НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

На правах рукопису
УДК 004.852

До захисту допущено
В. о. зав. кафедри ШІ

О. І. Чумаченко

«___» _____ 2022 р.

Магістерська дисертація

на здобуття ступеня магістра за спеціальністю 122 «Комп'ютерні науки»

на тему: «Інтелектуальна система безпечного збереження великих даних з використанням блокчейн технологій»

Виконав:
студент 2 курсу, групи КІ-11мп
Ігнатов Дмитро Андрійович _____

Керівник: завідувач кафедри ММСА
к.т.н., доцент, Тимощук О.Л. _____

Рецензент: завідувач кафедри СП
д.т.н., професор, Мухін В.Є. _____

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів
без відповідних посилань

Студент (*підпис*): _____

Київ
2022

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Рівень вищої освіти — другий (магістерський)
Спеціальність (ОПП) — 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри ШІ
О. І. Чумаченко
«___» _____ 2022 р.

ЗАВДАННЯ

на магістерську дисертацію студенту Ігнатову Дмитру Андрійовичу

1. Тема дисертації: «Інтелектуальна система безпечного збереження великих даних з використанням блокчейн технологій», науковий керівник дисертації Тимощук Оксана Леонідівна, к.т.н., доцент, затверджені наказом по університету від «03» листопада 2022 р. № 4046-с

2. Термін подання студентом дисертації: 12.12.2022 р.

3. Об'єкт дослідження: задача безпечного збереження великих даних.

4. Предмет дослідження: блокчейн технології, алгоритми та методи безпечного збереження даних.

5. Перелік завдань, які потрібно зробити:
1) здійснити огляд технічної літератури за темою роботи;

2) дослідити актуальність обраної теми;

3) ознайомитись із існуючими методами збереження даних

4) здійснити порівняльний аналіз наявних методів, виявити їх переваги та недоліки;

5) розробити та реалізувати систему, що вирішує задачу безпечного збереження великих даних;

6) провести експеримент, що засвідчує працеспроможність запропонованої системи, виконати аналіз результатів;

8) провести аналіз ринкових можливостей запуску стартап проекту;

7) розробити концептуальні висновки;

8) підготувати ілюстративний матеріал;

9) оформити пояснювальну записку.

6. Перелік графічного матеріалу.

7**.** Дата видачі завдання: 1 вересня 2022 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів роботи	Примітка
1.	Вивчення літератури за темою роботи.	01.09.2022 – 14.09.2022	Виконано
2.	Підготовка першого розділу.	14.09.2022 – 01.10.2022	Виконано
3.	Підготовка другого розділу.	01.10.2022 – 10.10.2022	Виконано
4.	Розробка програмного продукту.	10.10.2022 – 01.11.2022	Виконано
5.	Підготовка третього розділу	01.11.2022 – 18.11.2022	Виконано
6.	Підготовка частини стартап-проєкту	18.11.2022 – 21.11.2022	Виконано
9.	Концептуальні висновки. Перспективи розвитку отриманих рішень	21.11.2022 – 25.11.2022	Виконано
10.	Оформлення пояснювальної записки	25.11.2022 – 12.12.2022	Виконано

Студент Ігнатов Дмитро

Науковий керівник дисертації Тимощук Оксана

РЕФЕРАТ

Магістерська дисертація: 132 с., 32 табл., 17 рис., 24 джерел, 1 додаток.

ІНТЕЛЕКТУАЛЬНА СИСТЕМА, БЛОКЧЕЙН ТЕХНОЛОГІЇ, ВЕЛИКІ ДАНІ, ШИФРУВАННЯ, КРИПТОГРАФІЯ, РОЗПОДІЛЕНА СИСТЕМА, ДЕЦЕНТРАЛІЗОВАНА СИСТЕМА

Об’єктом дослідження є задача збереження великих даних.

Предмет дослідження – алгоритми та методи безпечного збереження з використанням блокчейн технологій.

Мета дослідження полягає у аналізі алгоритмів і методів безпечного збереження, що базуються на блокчейн технологіях, а також методів, що здатні обробляти великі дані.

У роботі запропоновано та розроблено систему безпечного збереження великих даних, що використовує блокчейн технології. Дана система здатна обробляти велику кількість вхідних даних, кодувати їх та реплікувати їх у розподіленій системі.

Проведено порівняння запропонованої системи з існуючими методами збереження даних: наявними популярними хмарними рішеннями, BitTorrent peer-to-peer системами та існуючими off-chain системами збереження даних.

ABSTRACT

Master thesis: 132 p., 32 tab., 17 fig., 24 references, 1 appendix.

INTELLIGENT SYSTEM, BLOCKCHAIN TECHNOLOGIES, BIG DATA, ENCRYPTION, CRYPTOGRAPHY, DISTRIBUTED SYSTEMS, DECENTRALIZED SYSTEMS

The object of following study is a big data storage problem.

The subjects of current thesis are algorithms and methods of safe data storage using blockchain technologies.

The purpose of study is to conduct research on effectiveness of different approaches to store big data tasks, such as methods to process big data.

The work considers alternative methods of authorization, such as authorization through biometrics, as well as the latest, authentication through the blockchain.

In following research new safe big data storage approach is proposed and implemented. That model type leverages blockchain technologies. Proposed system can process a large amount of incoming data, encrypt it and replicate it in a distributed system.

During the study, system comparisons were made. In particular, proposed system variations were compared with existing popular cloud based solutions, BitTorrent peer-to-peer networks, and available off-chain storage systems.

ЗМІСТ

ВСТУП

Хоча знання завжди давали перевагу в прагненні до багатства і влади, в сучасному світі, в епоху цифрових технологій, знання зайняли місце основної зброї. Ймовірність зловживань, пов'язаних з використанням зібраних і накопичених даних про особу, зросла з розвитком інформаційних технологій і загальнодоступних засобів масової комунікації. У зловмисників з'явилася можливість швидкої обробки персональних даних, що створює загрозу правам і законним інтересам особи.

В умовах глобалізації діяльність людини все більше переплітається із всесвітньою мережею Інтернет, і за останні кілька десятиліть її користувацька база збільшилася в рази.

Високі технології, що швидко розвиваються, продукують інструменти для вирішення спільних завдань. Більшість транзакцій зараз здійснюється в Інтернеті, і людство вже не уявляє свого життя без нього.

Гнучкість, простота і швидкість здійснення численних операцій купівлі-продажу роблять роботу в Інтернеті вигідною для покупців, але при цьому існує значний ризик для особистої інформації.

Безпека персональних даних є актуальним питанням, і вкрай важливо захищати будь-яку особисту інформацію, яка може бути знайдена в Інтернеті.

Безсумнівно, враховуючи широке використання ПК і мереж для обробки і передачі даних, ці регіони повинні бути повністю захищені від потенційного вторгнення сторонніх осіб. Вона спотворюється або втрачається. Статистика свідчить, що порушення цілісності та конфіденційності використовуваних даних призводить до фінансових втрат для більш ніж 80% компаній.

Більш поширеним джерелом небезпек для особистої інформації вважається Інтернет. Майже кожна людина в сучасній культурі має

електронну поштову скриньку, іноді кілька акаунтів (особистих і робочих), а також профілі в різноманітних соціальних мережах, в тому числі професійних соціальних мережах. У будь-якому випадку, злом профілю може призвести до втрати особистої інформації, яка була надіслана через сервіс, як правило, через пошту та соціальні мережі, або яка була розміщена на сторінці облікового запису, включаючи паспортні дані та іншу важливу інформацію. Конституція є основним законом держави, і всі способи протиправної діяльності, що призвели до втрати персональної інформації, є її порушенням. Якщо сконцентруватися на електронній комерції, оскільки покупки в Інтернеті стали звичним явищем для багатьох громадян, то окремо постає питання захисту персональних даних в мережі Інтернет.

