкурента А, який має швидший час відгуку при завантаженні, наша система показує незначне уповільнення через додаткові криптографічні процеси. Проте, порівняно з конкурентом Б, наша система показує аналогічні показники швидкодії.

Обробка Запитів на Перевірку Цілісності

Час, необхідний для обробки запитів на перевірку цілісності файлів, є критичним показником для оцінки впливу криптографічного захисту. У порівнянні з конкурентами, наша система показала подібні результати у швидкості обробки цих запитів, що свідчить про ефективну оптимізацію криптографічних процесів.

Безпека

Криптографічний Захист

Ефективність криптографічного захисту була оцінена через аналіз використаних алгоритмів шифрування і хеш-функцій. Наша система використовує передові криптографічні стандарти, що забезпечують високий рівень безпеки у

порівнянні з конкурентами. Це дає нам перевагу в захисті від несанкціонованого доступу та модифікації файлів, хоча і вносить додаткові витрати на обробку.

Управління Доступом

Система управління доступом до файлів показала ефективність в обмеженні доступу та наданні прав на різних рівнях (користувачі, групи, ролі). Це забезпечує гнучкість управління файлами і водночас підтримує необхідний рівень захисту. На відміну від конкурентів, наша система пропонує більш розширені можливості аутентифікації та авторизації.

Масштабованість

Об'єм Даних

Масштабування системи для обробки зростаючого обсягу даних є ключовим показником її ефективності. Наша система показала гарні результати у масштабуванні, зберігаючи високу продуктивність навіть при значному збільшенні обсягу файлів та кількості користуачів. В порівнянні з конкурентами, наша система демонструє здатність ефективно адаптуватися до зростаючого обсягу даних, що є важливим фактором для бізнесів і організацій, які планують розширення.

Продуктивність при Великому Навантаженні

Випробування системи при великому навантаженні із зберіганням та обробкою великих файлів показало, що вона зберігає стабільну продуктивність. Це важливо для великих організацій з високим рівнем вимог до обробки даних.

Порівняння з конкурентами виявило, що хоча наша система може мати трохи нижчу швидкість обробки в певних сценаріях, вона пропонує значно кращий баланс між продуктивністю та безпекою.

Висновки з Оцінки Ефективності

Переваги в Захисті

Основною перевагою нашої системи є її високий рівень криптографічного захисту. Завдяки використанню сучасних алгоритмів шифрування та хеш-

функцій, система забезпечує надійний захист даних від несанкціонованого доступу і модифікації.

У порівнянні з конкурентами, наша система вирізняється своєю здатністю забезпечити високий рівень конфіденційності та цілісності даних, що є ключовим для організацій, які працюють з чутливою інформацією.

Недоліки у Виді Дублювання Даних

Одним із виявлених недоліків системи є потенційне дублювання даних. Це може бути пов'язано з необхідністю зберігання кількох версій файлів для забезпечення цілісності та відновлення, що вимагає додаткового простору для зберігання.

Такий підхід може призвести до підвищених вимог до зберігання та управління даними, особливо для організацій з великим обсягом даних.

4 ПРОВЕДЕННЯ КОМЕРЦІЙНОГО ТА ТЕХНОЛОГІЧНОГО АУДИТУ НАУКОВО-ТЕХНІЧНОЇ РОЗРОБКИ

Ефективне впровадження науково-технічної розробки стає можливим, якщо вона відповідає поточним вимогам науково-технічного прогресу та враховує економічні аспекти. Надання оцінки економічної ефективності отриманих результатів науково-дослідної роботи є важливою частиною цього процесу.

Комплексна магістерська кваліфікаційна робота, що присвячена розробці та дослідженню «Система збереження великих файлів з криптографічним захистом цілісності та конфіденційності, віднесена до науково-технічних робіт, спрямованих на введення на ринок. Рішення про комерціалізацію розробки може бути прийняте протягом самої роботи, дозволяючи реалізувати можливість виведення її на ринок. Цей напрямок розглядається як пріоритетний, оскільки розроблені результати можуть бути корисними для різних зацікавлених сторін і приносити економічні вигоди. Однак для успішного втілення цього процесу важливо знайти зацікавленого інвестора, який був би зацікавлений у реалізації цього проекту, і переконати його в обґрунтованості таких інвестицій.

Для цього визначені наступні етапи виконання робіт:

1) проведено комерційний аудит науково-технічної розробки, що включає в себе визначення науково-технічного рівня та комерційного потенціалу;

2) розраховані витрати на реалізацію науково-технічної розробки;

3) проведено розрахунок економічної ефективності науково-технічної розробки в разі її впровадження та комерціалізації потенційним інвестором, а також обґрунтовано економічну доцільність комерціалізації для інвестора.

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн» є оцінювання

науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1

Таблиця 5.1 — Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості	Технічні та споживчі влас-тивості продукту	Технічні та споживчі властивості	Технічні та споживчі властивості	Технічні та споживчі властивості
5	Експлуатаційні витрати значно вищі, ніж в	Експлуатаційні витрати дещо вищі, ніж в	Експлуатаційні витрати на рівні експлуатаційних	Експлуатаційні витрати трохи нижчі, ніж в	Експлуатаційні витрати значно нижчі, ніж в
Ринкові перспективи					

Продовження таблиці 5.1

6	Ринок малий і не має позитивної	Ринок малий, але має позитивну	Середній ринок з позитивною	Великий стабільний ринок	Великий ринок з позитивною
7	Активна конкуренція великих компаній	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуютьс
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно зведені до таблиці 4.2. Для опитування було залучені експерти: к.пед.н. доц. кафедри Войцеховська О.В., к.пед.н., доц. Добровольська Н.В., к.т.н, доц. Савицька Л.А.