При здійсненні таких операцій важливо ретельно перевіряти веб-сайт, на якому купується товар, щоб переконатися, що він відповідає всім чинним законам. Також слід уникати підключення власної банківської картки до платіжної системи сайту, оскільки ця операція має додатковий ризик. Сайти з пошуку роботи та портали, які пропонують персоналізовані (тобто такі, що призначені для конкретного громадянина і містять його персональну інформацію) послуги для населення, також можуть становити загрозу для персональної інформації в Інтернеті.

Вивчення мільярдів індивідуальних взаємодій, за допомогою яких суспільство поширює ідеї, гроші, товари і чутки, стало можливим завдяки цифровим технологіям. У новій цифровій ері буде життєво важливо управляти людьми по-новому. У цьому сценарії тестування зв'язків потрібно буде починати раніше і частіше, ніж у минулому. Для створення суспільства, яке підлягає індивідуальному інформаційному контролю, необхідно створити так звані "живі лабораторії", де ідеї можуть бути протестовані. Всі люди могли б анонімно покращувати потік інноваційних ідей, не турбуючись про розголошення особистої інформації. У постіндустріальному суспільстві поняття "приватність" еволюціонувало і стало означати, що певна

інформація, яка є доступною для одних людей, але не для інших, вважається приватною. Конфіденційність означає вимогу уникати розголошення особистої інформації, і конфіденційність виступає в ролі судді, який визначає, який об'єкт повинен бути більш жорстко регульованим. Підтримка тонкого балансу між анонімністю та відкритістю має вирішальне значення, і споживачі можуть зробити це, встановивши браузер або використовуючи соціальні мережі. Наразі Інтернет має майже безмежні можливості для обміну даними, і ці можливості постійно вдосконалюються. Сьогодні Інтернет розглядається як соціальне середовище, яке є активним і пов'язує велику кількість людей.

Мета досліджень – розробка безпечного сховища інформації, здатного працювати з великими обсягами даних, на базі технології блокчейн.

Об'єкт дослідження – захищеність технології блокчейн та хмарних сховищ.

Предмет дослідження – методи та способи захисту великих даних в технологіях блокчейн та хмарних сховищ.

РОЗДІЛ 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ

1.1. Огляд технології блокчейн

Блокчейн (від англ. Blockchain) - це ланцюжок блоків або якщо бути точніше розподілена база даних. Вперше цей термін був застосований як назва для розподіленої бази даних криптовалюти біткоін. Однак дану технологію можна використовувати не тільки для криптовалюти[1].

Технологія зберігання інформації блокчейн дозволяє вирішити існуючу проблему, пов'язану з відсутністю гарантій з боку посередників, які виступають в якості третіх осіб при здійсненні тих чи інших дій. Наприклад, немає ніяких гарантій, що банк, який виступає гарантом вашого платежу, не збанкрутує.

Блокчейн - це база даних, у якій немає керуючого, самі учасники є керуючими. Так звані Майнери. Вона заснована на тимчасовій (p2p) мережі, загальному реєстрі і криптографії публічного і приватного ключа. Увійшовши в блокчейн-мережу, користувач підключається до інших комп'ютерів мережі для того, щоб обмінюватися з ними даними: блоками і записами. Отримавши нові дані, кожен користувач перевіряє їх коректність, і, переконавшись у достовірності, зберігає їх у себе, а також передає коректні дані далі по мережі.

Учасники мережі діляться на дві групи: звичайні користувачі, які створюють нові записи, і Майнери, які створюють блоки. Звичайні користувачі створюють і поширюють через мережу записи, наприклад, про грошові перекази або про передачу прав власності. Майнери збирають записи, перевіряють їх і записують в блоки, а потім розсилають ці блоки по мережі. Після чого звичайні користувачі отримують блоки і зберігають їх у себе, щоб можна було коректно створювати свої і достовірно перевіряти чужі

нові записи. Діяльність Майнерів називається Майнінг. Так само існують інші способи перевірки достовірності даних, наприклад PoS[2].

Як випливає з назви технології, в її основі лежить ланцюжок послідовно пов'язаних блоків. Нові блоки завжди додаються строго в кінець ланцюжка. Блок складається з заголовка і тіла, що містить записи. Блоки пов'язані з допомогою ключів, оскільки в заголовку кожного блоку зберігається ключ попереднього блоку. Це забезпечує захищеність мережі.

Ключ кожного блоку розрахований на дані всього блоку і ключ попереднього блоку. А це значить, що в ключі будь-якого блоку закодовані не тільки записи цього блоку, але і всі попередні блоки. При цьому ключ блоку повинен задовольняти правилам безпеки, встановлює рівень захищеності мережі. Наприклад, в біткоіни ключі перших блоків починалися з десяти нулів, що встановлювало ступінь складності створення нового блоку.

Майнер - це користувач блокчейн-мережі, який крім перевірки і поширення даних займається і створенням нових блоків. Це також може бути спеціалізована програмна платформа. Отримавши нові записи від інших учасників мережі, майнер збирає їх разом, формує заголовок майбутнього блоку і розраховує ключ блоку. Щоб знайти підходяще значення ключа, Майнеру доводиться робити величезну кількість перерахунків. Коли відповідний ключ знайдений, Майнер зберігає блок і відправляє його іншим учасникам мережі. Тепер всі записи в блоці підтверджені і захищені кодом, який вельми нелегко підробити. Причому, в ключі блоку закодований і ключ попереднього блоку, який тепер підробити не можна. Така витончена процедура розрахунку ключів ускладнює створення блоку, але ще більше вона ускладнює створення підроблених блоків, роблячи це майже неможливим.

Записи в тілі блоку також захищені шляхом зв'язування в ланцюжок. Кожен запис містить посилання на попередній запис-джерело, а також блокуючу умова, щоб розблокувати правило. Для опису правил і умов

використовується мова програмування, який дозволяє задавати складну логіку і правила взаємодії учасників. Джерел і результатів в кожному записі може бути кілька, тобто запис може перетворити кілька записів-джерел в кілька записів-результатів. Таким чином, блокчейн призводить нас до «розумним» контрактами, що дозволяє формалізувати відносини не тільки між людьми, але і між роботами і програмами, що створює передумови для використання технології в Інтернеті речей. Наприклад, в концепції «розумного» будинку, який контролює витрату електрики, газу, води, кількості продуктів в холодильнику, автоматично укладає контракти на поставки всього необхідного і оплачує їх. Блокчейн в своїй основі відкрита і публічна, і переглянути її вміст можна без проблем. Для цього є програми-аналізатори та онлайн-сервіси.

Основою технології блокчейн є електронний реєстр всіх фінансових транзакцій. Як правило, такі електронні «бухгалтерські книги» є однією з головних точок уразливості. Якщо зловмисники отримають доступ до головного реєстру записів, то це може привести до цілковитого «падіння» системи, адже, отримавши доступ до записів, зловмисник може вкрасти необмежену суму грошей або опанувати будь-якої особистої інформації, просто переглянувши список транзакцій.

У блокчейні реєстр записів є децентралізованим - це означає, що одиночний комп'ютер або система не можуть отримати контроль над всією «бухгалтерської книгою». А значить, для того, щоб отримати доступ до головного реєстру записів, знадобилося б організувати неймовірно складну і чітко скоординовану операцію, згідно з якою в один момент тисячі пристроїв були б атаковані одночасно.

Іншим принципом, що забезпечує виняткову безпеку, є сама ланцюг транзакцій. Головний реєстр записів є довгий ланцюжок послідовних блоків. Кожен блок, що входить до даний ланцюг, є лише частиною загальної

структури, яка бере свій початок від найпершої виробленої в даній системі операції.

Це означає, що будь-кому, хто вирішить змінити інформацію про однієї транзакції, перед цим доведеться вкрай акуратно і точно змінити всі записи, що ведуть до цієї транзакції. Виходячи з чого, передбачуване втручання виглядає вкрай складним процесом, що також є одним з плюсів в побудові безпеки блокчейна.

Крім того, є також і інші елементи, що забезпечують захист блокчейна.

Більше двох користувачів підтверджують і забезпечують безпеку проведеної транзакції. Навіть в більшості сучасних процесингових систем в перевірці задіяні тільки кілька рівнів верифікації - як правило, це продавець, покупець і якісь треті особи (найчастіше банк або кредитна агенція).

Однак в системі блокчейна існує від декількох сотень до декількох тисяч різних вузлів, на кожному з яких зберігається повна копія реєстру записів. Тому будь-який з цих вузлів також може брати участь в перевірці транзакції, і якщо вузол з якихось причин не приймає транзакцію, то вона буде скасована. Подібний розклад майже до мінімуму знижує можливість створення помилкової або шахрайської транзакції.

Криптографічні ключі, які використовуються системою в обмінних процесах, також є дивом сучасної кібербезпеки. Кожен зашифрований ключ являє собою довгу, складну послідовність даних, практично не піддається розшифровці. А якщо врахувати, що для підтвердження потрібні два таких унікальних ключа, то система починає виглядати мало не неприступною фортецею. При цьому вважається, що блокчейн володіє унікальною системою безпеки, тому що при подібному рівні захисту в ньому вдається зберегти майже повну прозорість транзакцій.

Але, як вже говорилося, навіть блокчейн не ідеальний. У нього, як і у будь- який інший системи, є слабкі місця. Так що, якщо ви плануєте використовувати криптовалюта і вкладати в неї свої кошти, або якщо вам в

майбутньому доведеться мати справу з блокчейном, то просто необхідно знати і розуміти потенційно вразливі місця технології. Тому постарайтеся запам'ятати такі особливості, що стосуються безпеки даної технології:

Якщо ви вирішите створити систему на основі технології блокчейн з нуля, то одна невелика помилка може стати фатальною і «покласти» всю вашу розробку. Звичайно ж, це не можна вважати недоліком самого блокчейна - це, скоріше, стосується особливостей його використання. Крім того, середньостатистичній людині набагато важче розібратися в блокчейне через його складності, що, в свою чергу, означає, що багато хто не до кінця розуміють ризики, пов'язані з використання системи, а також в повному обсязі використовують доступний функціонал.

Для роботи блокчейна необхідні як мінімум кілька сотень, а ще краще кілька тисяч узгоджено працюють вузлів. Саме через це система є вкрай вразливою до атак на початкових етапах роботи. Наприклад, якщо який-небудь користувач зможе отримати контроль над 51% вузлів системи, то він зможе повністю контролювати результат роботи. А якщо в системі всього 20 вузлів, то подібний варіант розвитку подій більш ніж можливий.

Структура блокчейна - це також одна з причин, по якій може бути порушено нормальне функціонування системи. Так, якщо система отримає надто широке поширення, а інфраструктура блокчейна виявиться не готовою до такого обсягу операцій, то в результаті може знизитися швидкість проведення транзакцій, можуть з'явитися проблеми зі зберіганням даних, а це все не кращим чином вплине на ефективність мережі.

Хоч і не можна сказати, що даний пункт безпосередньо пов'язаний з безпекою блокчейна, але політика системи може вплинути на її застосування і подальший розвиток. З огляду на те, що валюта в системі блокчейн є міжнародною і децентралізованої, це, по суті, знецінює національну валюту, контрольовану державою. І, природно, на даний момент керівні органи деяких держав прагнуть ввести більш суворі обмеження на використання

блокчейна. Уряди різних країн сподіваються взяти систему під контроль до того, як вона стане серйозним конкурентом і почне загрожувати їхній економіці. Непрямим чином це також є суттєвою загрозою безпеки блокчейна, яка може значно уповільнити поширення технології.

Наприклад, NiceHash - сторонній ринок для Майнінг біткоіни - був не так давно зламаний, в результаті чого було викрадено криптовалюта на більш ніж 60 мільйонів доларів. Як виявилося, дана платформа була небезпечною. Тобто, це не помилка безпеки самої системи блокчейн. Швидше, навпаки, кіберзлочинці отримали доступ до системи NiceHash за допомогою блокчейна[3].

При транзакціях в системі blockchain використовуються публічні і приватні криптографічні ключі. Самі по собі такі ключі зламати майже неможливо, проте кіберзлочинець може отримати їх більш простим і звичним способом. Наприклад, ключі можна дістати в тому випадку, якщо ви зберігаєте їх на небезпечній або слабозахищеній платформі. Так, якщо хтось зламає ваш поштовий ящик, то він зможе отримати доступ до всіх ваших листів, а значить, і до ключів вашого профілю в блокчейне. В такому випадку зловмисник зможе заволодіти вашими коштами, видавши себе за вас. І це одне з головних питань, що стосуються безпеки системи.

Користувачі системи також можуть попастися на інші, більш традиційні прийоми шахраїв. По суті, такі шахрайські схеми не вважаються слабким місцем в системі безпеки блокчейна. Так, наприклад, ви можете отримати електронного листа, в якому незнайомий вам людина буде переконувати вас, що саме ви стали тим щасливчиком, який виграв щось значне. Або ж шахраї можуть запропонувати вам витратити вашу криптовалюта на якийсь товар або послугу, яку ви ніколи не отримаєте.

1.2. Безпека інформаційної технології блокчейн та її похідних

На даний момент головною загрозою для блокчейна, щодо гіпотетичної, є «атака 51%», коли зловмисник може зробити відкат транзакцій, друкуючи альтернативні блоки на бічній ланцюжку (гілці) і гарантовано спростовуючи те, що відбувається в основний ланцюжку блокчейна. По суті, це схоже на «човниковий біг». Однак з огляду на ресурсомісткість рішення хеш-функції і емісії нових біткоіни, поки що такий варіант здається малоймовірним. Змова власників найбільших майнінгових пулів теж виглядає непереконливим (якщо не враховувати статистику найбільших виробників біткоіни). Але подібні приклади вже були: один з пулів - ghash.io - набрав потужність, близьку до 50%, після чого власники припинили прийом нових користувачів, щоб не створювати компрометуючу ситуацію[4].

Розглянемо сценарій, коли атакуючий намагається генерувати альтернативну ланцюжок швидше ніж чесні вузли. Навіть якщо це вдасться, він не зможе робити в системі будь-які зміни, наприклад створювати монети з повітря або брати монети, які йому ніхто не перекладав. Вузли не приймуть неправильну транзакцію в якості платежу, а чесні вузли ніколи не приймуть блок з неправильною транзакцією. Атакуючий може лише змінити одну зі своїх власних транзакцій повернувши собі платіж, який він нещодавно здійснив. Гонка між чесної ланцюжком і ланцюжком атакуючого може бути описана в термінах біноміального випадкового блукання. Вдале подія це зростання чесної ланцюжка на один блок з наближенням до мети на +1, невдале подія це зростання на один блок ланцюжка атакуючого зі зменшенням відриву на -1. Можливості атакуючого в гонці в умовах обмежень, схоже на опис проблеми Gambler's Ruin (розорення картяра). І так, картяр з необмеженим кредитом починає гру в умовах обмежень і може

потенційно провести необмежене число партій щоб спробувати досягти беззбитковості. Ми можемо порахувати вірогідність досягнення нею беззбитковості або те ж саме, що атакуючий обжене чесних будівельників ланцюжка (формула 1.1).