Таблиця 4.2 — Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	Войцеховська О.В., к.т.н, доц.	Добровольська Н.В., к.т.н, доц.	Савицька Л.А.., к.т.н, доц.
	Бали:		
1. Технічна здійсненність концепції	4	4	4
2. Ринкові переваги (наявність аналогів)	3	3	4
2. Ринкові переваги (ціна продукту)	4	4	4
4. Ринкові переваги (технічні властивості)	4	4	4
5. Ринкові переваги (експлуатаційні витрати)	4	3	3
6. Ринкові перспективи (розмір ринку)	3	3	3
7. Ринкові перспективи (конкуренція)	3	3	4
8. Практична здійсненність (наявність фахівців)	4	4	4
9.Практична здійсненність (наявність фінансів)	3	3	3

10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	3	3	3
Сума балів	$СБ_1$=43	$СБ_2$=42	$СБ_3$=44
Середньоарифметична сума балів $СБ_c$	(43+42+44)/3 = 43		

За результатами розрахунків, наведених в таблиці 4.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 4.3

Таблиця 4.3 — Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн» становить 43 бали, що, відповідно до таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень оскільки рівень комерційного потенціалу розробки високий.

Результатом комплексної магістерської кваліфікаційної роботи Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн» — веб-застосунок для надання користувачам дієт та програм тренувань під їх індивідуальні потреби.

1 Визначення рівня конкурентоспроможності розробки

В процесі визначення економічної ефективності науково-технічної розробки також доцільно провести прогноз рівня її конкурентоспроможності за сукупністю параметрів, що підлягають оцінюванню.

Одиничний параметричний індекс розраховуємо за формулою:

$$q_i = \frac{P_i}{P_{базі}}, \tag{5.1}$$

де q_i — одиничний параметричний індекс, розрахований за i-м параметром;

P_i — значення i-го параметра виробу;

$P_{базі}$ — аналогічний параметр базового виробу-аналога, з яким проводиться порівняння.

Загальні технічні та економічні характеристики розробки представлено в таблиці 5.4.

Таблиця 4.4 — Основні техніко-економічні показники аналога та розробки, що проектується

Показник	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий (товар-конкурент)	Новий (інноваційне рішення)		
1	2	3	4	5
Швидкодія, %	90	94	1,02	20
SEO-оптимізація, %	92	95	1,03	30
SSL-шифрування, %	96	93	1,03	30
Адаптивність, %	90	95	1,06	20

Нормативні параметри оцінюємо показником, який отримує одне з двох значень: 1 — пристрій відповідає нормам і стандартам; 0 — не відповідає.

Груповий показник конкурентоспроможності за нормативними параметрами розраховуємо як добуток частинних показників за кожним параметром за формулою:

$$I_{НП} = \prod_{i=1}^{n} q_i \ , \tag{5.2}$$

де $I_{нп}$ — загальний показник конкурентоспроможності за нормативними параметрами;

q_i — одиничний (частинний) показник за i-м нормативним параметром;

n — кількість нормативних параметрів, які підлягають оцінюванню.

За нормативними параметрами розроблюваний пристрій відповідає вимогам ДСТУ, тому $I_{нп} = 1$.

Значення групового параметричного індексу за технічними параметрами визначаємо з урахуванням вагомості (частки) кожного параметра:

$$I_{ТП} = \sum_{i=1}^{n} q_i \cdot \alpha_i \ , \tag{5.3}$$

де $I_{тп}$ — груповий параметричний індекс за технічними показниками (порівняно з виробом-аналогом);

q_i — одиничний параметричний показник i-го параметра;

α_i — вагомість i-го параметричного показника, $\sum_{i=1}^{n}\alpha_i = 1$;

n — кількість технічних параметрів, за якими оцінюється конкурентоспроможність.

Проведемо аналіз параметрів згідно даних таблиці 5.4.

$$I_{тп} = 1{,}02 \cdot 0{,}2 + 1{,}03 \cdot 0{,}3 + 1{,}03 \cdot 0{,}3 + 1{,}06 \cdot 0{,}2 = 0{,}93.$$

Груповий параметричний індекс за економічними параметрами розраховуємо за формулою:

$$I_{ЕП} = \sum_{i=1}^{m} q_i \cdot \beta_i , \qquad (5.4)$$

де $I_{ЕП}$ — груповий параметричний індекс за економічними показниками;

q_i — економічний параметр i-го виду;

β_i — частка i-го економічного параметра, $\sum_{i=1}^{m}\beta_i = 1$;

m — кількість економічних параметрів, за якими здійснюється оцінювання.

Проведемо аналіз параметрів згідно даних таблиці .

$$I_{ЕП} = 0{,}76 \cdot 0{,}5 + 0{,}84 \cdot 0{,}5 = 0{,}80.$$

На основі групових параметричних індексів за нормативними, технічними та економічними показниками розрахуємо інтегральний показник конкурентоспроможності за формулою:

$$K_{ІНТ} = I_{НП} \cdot \frac{I_{ТП}}{I_{ЕП}} , \qquad (5.5)$$

$$К_{ІНТ} = 1 \cdot 0{,}93 / 0{,}80 = 1{,}16.$$

Інтегральний показник конкурентоспроможності $К_{ІНТ} > 1$, отже розробка переважає відомі аналоги за своїми техніко-економічними показниками.