Нехай:

p – ймовірність, що чесний хост знайде наступний блок;

q – ймовірність, що атакуючий знайде наступний блок;

q_z – ймовірність, що атакуючий виграє гонку якщо він відстав на z блоків.

$$q_z = \begin{cases} 1, & p \leq q \\ (q/p)^z, & p > q \end{cases} \qquad (1.1)$$

Припустимо, що p> q, тоді ймовірність експоненціально зменшується при збільшенні числа блоків, на яке відстав атакуючий. Таким чином, якщо йому не вдасться вирватися вперед на самому початку, то його шанси перемогти в подальшому стають зникаюче малі. Розглянемо тепер, як довго одержувач платежу повинен чекати, щоб бути впевненим що відправник не зможе змінити транзакцію. Припустимо, що відправник - це атакуючий, який хоче щоб одержувач повірив що платіж проведено, але через деякий час повернути платіж собі назад. Одержувач буде сповіщений коли таке трапиться, але відправник сподівається, що буде вже пізно. Одержувач генерує нову пару ключів і віддає публічний ключ відправника незабаром після свого підпису. Це не дає можливості відправнику підготувати ланцюжок блоків заздалегідь працюючи над нею з випередженням для виконання транзакції в даний момент. Тільки коли транзакція послана, нечесний відправник може почати в секреті працювати над паралельною ланцюжком, що містить альтернативну версію цієї транзакції. Одержувач чекає, поки транзакція буде додана в блок і Z блоків буде додано після цього. Він не знає на якому етапі будівництва знаходиться атакуючий, але вважаючи що чесні

блоки будувалися з однаковим середнім часом на один блок, очікуване значення приросту атакуючого може бути знайдено через розподіл Пуассона (формула 1.2).

$$\lambda = z\frac{q}{p} \tag{1.2}$$

Для отримання ймовірності, з якою атакуючий все ще може вийти вперед (формула 1.3), помножимо щільність пуассоновського розподілу кожного значення прогресу атакуючого на ймовірність того, що він вийде вперед з даної точки (формула 1.4).

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} * \begin{cases} (q/p)^{(z-k)} & k \leq z \\ 1 & k > z \end{cases} \tag{1.3}$$

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} * (1 - (q/p)^{(z-k)}) \tag{1.4}$$

Чисельно проаналізувавши отримане вираз, можна легко переконатися, що ймовірність експоненціально зменшується зі зростанням z.

У різних продуктах технології блокчейн використовуються різні методи підтвердження блоків. Так, в біткоіни застосовується метод proof-of-work - Підтвердження блоків обчислювальною потужністю. Інший варіант закриття блоків - proof-of-stake, коли блоки друкувати не обчислювальною потужністю, а за допомогою грошей, що перебувають у людей на руках. У цьому випадку, щоб провести «атаку 51%», необхідно мати 51% монет, що обертаються в системі. Як і у випадку з «човниковим бігом», якщо зловмисник володіє більш ніж 51% монет, він також зможе створити альтернативну ланцюжок, яка перетвориться в основну. Ця ситуація нагадує

голосування на зборах акціонерів, коли у одного з власників є на руках контрольний пакет, що блокує голоси інших власників[5].

Якщо безпеку блокчейна викликає мінімум побоювань, то збереження біткоіни, навпаки, породжує багато запитань, адже, як і звичайні паперові гроші, криптовалюта теж можна викрасти. Ключ записи в блокчейне - це хеш-функція від публічного ключа. Невпевнено або недбале зберігання закритого ключа може призвести до крадіжки або втрати біткоіни. За даними Harvard Business Review, вартість втрачених біткоіни вже становить близько $ 950 млн. Найпростіший спосіб убезпечити себе - створити пароль гаманця. Але якщо хакер викраде і ваш гаманець, і ваш пароль, відновити викрадені біткоіни буде практично неможливо, оскільки транзакції, подані з вкраденими ключами, здаються перевіряючим вузлів відрізнятись від законних транзакцій. Деякі скептики стверджують, що хакери зможуть зламати ключ, використовуючи сервіси, що обчислюють паролі по хешу. Однак, з огляду на нинішні обчислювальні потужності, це виглядає малоймовірним. Але якщо раптом з'явиться алгоритм, що дозволяє ефективно факторизовать еліптичні криві, то виникне ймовірність того, що до адрес-гаманцях, з яких витрачалися гроші, легко можна буде підібрати закриті ключі.

Чи не менше запитань викликає і надійність бірж, що зберігають криптовалюта. У серпні 2016 року зі гонконгської біржі Bitfinex - однієї з чотирьох найбільших в світі майданчиків для кріптовалютних торгів - викрали 119 756 біткоіни (близько $ 65 млн). За Bitfinex закріпилася репутація одного з найбільш надійних і безпечних організацій: більшість призначених для користувача засобів зберігалися в гаманцях з мультіподпісью і в «холодних сховищах». Незважаючи на це, зловмисникам вдалося обійти захист Bitgo, в тому числі двухфакторну аутентифікацію і механізм мультіподпісі, і зробити масову крадіжку з індивідуальних гаманців користувачів. Деталі злому так і не були донесені до широкого загалу, проте в

ЗМІ тиражувалося думку про можливу причетність співробітників Bitfinex до злому, що знову ставить питання про людський фактор.

За даними blockchain.info, Майнер, видобувні біткоіни, виробляють 450 тисяч трлн обчислень в секунду. Кожна спроба вимагає енергії. Щорічно на виробництво біткоіни йде від 2 терават-годин (це більше, ніж споживає 150-тисячне місто в Каліфорнії) до 40 терават-годин (це дві третини від споживання 10-мільйонного округу Лос-Анжелес).

У США кожна транзакція біткойнов коштує близько 6 доларів, і в результаті більшість Майнерів відкривають «ферми» в країнах з дешевою електроенергією. За даними bitcoinity.org, в лютому 2016 року ТОП-виробників біткоіни виглядав наступним чином:

1) Antpool - 25,90%;

2) F2pool - 22,60%;

3) Bitfury - 15,09%;

4) BtcChina - 12,78%;

5) bw.com - 6,34%;

6) інші - 17,29%.

Чотири з п'яти перерахованих Майнерів, за винятком Bitfury, мають китайську «прописку». Це породжує ризик «картельних змов» та «загрози 51%», а також ставить питання про мотивацію компаній, в дійсності контролюючих систему, - їх інтереси можуть відрізнятися від інтересів людей, які тримають активи в біткоіни. Нарешті, ніхто не відміняв політичні ризики і вплив китайських регуляторів на інтернет-галузь.

У 2017 р криптовалюта Ethereum, побудована на мові Тьюринга Віталіком Бутеріном, постраждала від «контрагента» - Decentralized Autonomous Organization (децентралізованої автономної організації). DAO - цифрова компанія або віртуальний хедж-фонд, частку в якому можна отримати шляхом вкладення особистих коштів і покупки на них DAO-токенів - стала жертвою власних недоробок. У коді DAO виявили уразливості, які

дозволили викрасти токени. Ethereum довелося здійснити хардфорк і провести зворотний повернення токенов. Хардфорк передбачає поділ і поява іншої мережі, яка містить в собі всю інформацію і ланцюжок блоків блокчейн первинної мережі. Після поділу нова мережа починає жити своїм життя і створювати власну історію транзакцій. Зазвичай хардфорк роблять в тому випадку, коли в співтоваристві з'являються розробники, яких не влаштовують поточні можливості мережі, вони хочуть відокремитися і поліпшити мережу на свій розсуд[6].

Проблем з безпекою у Ethereum не було проте проект зазнав репутаційні втрати через недоліки DAO. Одночасно з блокчейном розвиваються і інші технології, і ступінь їх впливу і небезпеки поки не до кінця очевидна. Так, дослідники з Університету Ньюкасла представили механізм управління ботнетом для відправки повідомлень ботам в мережі біткоіни. Ботнет (англ. Botnet, походить від слів robot і network) - комп'ютерна мережа, що складається з певної кількості хостів, з запущеними ботами - автономним програмним забезпеченням. приховано встановлюються на пристрій жертви і дозволяє зловмиснику виконувати якісь дії з використанням ресурсів зараженого комп'ютера. Не виключено, що мало-помалу боти відсунутий людей від Майнінг біткоіни, як це сталося у випадку з ботнетом інтернету речей Mirai. У квітні 2017 року фахівці IBM виявили, що Mirai активно встановлює код Майнінг біткоіни на комп'ютери деяких зі своїх жертв. Так що «ботізація» Майнінг - поки не до кінця очевидна, але цілком відчутна перспектива.

Залишаться і звичайні проблеми інформаційної безпеки. Ніхто нічого не зможе зробити, якщо з вашого комп'ютера вкрадуть не тільки гаманець, але і пароль до нього. І тут блокчейн не панацея, а додаткова вразливість. Адже випадок з DAO все ж винятковий, і заради однієї людини ніхто відкочувати блоки не буде. Хоча розслідувати крадіжку грошей, можливо, буде простіше. У відкритій системі відразу буде видно, коли гроші зняли і

куди перевели. Але ось розплутувати подальшу ланцюжок вже складно. Правоохоронні органи, як правило, не справляються і зі звичайною крадіжкою грошей через Інтернет, а про те, щоб розслідувати факт крадіжки в блокчейн-ланцюжку, і говорити нема чого. Їм просто не вистачить компетенції.

Коли говорять про «успішні атаки на біткоіни», найчастіше мають на увазі успішні атаки на біржу, яка торгує біткоіни, що, звичайно, в принципі невірно. Адже сама ланцюжок блоків при цьому не компрометується. Але постраждалим від цього не легше. Останній гучний випадок стався в серпні 2016 року. З гонконгської криптовалютної біржі Bitfinex викрали 119 756 біткоіни (близько 65 млн дол.). Bitfinex є однією з чотирьох найбільших майданчиків для криптовалютних торгів і належить фінансово-технологічної компанії iFinex. Остання була заснована в 2012 році зі штаб-квартирою в Гонконгу. За п'ять років існування біржа тільки один раз піддалася злому «гарячого гаманця» - в травні 2015 го, проте втрати були незначні.

До сих пір Bitfinex зберігала репутацію однієї з найбільш надійних і безпечних криптовалютних бірж. Більшість призначених для користувача засобів зберігалися в гаманцях з мультипідписом і в «холодних сховищах». Це особливо підігріло інтерес до проведеної крадіжці, адже «холодне зберігання» вважалося найнадійнішим способом криптовалюта і використовувалося для заощадження.

1.3. Хмарне сховище даних

День у день виникає гостра необхідність в надійному зберіганні і швидкому зручному обміні великими обсягами найрізноманітніших даних

між користувачами мережі, що знаходяться в самих різних куточках нашої планети.

Одним з найбільш оптимальних, ефективних, зручних і тому затребуваних рішень стали так звані хмарні сервіси для зберігання даних, що дозволяють зберігати, оперативно редагувати і швидко передавати значні обсяги найрізноманітніших файлів. Причому доступ до тих або інших даних можуть одночасно мати багато користувачів, або ж інформація може носити конфіденційних характер і бути призначеною тільки для особистого користування[7].

У загальних рисах хмарне сховище даних являє собою віртуальну мега-флешку, віртуальний файлообмінник або сервер з даними, доступ до яких можна отримати з будь-якого пристрою, підключеного до мережі і використовуваному хмарного сервісу.

Однак на відміну від традиційного сервера, що завантажується в хмарні сховища інформація розміщена одночасно на безлічі мережевих серверів, в тому числі, що знаходяться за тисячі кілометрів на іншому кінці Землі.

Маючи рахунок у сервісі, що надає «хмара» певного, як правило, відносно невеликого обсягу, через сайт або встановлюється на комп'ютер або інший пристрій (планшет, смартфон) клієнт-програму, можна автоматично копіювати інформацію з жорсткого диска в «хмару».

Причому, синхронізувавши пристрій і віртуальне сховище, можна автоматично мати під рукою найбільш актуальну інформацію. Це означає, що редаговані файли на комп'ютері або прямо в «хмарі» будуть автоматично оновлюватися або переноситься у відредагованому варіанті туди, де інформація не зазнала зміни. Це зручно і дозволяє істотно знизити ризик випадкової втрати цінних даних.

Причому, синхронізувавши пристрій і віртуальне сховище, можна автоматично мати під рукою найбільш актуальну інформацію. Це означає, що редаговані файли на комп'ютері або прямо в «хмарі» будуть автоматично

оновлюватися або переноситься у відредагованому варіанті туди, де інформація не зазнала зміни. Це зручно і дозволяє істотно знизити ризик випадкової втрати цінних даних.

Безкоштовно хмарні сервіси надають небагато місця, проте платний аккаунт істотно розширює обсяг і функціонал «хмари». Також стежачи за програмою лояльності і проведеними хмарними сервісами акціями, можна абсолютно безкоштовно розширити обсяг свого віртуального сховища до значних розмірів.

1.3.1. Переваги хмарного сховища даних

Основні переваги використання хмарних сховищ даних, на відміну від сумного жорсткого диска, свого комп'ютера або навіть традиційного сервера очевидні.

«Хмари» дозволяють заливати, зберігати і обмінюватися файлами в великих обсягах (понад 20 МБ). Передати важкий файл по електронній пошті складно або зовсім неможливо. На флешці - довго, незручно, а часто не представляється можливим на увазі фізично велику відстань між відправником і отримувачем. Рішення - завантажити інформацію в хмарне сховище, створивши там папку, після чого поділиться з адресатами посиланням поштою.

Доступ до збережених в «хмарі» файлів можна отримати де і коли завгодно з будь-якого пристрою, головне мати вихід у всесвітню павутину. Це особливо зручно, наприклад, при необхідності звернення до будь-яким загальним робочим документам, які, при відповідних настройках, що проводяться власником облікового запису, можна редагувати в режимі онлайн.

Спеціальні алгоритми резервування даних, що зберігаються, що застосовуються на хмарних сервісах, практично повністю виключають ймовірність втрати важливих даних через прикрі збої або пошкоджень і виходу з ладу жорсткого диска або проблем з традиційним сервером.

І це тільки лише можливості безкоштовного аккаунта. Користування хмарним сервісом на платній основі розширює список переваг.

1.3.2. Недоліки хмарного сховища даних

Однак в роботі з «хмарами» справедливості заради можна відзначити і кілька основних мінусів.

Перший і головний - це недостатнє опрацювання питання забезпечення безпеки. Незважаючи на пропаговану політику конфіденційності, до інформації, що завантажується залишається відкритим доступ співробітників сервісу і його програмного забезпечення.

Другий недолік випливає частково з першого - при зломі сховища або сервісу, наприклад, в результаті хакерської атаки, конфіденційні файли можуть потрапити під загальний доступ або в руки до зловмисників.

1.4. Висновки до розділу 1

В цьому розділі був зроблений аналіз проблемної області. Було переглянуто основні можливості блокчейну для захисту файлів, та користувача у системі в цілому. Переглянуто можливість використання хмарного сховища для таких цілей.

РОЗДІЛ 2. ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1. Основні поняття блокчейну

Блокчейн — це серія блоків, об’єднаних у ланцюг. Більшість блокчейнів є транзакційними кінцевими автоматами, які приймають транзакції (вхід) і генерують новий стан (вихід) як показано на рисунку 2.1:

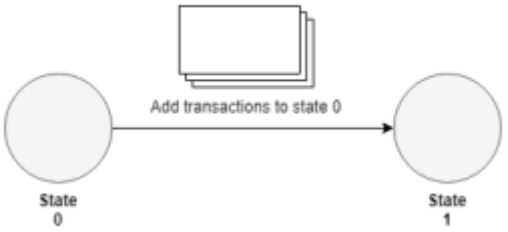

Рисунок 2.1 – Транзакційний стан

Блокчейн або, точніше, технологія розподіленої регістру була створена для підтримки збереження та передачі нової форми валюти під назвою криптовалюта, перша з яких називалася Біткойн. Ця технологія була винайдена особою або групою осіб під псевдонімом Сатоші Накамото [9]. Завдяки шаленому успіху біткойна він став синонімом блокчейна сьогодні. Але технологія блокчейн не обмежується лише необхідною умовою для існування криптовалюти, скоріше це лише одна частина набагато більшої екосистеми, що складається з кілька різних варіантів використання. Лише нещодавно були досліджені випадки використання на основі блокчейну у різних додатках, таких як системи обліку поставок, програми лояльності, обмін даними та нерухомість.