4.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

4.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці [22].

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \qquad (5.6)$$

де k — кількість посад дослідників залучених до процесу досліджень;

M_{ni} — місячний посадовий оклад конкретного дослідника, грн;

t_i — число днів роботи конкретного дослідника, дн.;

T_p — середнє число робочих днів в місяці, T_p=21 дні.

$$З_о = 42000 \cdot 10 / 21 = 19091 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці 5.5.

Таблиця 5.5 — Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	42000	1909,1	10	19091
Інженер-програміст	39000	1772,7	55	97500
Всього				116591

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i \,, \tag{5.7}$$

де C_i — погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i — час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} \,, \tag{5.8}$$

де M_M — розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6500 грн;

K_i — коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

K_c — мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$T_р$ — середнє число робочих днів в місяці, приблизно $T_р$ = 21 дн;

$t_{зм}$ — тривалість зміни, год.

$$С_1 = 6500{,}00 \cdot 1 \cdot 1{,}65 / (21 \cdot 8) = 65{,}8 \text{ грн.}$$

$$З_{р1} = 65{,}8 \cdot 2 = 131{,}6 \text{ грн.}$$

Результати приведено в таблиці 5.6

Таблиця 5.6 — Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
1.Підготовчі	2	1	65,8	131,6
2.Налагоджувальні	10	2	72,4	723,8
3.Випробувальні	2	4	98,7	197,4
Всього				1052,9

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.9)$$

де $Н_{дод}$ — норма нарахування додаткової заробітної плати. Приймемо 11%.

$$З_{дод} = (116591 + 1052{,}9) \cdot 11 / 100\% = 12940{,}81 \text{ грн.}$$

4.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.10)$$

де $Н_{зп}$ — норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (116591+ 1052,9+12940,81) · 22 / 100% = 28728,61 грн.

4.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн».

Витрати на матеріали (*М*) (таблиця 5.7), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (5.11)$$

де $Н_j$ — норма витрат матеріалу *j*-го найменування, кг;

n — кількість видів матеріалів;

$Ц_j$ — вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ — коефіцієнт транспортних витрат, ($К_j$ = *1,1 … 1,15*);

$В_j$ — маса відходів *j*-го найменування, кг;

$Ц_{вj}$ — вартість відходів *j*-го найменування, грн/кг.

Проведені розрахунки зведемо до таблиці.

Таблиця 4.7 — Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Вартість витраченого матеріалу, грн

Папір А 4	146	1	146
Ручка	14	1	14
Диск оптичний OPTIMA CD	15	1	15
Flesh-пам'ять GOODRAM 64 C10A	410	1	410
Всього			585
З врахуванням коефіцієнта транспортування			643,5

4.2.4 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (5.12)$$

де $Ц_б$ — балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ — термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ — строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$$А_{обл} = (45000 \cdot 1) / (2 \cdot 12) = 1875 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці 5.8.

Таблиця 5.8– Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного	Термін використання	Амортизаційні відрахування, грн

		використання, років	обладнання, місяців	
Комп’ютер	45000	2	1	1875,00
Приміщення лабораторії	190000	20	1	791,67
Всього				2666,67

4.2.5 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (5.13)$$

де W_{yi} — встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i — тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ — вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e = 7,5$ грн;

$K_{вni}$ — коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i — коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$$В_e = 0,25 \cdot 250,0 \cdot 7,5 \cdot 0,5 / 0,8 = 292,97 \text{ грн.}$$

4.2.6 Службові відрядження

До статті «Службові відрядження» дослідної роботи на тему Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також

витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{H_{св}}{100\%}, \quad (5.14)$$

де $H_{св}$ — норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$$В_{св} = (116591 + 1052{,}9) \cdot 20 / 100\% = 23528{,}75 \text{ грн.}$$

4.2.7 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{H_{ів}}{100\%}, \quad (5.15)$$

де $H_{ів}$ — норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$$І_в = (116591 + 1052{,}9) \cdot 50 / 100\% = 58821{,}88 \text{ грн.}$$

4.2.8 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків;

витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{H_{нзв}}{100\%}\ , \qquad (5.16)$$

де $H_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$$В_{нзв} = (116591 + 1052{,}9) \cdot 100 / 100\% = 117643{,}77 \text{ грн.}$$

Витрати на проведення науково-дослідної роботи на тему Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + B_{спец} + B_{прг} + A_{обл} + B_е + B_{св} + B_{сп} + I_в + B_{нзв}\ . \qquad (5.17)$$

$$В_{заг}=116591+1052{,}9+12940{,}81+28728{,}61+643{,}5+2666{,}67+292{,}97+23528{,}75+58821{,}88+117643{,}77= 363004{,}48 \text{ грн.}$$

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}\ , \qquad (5.18)$$

де η — коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,9.

$$ЗВ = 363004{,}48 / 0{,}9 = 403338{,}31 \text{ грн.}$$

4.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн» передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку основу майбутнього економічного ефекту будуть формувати:

ΔN — збільшення кількості споживачів яким надається відповідна інформаційна послуга у періоди часу, що аналізуються;

N – кількість споживачів яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки, приймемо 1 особа

$Ц_б$ — вартість послуги у році до впровадження інформаційної системи, приймемо 2000,00 грн;

$\pm\Delta Ц_о$ — зміна вартості послуги від впровадження результатів, приймемо зростання на 500,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.19)$$

Де λ — коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2021 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0{,}8333$;

ρ — коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо ρ=40%;

ϑ — ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році ϑ=18%;

Збільшення чистого прибутку 1-го року:

$$\Delta П_1 = (1 \cdot 500 + 2000 \cdot 1500) \cdot 0{,}83 \cdot 0{,}4 \cdot (1\text{-}0{,}18/100\%) = 640684{,}79 \text{ грн.}$$

Збільшення чистого прибутку 2-го року:

$$\Delta П_2 = (1 \cdot 500 + 2000 \cdot (1500 + 1200)) \cdot 0{,}83 \cdot 0{,}4 \cdot (1\text{-}0{,}18/100\%) = 1153578{,}9 \text{грн.}$$

Збільшення чистого прибутку 3-го року:

$$\Delta П_3 = (1 \cdot 500 + 2000 \cdot (1500 + 1200 + 850)) \cdot 0{,}83 \cdot 0{,}4 \cdot (1\text{-}0{,}18/100\%) = 1516585{,}2$$

грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.20)$$

де $\Delta П_i$ — збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T — період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ=18%;

t — період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = 640684{,}79/(1+0{,}18)^1 + 1153578{,}9/(1+0{,}18)^2 + 1516585{,}2/(1+0{,}18)^3 =$$

$$= 2216736{,}01 \text{ грн.}$$

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \quad (5.21)$$

де $k_{інв}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2;

$ЗВ$ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 403338,31 грн.

$$PV = k_{інв} \cdot ЗВ = 2 \cdot 403338{,}31 = 806676{,}61 \text{ грн.}$$

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \quad (5.22)$$

де $ПП$ — приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 2216736,01 грн;

PV — теперішня вартість початкових інвестицій, 806676,61 грн.

$$E_{абс} = ПП - PV = 2216736{,}01 - 806676{,}61 = 1410059{,}39 \text{ грн.}$$

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.23)$$

де $E_{абс}$ — абсолютний економічний ефект вкладених інвестицій, грн;

PV — теперішня вартість початкових інвестицій, грн;

$T_ж$ — життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_в = \sqrt[Тж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+1410059{,}39\ /806676{,}61)^{1/3} — 1 = 0{,}65.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f\,, \tag{5.24}$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,1;

f — показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,25.

$\tau_{мін}$ = 0,1+0,25 = 0,35 < 0,65 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Інформаційна технологія онтологічного моделювання бази знань з організації бібліотеки» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}\,, \tag{5.25}$$

де $E_в$ — внутрішня економічна дохідність вкладених інвестицій.

$$T_{ок} = 1 / 0{,}65 = 1{,}5 \text{ р.}$$

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн» становить 43 бали, що, свідчить про комерційну важливість проведення даних досліджень оскільки рівень комерційного потенціалу розробки високий. При оцінюванні рівня

конкурентоспроможності, згідно узагальненого коефіцієнту конкурентоспроможності розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 1,16 рази.

Також термін окупності становить 1,5 роки, що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок. Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою Веб-застосунок для організації тренувань та дієти з інтеграцією штучного інтелекту. Частина 1. «Розподілена архітектура та UI/UX дизайн».

Висновки

У комплексній магістерській кваліфікаційній роботі спроектовано розподілену архітектуру веб-застосунку для організації дієти та тренувань з інтеграцією штучного інтелекту та розроблено UI/UX дизайн веб-застосунку.

Проведено аналіз сучасних підходів для вибору архітектури веб-застосунку. Визначені основні критерії для проектування архітектури та проаналізовані функціональні характеристики аналогічних застосунків. Також проведено аналіз предметної області, на основі якого була визначена категорія потенційних користувачів.

Проаналізовано різні типи високорівневої архітектури, а саме — монолітну, мікросервісну та серверлес архітектури, в результаті чого створено розподілену архітектуру шляхом змішування монолітної та мікросервісної архітектур. Обґрунтовано вибір архітектурного стилю при розробці веб-застосунку, в результаті чого був обраний клієнт-серверний архітектурний стиль, оскільки такий клієнт-серверний розподіл допомагає забезпечити ефективну обробку запитів, підвищити масштабованість та краще керувати ресурсами. Розроблено загальну структуру веб-застосунку та визначені основні елементи та зв’язки між ними.

Вдосконалено метод клієнт-серверної взаємодії за допомогою введення проміжного серверу, що дало змогу виконувати запити до бази даних безпосередньо з клієнтської частини, результатом чого стало підвищення швидкості виконання таких запитів. Спроектовано архітектуру веб-застосунку та реалізовано інтерфейс для формування запиту до штучного інтелекту. Проаналізовано трафік та швидкість завантаження сторінки за допомогою сервісу Google PageSpeed Insights. Зроблено висновок, що швидкість завантаження веб-застосунку знаходиться в зеленій зоні, що є досить високим результатом.

В роботі проведений економічний аналіз доцільності розробки. Економічний ефект складає 0.65 грн. Термін окупності складає 1,5 роки.

ВИСНОВКИ

У данній дипломній роботі була розроблена система збереження великих файлів з використанням криптографічного захисту цілісності та конфіденційності. Ця система включає в себе використання блокчейн-технологій, смарт-контрактів і криптографічних методів для забезпечення безпеки та захисту даних користувачів.

Оцінка ефективності системи показала, що вона успішно виконує свої завдання з вищим рівнем безпеки файлів порівняно з аналогічними системами без криптографічного захисту. Продуктивність системи залишається на прийнятному рівні, а швидкість завантаження та доступу до файлів майже не відчутна для користувачів.