Блокчейн є децентралізованим за дизайном [18] і має однакову інформацію для всіх вузлів, що призводить до того, що це є ресурс, яким володіють усі, без жодної точки відмови.

Аби перевірити транзакції під час використання блокчейну, потрібно обчислити складні криптографічні алгоритми. Це може зробити будь-хто, хто

використовує необхідне програмне забезпечення, і після успішного підтвердження даних транзакцій, яке називається консенсусом, до блокчейну можна додати новий блок. Цей процес називається майнінгом. Успішному майнеру виплачується комісія у вигляді BTC за Біткойн і Ether за Ethereum. Таблиця 2.1 пояснює кілька відмінностей між Bitcoin та Ethereum – двома найбільшими існуючими популярним реалізаціями блокчейну.

Таблиця 2.1 – Порівняння біткойну і ефіріуму

Характеристика	Bitcoin	Ethereum
Дата появи	2009	2015
Винахідник	Сатоші Накамото	Віталій Бутерін
Одиниця валюта	Біткоїн	Ефір
Найменша одиниця	Сатоші	Веї
Символ	btc	eth
Загальна кількість	21 мільйон btc	необмежена
Час на створення блоку	10 хвилин	10-12 секунд
Використання	Система платежів	Система програмування
Винагорода майнера	12.5 btc	3 eth
Алгоритм майнінгу	SHA-256	Ethash

2.1.1. Концепції блокчейну

Блоки генеруються і заповнюються транзакціями. Після заповнення блок передається через мережу та виконується майнінг. Після завершення майнінгу цей блок з’єднується з попереднім блоком, створюючи бухгалтерську книгу. Цей процес можна повторювати нескінченну кількість

разів, щоб створити нескінченний ланцюжок блоків. Процес з’єднання блоків у ланцюг показано на рисунку 2.2:

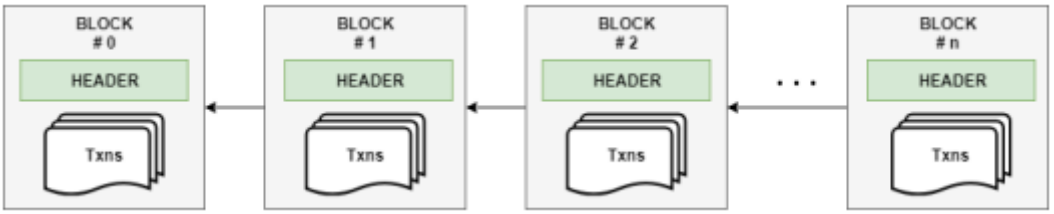

Рисунок 2.2 – Ланцюжок блокчейну

Кожен блок зберігає значення хешу попереднього блоку, якщо хтось змінить вміст історичного блоку, його значення хешу змінюється та не збігатиметься з хешами, збереженими в наступних блоки. Коли цей ланцюжок реплікується між кількома різними користувачами, копія блокчейну особи, яка змінює вміст, не збігатиметься з вмістом інших користувачів, таким чином створюючи постійний незмінний децентралізований ланцюжок блоків.

Однорангове знаходження [14] є важливим питанням будь-якого блокчейну. Завдяки цьому однорангові пристрої знаходять один одного та синхронізують свій стан блокчейну. P2P не є новою технологією, вона і раніше використовувалась у таких програмах, як BitTorrent чи Napster.

Розподілені системи [17] все ще є централізованими і потребують сервера для підключення до кожного клієнта і обмінюватися даними з ним. Блокчейни та інші пов’язані системи розроблені таким чином, щоб бути незалежними від сервера і натомість покладатися на синхронізацію P2P і механізми групового консенсусу для того, аби спілкуватися один з одним і приймати рішення.

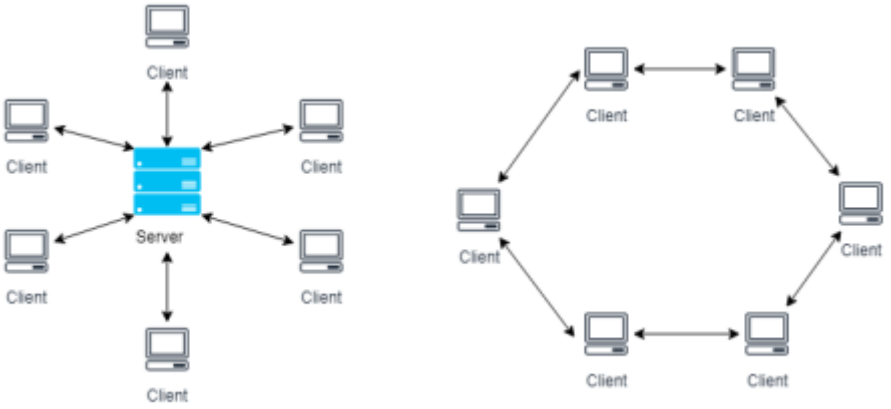

Рисунок 2.3 – Централізовані і децентралізовані мережі

Простіше кажучи, майнінг – це процес виконання інтенсивних математичних або, якщо бути більш точним, криптографічних обчислень. Ці розрахунки складні й вимагають великих обсягів обчислювальної потужності та енергії для роботи. Комп'ютерам, які успішно справляються з цими операціями, виплачується винагорода у вигляді новоствореної валюти, яка раніше не існувала. Це пряма аналогія з видобутком золота з землі. Виконуючи цей процес, майнери перевіряють автентичність платежу та додають перевірені транзакції до мережі. Майнінг – це те, що робить платіжну систему безпечною та надійною без довіреної і централізованої третьої сторони, яка перевіряла б транзакції, як у випадку традиційних методів. Криптографічні обчислення, які виконуються цими майнерами, називаються доказом роботи (PoW). Майнінг є важким, але перевірка є відносно простою і виконується всіма вузлами системи. Підтвердження роботи, необхідне для підтвердження транзакції, називається консенсусом.

Новий блок створюється приблизно за 10 хвилин у біткойні, тоді як у ефіріумі це займає 10-12 секунд. Кожен блок створює 12,5 нових біткойна у біткойні, а ефіріум під час появи нового блоку генерує 3 ефіри.

Термін «розподілені програми» (dapps) використовується для позначення будь-якої програми, яка здатна взаємодіяти та обмінюватися даними з блокчейном. Dapps відрізняються від звичайного веб чи мобільних

застосунків, тому що їх серверна логіка написана та виконується повністю в блокчейні.

2.1.2. Публічні та приватні блокчейни

Біткойн та ефіріум передусім призначені для використання як публічні блокчейни [15]. Це означає, що дані, які зберігаються в реєстрі, видимі для читання та запису. Однак ці мережі можуть бути налаштованим для роботи у вигляді приватних блокчейнів. Це, в основному, означає, що рівень контролю доступу побудований на звичайному блокчейні для здійснення контролю над тим, кому дозволено читати або записувати дані в блокчейн. Ці приватні блокчейни також можуть називається блокчейнами за дозволом.

Таблиця 2.2 – Порівняння публічних і приватних блокчейнів

Публічні блокчейни	Приватні блокчейни
Будь-хто має право приєднатись	Для приєднання потрібен дозвіл
Ніхто не володіє контролем	Контролюється однією людиною
Використовується для валют	Використовується як сховище даних
Велика кількість вузлів	Менша кількість вузлів
Повільніші транзакції	Швидші транзакції
Масштабування обмежене	Масштабованість налаштовується
Підходить для анонімності	Анонімність неможлива

2.2. Блокчейн Ethereum

Ефіріум є популярною реалізацією блокчейну та забезпечує платформу для виконання смарт контрактів [3]. Ці розумні контракти можна використовувати для полегшення проведення грошових операцій, а також для зберігання важливих даних в розподіленому реєстрі [16]. Ефіріум має два типи валют: газ і ефір. Газ – це ефір, який повинні оплачувати вузли, які виконують розумні контракти. Ефір – це криптовалюта в ефіріумі, яку можна використовувати для переказу грошей і оплати газу для роботи смарт контрактів. Ефіріум представляє низку концепцій для надсилання та отримання ефіру, виконання транзакцій смарт контрактів і читання даних як у, так і з блокчейна загалом.