Система також виявилася масштабованою і може легко впоратися з великим обсягом даних та багатьма користувачами. Механізми управління доступом дозволяють обмежувати права доступу до файлів, забезпечуючи конфіденційність та визначаючи, хто має доступ до яких ресурсів.

Однак варто враховувати, що впровадження криптографічного захисту файлів може призвести до певного збільшення обчислювальних витрат і витрат на обслуговування системи. Однак ці додаткові витрати можуть бути виправданими, особливо коли мова йде про збереження конфіденційних або важливих даних.

Загалом, система з криптографічним захистом цілісності та конфіденційності файлів виявилася дієвим і ефективним рішенням для забезпечення безпеки даних у великих обсягах. Вона відповідає сучасним стандартам і вимогам щодо збереження та обробки файлів, забезпечуючи надійний захист і конфіденційність інформації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. "Data Integrity in Cloud Storage: Techniques and Challenges" - S. Kumar, R. Mittal, та ін.
2. "Secure and Efficient Cloud Storage Services" - H. Gao, C. Wang, та D. W. Oard.
3. "Blockchain-Based Data Integrity Protection in Cloud Storage" - X. Zhang, J. Liu, та ін.
4. "A Survey of Data Integrity Techniques in Cloud Computing" - S. R. Saripalli, A. K. Sahu, та R. C. Joshi.
5. "Confidentiality and Data Security in Cloud Computing: A Comprehensive Survey" - S. Subashini та V. Kavitha.
6. "Secure Data Storage and Retrieval in Cloud Computing Using Hybrid Cryptographic Techniques" - S. Kalaiselvi, R. Jeyshankar, та ін.
7. "A Comprehensive Study on Cloud Computing" - S. H. Khobragade та S. H. Khobragade.
8. "Data Storage and Retrieval in Cloud Computing: Issues and Challenges" - K. Kumar, G. Gupta, та N. Jain.
9. "Blockchain-Based Data Security in Cloud Computing: A Survey" - N. A. Al-Jaroodi та I. H. Mohamed.
10. "Cryptographic Techniques for Secure Data Storage in Cloud Computing: A Review" - A. T. Khan, M. A. Hossain, та R. M. Islam.
11. "Blockchain and IoT Integration: A Systematic Survey" - A. Dorri, S. S. Kanhere, та R. Jurdak.
12. "Decentralized Storage in Blockchain-Based Cloud Computing" - A. Dinh, I. T. Kim, та Y. Hur.
13. "A Review on Data Security in Cloud Computing" - A. Kaur та V. Juneja.
14. "Privacy-Preserving Cloud Computing: A Survey" - B. Wang, H. Li, та M. Li.
15. "Cloud Computing Security Issues and Challenges: A Survey" - J. S. Sanghvi та D. S. Kothari.
16. "A Comprehensive Survey on Blockchain Technology: Its Application and Future

Prospects" - E. Mahmood, R. A. Hu, та R. Hu.

17. "Big Data Security and Privacy Issues in the Internet of Things" - Y. Zhang, R. Zhang, та S. Chen.
18. "Blockchain for Secure Sharing of Healthcare Data: A Review of Current Approaches" - L. F. L. C. Santos, M. P. C. Oliveira, та M. S. M. Fernandes.
19. "A Survey of Blockchain Security Issues and Solutions" - X. Xiao, J. Xie, та K. Xu.
20. "Data Security and Privacy Preservation in Cloud-Assisted Healthcare Systems: A Survey and Research Directions" - H. He, Y. Zheng, та S. Guo.
21. "A Comprehensive Survey on Data Security and Privacy Preservation in Cloud Computing" - X. S. Rong, L. H. Nguyen, та E. Hossain.
22. "Enhancing Data Security in Cloud Computing Using Multi-Layer Encryption Algorithm" - R. E. V. Subash, S. S. Mohamed, та S. G. Nayar.
23. "Blockchain for Data Security and Privacy in Cloud Computing: A Review" - X. Chen, L. Xu, та S. Dai.
24. "Ensuring Data Security in Cloud Computing: A Comprehensive Review" - T. N. Duong, G. M. Tran, та M. P. Mattson.
25. "A Comprehensive Survey of Cloud Computing: Enabling Technology for Future IoT Applications" - Y. M. M. Ahmed, K. T. T. Yen, та S. Huynh.

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп’ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
проф., д.т.н.. Азаров О.Д..
“29” вересня 2023 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи
“Розподілена система з підтримки функціонування автопаркінгу”
08-54.МКР.031.00.000 ПЗ

Науковий керівник: доцент
к.т.н. каф.ОТ
_____ Кадук О.В
Студент групи 2КІ-22м
_____ Станішевський Д.Ю.

ВНТУ 2023

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Актуальність роботи полягає у розробці системи збереження великих файлів з використанням криптографічного захисту цілісності та конфіденційності. Для використання в розподілених серверних системах з різнотипною архітектурою

1.2 Наказ про затвердження теми МКР.

2 Мета МКР і призначення розробки

2.1 Мета роботи — поліпшити безпеку файлових систем за допомогою сучасних методів криптографії

2.2 Призначення розробки — визначається необхідністю використання безпечної файлової системи

3 Вихідні дані для виконання МКР

3.1 Проведення аналізу існуючих систем, методів та принципів.

3.2 Розробка алгоритму системи.

3.4 Проведення верифікації та аналізу отриманих результатів.

3.5 Виконання розрахунків для доведення доцільності нової розробки з економічної точки зору.