2.2.1. Продуктивність і масштабованість

Як було оголошено раніше, новий блок в ефіріумі генерується кожні 10–12 секунд. Через навмисне обмеження кількості обчислень на блок, кількість транзакцій за секунду обмежена середнім значенням 15 за секунду. Інші методи, такі як затінення та зберігання поза мережею, використовуються для того, аби покращити цей коефіцієнт масштабування.

Витрати на транзакції в газі має нести клієнт ефіріумі, який надсилає транзакцію, вони розраховуються наступним чином:

$$TxCost = gasLimit * gasPrice$$

Середній ліміт газу на блок становить близько 8 000 000. Це значення визначено майнерами та змінюється кожен блок. Із середнім часом створення

блоку 12 секунд і ціною газу за транзакцію в середньому від 20 000 до 60 000, ми можемо очікувати діапазон від 10 до 20 транзакцій на секунду.

2.2.2. Адреса гаманця

Ефіріум використовує ECC (криптографія еліптичної кривої) для генерації відкритих, закритих ключів і ECDSA (Цифрові підписи еліптичної кривої) для перевірки та підписання транзакцій. На рисунку 3.4 показано еліптичну криву під назвою secp256k1, яка використовується в ефіріумі, а також у Bitcoin та інших відомих блокчейнах.

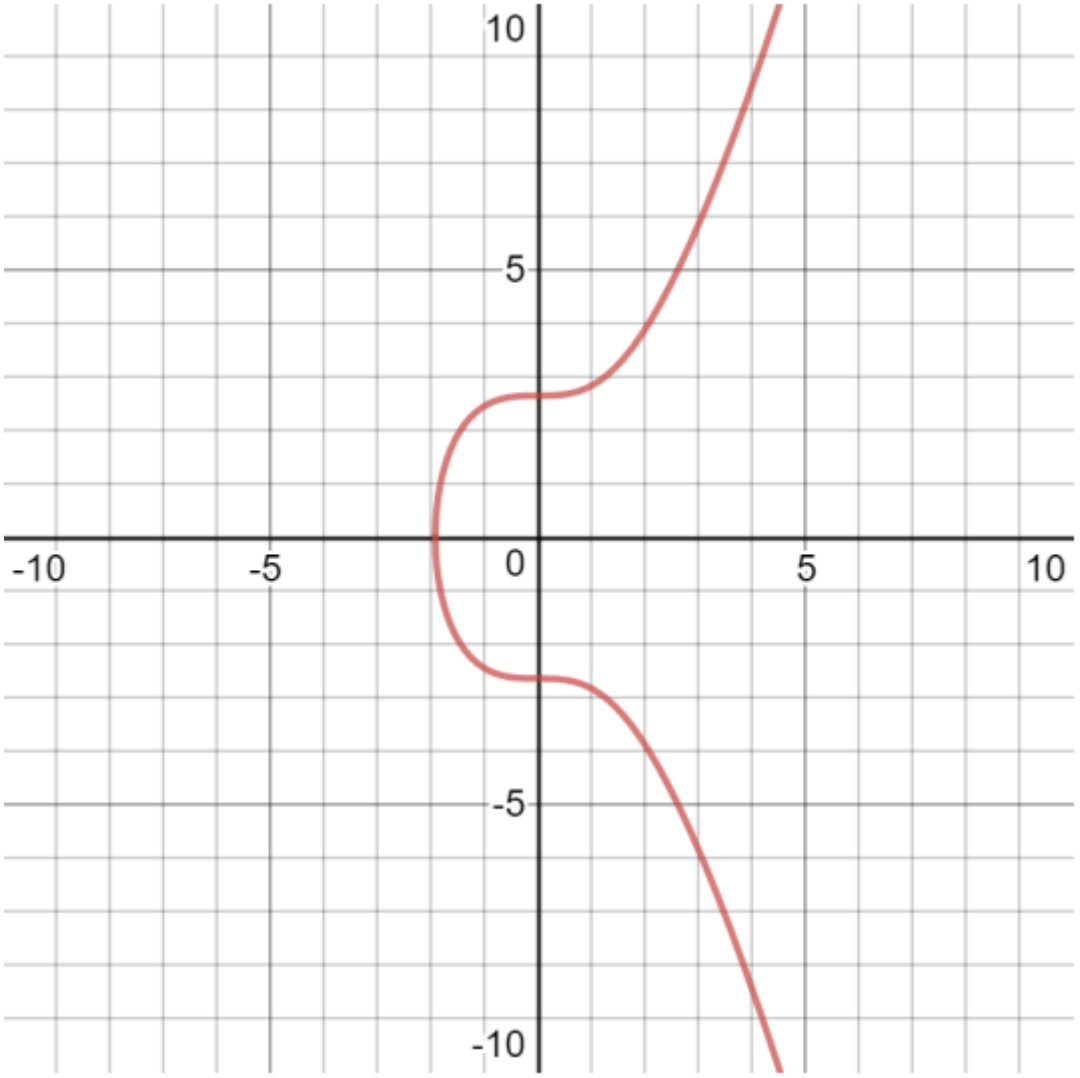

Рисунок 2.4 – Еліптична крива

Вибираються дві точки, і з їх допомогою генерується пара відкритого-приватного ключа. Приватний ключ використовується для підписання транзакцій, тоді як з хешу Keccak-256 (варіант SHA-3) відкритого ключа береться 20 крайніх правих байтів, які реєструються як адреса ефіріуму.

Адреси в Ethereum бувають двох типів: зовнішні облікові записи і облікові записи контрактів.

Зовнішні облікові записи використовуються вузлами, аби надсилати і отримувати ефір. Це найбільш використовувані акаунти.

Контрактні аканти містять код контракту, а їх адреса використовується для виконання смарт контрактів. Вони не мають пов’язаного приватного ключа, і їх адресу можуть викликати клієнти для виконання коду, що зберігається в ньому.

2.2.3. Криптовалюта в Ethereum

Ефіріум платить успішним майнерам криптовалютою під назвою ефір. Її можна поділити на менші одиниці до абсолютно найменшої одиниці, яка називається вей. Ім'я вей є даниною пам'яті одному з перших візіонерів криптовалюти – Вей Даю, який є автором раннього протоколу на основі блокчейну і криптовалюта b-money [8].

Як і будь-яка інша валюта, ефіріум не застрахований від волатильності та має широкий діапазон. Аби запобігти волатильність суми (так званого газу), яку потрібно заплатити за виконання транзакцій в мережі ефіріум, вона не залежить від фактичних курсів обміну ефіру.

Газ можна визначити як спеціальну одиницю, яка використовується в ефіріумі для розрахунку витрат, понесених для виконання цієї транзакції. Ліміт газу має бути зазначений в угоді та повинен бути адекватним. Якщо ліміт занадто низький, малоймовірно, що якісь майнери оброблять цю транзакцію. Після того, як транзакція виконується, майнер, який обробив цю транзакцію, отримує виплату в ефірі відповідно до витраченого газу. Газ, що залишився, повертається на рахунок, який замовив транзакцію.

Таблиця 2.3 показує різні номінали в ефіріумі і їх відповідний курс обміну з ефіром.