4 Вимоги до виконання МКР

Головна вимога — що система працювала, була захищеною та відказостійкою

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи МКР

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз існуючих технологій, огляд аналогів системи.	19.09.2023	28.09.2023	Розділ 1
2	Визначення архітектури розподіленої системи	5.10.2023	15.10.2023	Розділ 2
3	Розробка алгоритму та демонстраційного додатку	16.10.2023	23.10.2023	Розділ 3
4	Підготовка економічної частини	2.11.2023	12.11.2023	Розділ 5
5	Оформлення пояснювальної записки, графічного матеріалу і презентації	4.12.2023	8.12.2023	ПЗ, графічний матеріал і презентація
6	Підготовка і підпис супроводжуючих документів, нормоконтроль та тест на плагіат	8.12.2023	11.12.2023	Оформленні документи

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання МКР

8.1 При оформлюванні МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104–2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія»;

— документи на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ–03.02.02 П.001.01:21

ДОДАТОК Б

Код смарт контракту системи

```
// Оголошення версії Solidity для компілятора
pragma solidity ^0.8.0;

// Імпорт бібліотеки для роботи з хеш-функціями
import "./SafeMath.sol";

// Конструктор контракту
contract FileStorage {
    // Адреса власника контракту
    address public owner;

    // Структура для зберігання файлів
    struct File {
        string fileName;
        bytes32 fileHash;
        uint256 fileSize;
        bool exists;
    }
    // Зберігання файлів
    mapping(string => File) public files;
    // Подія для сповіщення про завантаження файлу
    event FileUploaded(string fileName, bytes32 fileHash, uint256 fileSize);
    // Модифікатор для перевірки, чи є виконавець власником контракту
    modifier onlyOwner() {
```

```
        require(msg.sender == owner, "Only contract owner can call this function");
        _;
    }
    // Конструктор контракту
    constructor() {
        owner = msg.sender;
    }
    // Функція для завантаження файлу
    function uploadFile(string memory _fileName, bytes32 _fileHash, uint256 _fileSize)
public onlyOwner {
        require(!files[_fileName].exists, "File with the same name already exists");
        // Зберігаємо інформацію про файл
        files[_fileName] = File(_fileName, _fileHash, _fileSize, true);
        // Відправляємо подію про завантаження файлу
        emit FileUploaded(_fileName, _fileHash, _fileSize);
    }
    // Функція для перевірки цілісності файлу
    function verifyFile(string memory _fileName, bytes32 _fileHash) public view returns
(bool) {
        require(files[_fileName].exists, "File does not exist");

        // Перевіряємо, чи співпадає переданий хеш із збереженим
        return files[_fileName].fileHash == _fileHash;
    }
}
```

ДОДАТОК В

Блок-схеми алгоритма завантаження файлів

ДОДАТОК Г

Блок-схема алгоритму цілісності

ДОДАТОК Д

Лістинг програми

```
using System;
using System.IO;
using System.Numerics;
using System.Security.Cryptography;
using Nethereum.Web3;
using Nethereum.Web3.Accounts;

class Program
{
    static void Main()
    {
        Console.WriteLine("Введіть шлях до файлу для завантаження:");
        string filePath = Console.ReadLine();

        if (File.Exists(filePath))
        {
            string fileHash = ComputeSha256Hash(filePath);
            Console.WriteLine($"Хеш SHA-256 для файлу: {fileHash}");

            string encryptedFilePath = EncryptFile(filePath);
            Console.WriteLine($"Зашифрований файл збережено: {encryptedFilePath}");

            string senderAddress = "ВАША_АДРЕСА";
            string privateKey = "ВАШ_ПРИВАТНИЙ_КЛЮЧ";
```

```
            string url = "URL_ПРОВАЙДЕРА";


            UploadFileToSmartContract(senderAddress, privateKey, url, fileHash,
encryptedFilePath);
        }
        else
        {
            Console.WriteLine("Файл не знайдено.");
        }
    }


    static string ComputeSha256Hash(string filePath)
    {
        using (SHA256 sha256 = SHA256.Create())
        {
            using (FileStream fileStream = File.OpenRead(filePath))
            {
                byte[] hash = sha256.ComputeHash(fileStream);
                return BitConverter.ToString(hash).Replace("-", "").ToLowerInvariant();
            }
        }
    }


    static string EncryptFile(string filePath)
    {
        string encryptedFilePath = filePath + ".encrypted";
```

```
        using (Aes aes = Aes.Create())
        {
            aes.KeySize = 256;
            aes.BlockSize = 128;
            aes.Mode = CipherMode.CBC;

            byte[] key = aes.Key;
            byte[] iv = aes.IV;

            using (FileStream fileStream = new FileStream(filePath, FileMode.Open))
            using (FileStream encryptedStream = new FileStream(encryptedFilePath, FileMode.Create))
            using (CryptoStream cryptoStream = new CryptoStream(encryptedStream, aes.CreateEncryptor(key, iv), CryptoStreamMode.Write))
            {
                fileStream.CopyTo(cryptoStream);
            }

            // Зберегти ключ і IV разом з зашифрованим файлом (у реальному використанні це може бути небезпечно)
            File.WriteAllBytes(encryptedFilePath + ".key", key);
            File.WriteAllBytes(encryptedFilePath + ".iv", iv);
        }

        return encryptedFilePath;
    }
```

```
    static async void UploadFileToSmartContract(string senderAddress, string privateKey, string url, string fileHash, string encryptedFilePath)
    {
        var account = new Account(privateKey);
        var web3 = new Web3(account, url);


        string contractAddress = "АДРЕСА_СМАРТ-КОНТРАКТУ";
        var contract = web3.Eth.GetContract(ABI_СМАРТ-КОНТРАКТУ, contractAddress);
        var uploadFunction = contract.GetFunction("uploadFile");
        var receipt = await uploadFunction.SendTransactionAndWaitForReceiptAsync(senderAddress, new HexBigInteger(900000), null, null, fileHash, encryptedFilePath);


        Console.WriteLine($"Транзакція успішно відправлена. Tx Hash: {receipt.TransactionHash}");
    }


    static async System.Threading.Tasks.Task<string> UploadFileToIpfs(string filePath)
    {
        IpfsClient ipfs = new IpfsClient();
        var cid = await ipfs.FileSystem.AddFileAsync(filePath);
        return cid.Id;
    }


    static async void UploadFileHashAndCidToSmartContract(string senderAddress, string privateKey, string url, string fileHash, string cid)
    {
```

```
        var account = new Account(privateKey);
        var web3 = new Web3(account, url);

        string contractAddress = "АДРЕСА_СМАРТ-КОНТРАКТУ";

        var contract = web3.Eth.GetContract(ABI_СМАРТ-КОНТРАКТУ, contractAddress);
        var uploadFunction = contract.GetFunction("uploadFile");

        var receipt = await uploadFunction.SendTransactionAndWaitForReceiptAsync(senderAddress, new HexBigInteger(900000), null, null, fileHash, cid);

        Console.WriteLine($"Транзакція успішно відправлена. Tx Hash: {receipt.TransactionHash}");
    }

    static async System.Threading.Tasks.Task CheckFileIntegrity(
        string senderAddress, string privateKey, string url, string fileName, string expectedHash)
    {
        var account = new Account(privateKey);

        var web3 = new Web3(account, url);

        string contractAddress = "АДРЕСА_СМАРТ-КОНТРАКТУ";

        string contractABI = "ABI_СМАРТ-КОНТРАКТУ";
```

```
        var contract = web3.Eth.GetContract(contractABI, contractAddress);

        var checkFileIntegrityFunction = contract.GetFunction("checkFileIntegrity");

        var result = await checkFileIntegrityFunction.CallAsync<string>(fileName,
expectedHash);

        Console.WriteLine($"Результат перевірки цілісності: {result}");
    }
}

public interface IDataRepository
{
    void AddData(string data);

    string GetData(int id);

    void UpdateData(int id, string data);

    void DeleteData(int id);
}

public class FileDataRepository : IDataRepository
{
    private const string FilePath = "./data.txt";
```

```
public void AddData(string data)
{
    // Додавання даних до файлу
    File.AppendAllText(FilePath, data + Environment.NewLine);
    Console.WriteLine("Додаємо дані до файлу");
}

public string GetData(int id)
{
    // Припустимо, що кожен рядок - це окремий запис
    var lines = File.ReadAllLines(FilePath);

    if (id >= 0 && id < lines.Length)

    {
        Console.WriteLine("Отримуємо дані з файлу");
        return lines[id];
    }
    return "Дані не знайдено";
}

public void UpdateData(int id, string data)
{
    // Оновлення запису в файлі
    var lines = File.ReadAllLines(FilePath).ToList();
```

```
        if (id >= 0 && id < lines.Count)
        {
            lines[id] = data;

            File.WriteAllLines(FilePath, lines);

            Console.WriteLine("Оновлюємо дані в файлі");
        }
    }

    public void DeleteData(int id)
    {
        // Видалення запису з файлу
        var lines = File.ReadAllLines(FilePath).ToList();

        if (id >= 0 && id < lines.Count)
        {
            lines.RemoveAt(id);

            File.WriteAllLines(FilePath, lines);

            Console.WriteLine("Видаляємо дані з файлу");

        }
    }
}
```

```
public class User
{
    public int UserId { get; set; }

    public string Username { get; set; }

    public string FirstName { get; set; }

    public string LastName { get; set; }

    public string Email { get; set; }

    public DateTime Birthdate { get; set; }

    public List<Contract> Contracts { get; set; }
}
public class Administrator : User
{
    public string Department { get; set; }
    public List<Contract> ManagedContracts { get; set; }
}
public class Contract
{
    public int ContractId { get; set; }

    public string ContractName { get; set; }
```

```
    public DateTime StartDate { get; set; }

    public DateTime EndDate { get; set; }

    public decimal Amount { get; set; }

    public User Creator { get; set; }

    public List<User> Participants { get; set; }

    public DataMantainer DataMantainer { get; set; }

    public SmartContract AssociatedSmartContract { get; set; }
}
public class DataMantainer
{
    public int DataMantainerId { get; set; }

    public string OrganizationName { get; set; }

    public string ContactPerson { get; set; }

    public string ContactEmail { get; set; }

    public string ContactPhone { get; set; }
```

```
    public List<Contract> ManagedContracts { get; set; }
}
public class SmartContract
{
    public int SmartContractId { get; set; }

    public string ContractCode { get; set; }

    public string Description { get; set; }

    public DateTime DeploymentDate { get; set; }

    public Administrator DeployedBy { get; set; }

    public List<Contract> AssociatedContracts { get; set; }
}
public interface IDataWorker
{
    string Read(string filePath);

    void Write(string filePath, string data);

    bool Verify(string filePath, string expectedHash);
}
```

```
public class CloudStorageDataWorker : IDataWorker
{
    private readonly string cloudStorageUrl;

    public CloudStorageDataWorker(string storageUrl)
    {
        cloudStorageUrl = storageUrl;
    }

    public string Read(string filePath)
    {
        // Читання даних з хмарного сховища за заданим шляхом (filePath).
        using (WebClient client = new WebClient())
        {
            try
            {
                string data = client.DownloadString(cloudStorageUrl + filePath);
                return data;
            }
            catch (WebException ex)
            {
                // Обробка помилки доступу до хмарного сховища.
                Console.WriteLine("Error reading from cloud storage: " + ex.Message);
                return null;
            }
        }
```

```
    }
    public void Write(string filePath, string data)
    {
        // Запис даних до хмарного сховища за заданим шляхом (filePath).
        using (WebClient client = new WebClient())
        {
            try
            {
                client.UploadString(cloudStorageUrl + filePath, data);
            }
            catch (WebException ex)
            {
                // Обробка помилки доступу до хмарного сховища.
                Console.WriteLine("Error writing to cloud storage: " + ex.Message);
            }
        }
    }
    public bool Verify(string filePath, string expectedHash)
    {
        // Перевірка цілісності даних у хмарному сховищі за заданим шляхом (filePath)
і очікуваним хешем (expectedHash).
        string data = Read(filePath);
        if (data != null)
        {
            using (SHA256 sha256 = SHA256.Create())
            {
                byte[] hashBytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(data));
```

```
            string computedHash = BitConverter.ToString(hashBytes).Replace("-",
"").ToLower();
            return computedHash == expectedHash;
        }
    }
    return false;
  }
}public class FTPDataWorker : IDataWorker
{
  private readonly string ftpServer;
  private readonly string ftpUsername;
  private readonly string ftpPassword;

  public FTPDataWorker(string server, string username, string password)
  {
    ftpServer = server;
    ftpUsername = username;
    ftpPassword = password;
  }

  public string Read(string filePath)
  {
    // Читання даних з FTP-сервера за заданим шляхом (filePath).
    string serverPath = ftpServer + filePath;
    try
    {
      using (WebClient client = new WebClient())
```

```
        {
            client.Credentials = new NetworkCredential(ftpUsername, ftpPassword);
            string data = client.DownloadString(serverPath);
            return data;
        }
    }
    catch (WebException ex)
    {
        // Обробка помилки доступу до FTP-сервера.
        Console.WriteLine("Error reading from FTP server: " + ex.Message);
        return null;
    }
}
public void Write(string filePath, string data)
{
    // Запис даних до FTP-сервера за заданим шляхом (filePath).
    string serverPath = ftpServer + filePath;
    try
    {
        using (WebClient client = new WebClient())
        {
            client.Credentials = new NetworkCredential(ftpUsername, ftpPassword);
            client.UploadString(serverPath, data);
        }
    }
    catch (WebException ex)
```

```
        {
            // Обробка помилки доступу до FTP-сервера.
            Console.WriteLine("Error writing to FTP server: " + ex.Message);
        }
    }

    public bool Verify(string filePath, string expectedHash)
    {
        // Перевірка цілісності даних на FTP-сервері за заданим шляхом (filePath) і очікуваним хешем (expectedHash).
        string data = Read(filePath);
        if (data != null)
        {
            using (SHA256 sha256 = SHA256.Create())
            {
                byte[] hashBytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(data));
                string computedHash = BitConverter.ToString(hashBytes).Replace("-", "").ToLower();
                return computedHash == expectedHash;
            }
        }
        return false;
    }
}
public class AuthManager
{
    private readonly UserManager<ApplicationUser> _userManager;
```

```
    private readonly SignInManager<ApplicationUser, string> _signInManager;
    public AuthManager()
    {
        var userStore = new UserStore<ApplicationUser>(new ApplicationDbContext());
        _userManager = new UserManager<ApplicationUser>(userStore);
        _signInManager = new SignInManager<ApplicationUser, string>(_userManager,
HttpContext.Current.GetOwinContext().Authentication);
    }
    public bool CreateUser(string username, string email, string password)
    {
        var userStore = new UserStore<ApplicationUser>(new ApplicationDbContext());
        var userManager = new UserManager<ApplicationUser>(userStore);
        var user = new ApplicationUser { UserName = "exampleuser", Email =
"user@example.com" };


        var result = userManager.Create(user, "Password123!");


        if (result.Succeeded)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
```

```
    public SignInStatus SignIn(string username, string password, bool rememberMe)
    {
        var userStore = new UserStore<ApplicationUser>(new ApplicationDbContext());
        var userManager = new UserManager<ApplicationUser>(userStore);
        var signInManager = new SignInManager<ApplicationUser, string>(userManager,
HttpContext.GetOwinContext().Authentication);
        var result = signInManager.PasswordSignIn("exampleuser", "Password123!", true,
shouldLockout: false);


        switch (result)
        {
            case SignInStatus.Success:
                // Авторизація успішна
                break;
            case SignInStatus.LockedOut:
                // Обліковий запис заблоковано
                break;
            case SignInStatus.RequiresVerification:
                // Вимагається підтвердження
                break;
            case SignInStatus.Failure:
            default:
                // Помилка авторизації
                break;
        }
    }
}
```

ДОДАТОК Е
ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА
НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи «Система збереження великих файлів з використанням криптографічного захисту конфіденційності та цілісності»

Тип роботи: магістерської дипломна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 90,7% Схожість 9,1%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Станішевський Д. Ю.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Кадук О.В.
(підпис) (прізвище, ініціали)