Таблиця 2.3 – Номінали в Ethereum

Назва	Доля ефіру	Спеціальна назва	На честь
wei	10^{-18}	-	Wei Dai
kwei	10^{-15}	ada	Ada Lovelace
mwei	10^{-12}	babbage	Charles Babbage
gwei	10^{-9}	shannon	Claude Shannon
micro	10^{-6}	szabo	Nick Szabo
milli	10^{-3}	finney	Harold Finney
ether	1	-	-
kether	10^{3}	einstein	Albert Einstein
mether	10^{6}	-	-
gether	10^{9}	-	-
tether	10^{12}	-	-

2.2.4. Алгоритм майнінгу

Ethereum використовує алгоритм Eth-Hash [14] для пошуку криптографічного хешу для блоку (процес майнінгу). Перед початком майнінгу створюється великий спрямований ациклічний граф (DAG). Процес майнінгу намагається вирішити певну умову в ньому. Цей процес є доказом роботи (PoW) в ефіріумі і призначений бути таким, аби для перевірка іншими вузлами відбувалась дуже швидко в лінійному часі, використовуючи невелику кількість ресурсів.

Nonce вгадується майнерами, щоб створити блок і додати його до ланцюжка. Точно вгадати nonce неймовірно важко, тому значення складності встановлюється і змінюється після кожного блоку. Якщо блок видобуто за менший за середній час видобутку (10 - 12 секунд), складність збільшується, і зменшується, коли час майнінгу перевищує середній час видобутку.

2.2.5. Блок генезису

Початковий блок (genesis block) є першим блоком усього ланцюжка і називається блоком 0 в Ethereum. Він має створюватись вручну для приватних блокчейнів та містити хеш нулів. Цей блок представлено у вигляді файлу JSON і він завантажується першим, коли запускається новий вузол і додається до існуючої мережі ефіріум.

```
{
    "nonce": "0x0000000000000042",
    "mixhash":
    "0x0000000000000000000000000000000000000000000000000000000000000000",
    "difficulty": "0x400",
    "alloc": {},
    "coinbase": "0x0000000000000000000000000000000000000000",
    "timestamp": "0x00",
    "parentHash":
    "0x0000000000000000000000000000000000000000000000000000000000000000",
    "extraData": "0x436861696e536b696c6c732047656e6573697320426c6f636b",
    "gasLimit": "0xffffffff",
    "config": {
        "chainId": 63723,
        "homesteadBlock": 0,
        "eip155Block": 0,
        "eip158Block": 0
    }
}
```

Рисунок 2.5 – Приклад початкового блоку

2.2.6. Режим мережі Ethereum

Ethereum Foundation підтримує два типи публічних мереж. Перший тип – головна мережа з ідентифікатором мережі 1. Існують інші тестові мережі, які розробники можуть використовувати для тестування своїх розподілених програм, створених з використанням ефіру.

Таблиця 2.4 містить коротке пояснення різних основних і тестових мереж.

Таблиця 2.4 – Тестові мережі Ethereum

Мережа	Тип	Ідентифікатор	Статус
main	основна	1	online
morden	тестова	2	retired
ropsten	тестова	3	online
rinkeby	тестова	4	online

Приватні мережі можна налаштовувати як у закритому, так і у відкритому режимі. Усі вузли в цій мережі повинні починати з ідентифікатором, відмінним від тих, що використовуються загальнодоступними мережами. Ще одна вимога полягає в тому, що усі вузли приватної мережі мають бути ініціалізовані одним і тим же блоком генезису. Випадкове число 63723 було обрано для цієї дисертації. Якщо необхідно, аби до приватної мережі не було доступа в будь-кого, хто має ідентифікатором мережі та блоком генезису, можна вимкнути режим виявлення для інших вузлів і натомість додавати їх вручну.

2.2.7. Geth

Geth — це клієнт для ефіріуму, реалізований на Golang, і який використовується для підключення до мережі ефіріум. Існують також інші реалізації на C++ і Python. Після встановлення клієнт може бути ініціалізований за допомоги блоку генезису (перший блок у блокчейні), запущений і підключений до різних мереж, доступних в ефіріумі. Ці мережі ідентифікуються за ідентифікатором мережі. Різні вузли з однаковим блоком генезису та мережевим ідентифікатором можна синхронізувати один з одним

і майнити транзакції. Зарезервовані ідентифікатори мережі: 1 для основної мережі, 2 і 3 для тестових мереж. Для налаштування приватної мережі можна використовувати інший ID мережі. Доступні різні версії для Windows, Linux, MacOS і Android.

Таблиця 2.5 – API Ethereum

API	Команда	Функція
geth	geth attach	Відкрити доступ до мережі
eth	eth.coinbase	Адреса головного акаунту
eth	eth.getBalance	Баланс у вей
eth	eth.sendTransaction	Надіслати ефір у вей
eth	eth.pendingTransactions	Список транзакцій в обробці
web3	web3.fromWei	Конвертація вей в ефір
web3	eth.toWei	Конвертація ефіру у вей
miner	miner.start	Розпочати майнінг
miner	miner.stop	Закінчити майнінг
admin	admin.nodeInfo.enode	Повертає значення ENODE geth везла

2.2.8. Режими синхронізації

Geth має три режими роботи: повний, швидкий та легкий, кожен з яких має своє застосування та завдання.

Повний – у цьому режимі Geth працює за замовчуванням при запуску без будь-яких стартових параметрів. Запитується повний стан бази даних разом із заголовками та тілом, і кожен елемент перевіряється, починаючи з блоку генезису.

Швидкий – цей режим отримує заголовки та тіло блоку, але відкладає обробку будь-яких транзакцій до поточного блоку. Після цього він функціонує так само, як і режим повної синхронізації.

Легкий – цей режим просто використовує блокчейн для поточного стану. Він повинен запитувати повні вузли з попередніх станів, щоб запитувати елементи. Для периферійних пристроїв, таких як Raspberry Pi, де ми не хочемо зберігати багато даних, це ідеальний варіант. Щоб обслуговувати і підтримувати запити від легкого вузла, повний вузол повинен виділяти частину своїх ресурсів.

2.2.9. Ethereum Virtual Machine

На всіх вузлах geth працює віртуальна машина Ethereum Virtual Machine (EVM) – віртуальне середовище, в якому виконується код смарт-контракту. Контракти можуть бути написані на різних мовах програмування, включаючи Solidity, Vyper, LLL та інші, а потім скомпільовані в байт-код EVM.

2.2.10. Інтерфейс

Міжпроцесний зв'язок (IPC) або віддалені виклики процедур (RPC) використовуються в залежності від того, чи хочемо ми зв'язатися з нашим вузлом з одного комп'ютера або з різних комп'ютерів в мережі.

Міжпроцесний зв'язок (Inter-Process Communication, IPC) є кращою технікою підключення до вузла ethereum з процесів, що працюють в одній системі. Єдиний спосіб підключитися до вузла, якщо він запущений без

будь-яких налаштувань – через IPC. Коли ми хочемо підключитися до системи, ми повинні звернутися до файлу geth.ipc, який Geth створює в своєму домашньому каталозі. IPC можна використовувати тільки на localhost і не можна використовувати на будь-яких інших машинах з міркувань безпеки.

Якщо вузол Ethereum працює на іншій системі, слід використовувати віддалений виклик процедур. Geth пропонує HTTP і WebSocket проксі-сервери для з'єднань Ethereum. Крім того, JSON-RPC – це назва формату інтерфейсу для ethereum. Він служить інтерфейсом, через який інтерфейсна програма, побудована на будь-якій мові, може взаємодіяти з бізнес-логікою і даними, що зберігаються в блокчейні.

2.3. Смарт-контракт

Блокчейн Ethereum містить Class-подібний код, який використовується в смарт-контрактах (EVM). Будь-який вузол з достатньою кількістю ефіру, щоб покрити витрати на виконання транзакції, може виконувати методи, описані в смарт-контракті, як транзакцію. Через розподілені додатки громадськість може отримати доступ до цього контракту (також звані dapps). Ці смарт-контракти компілюються за допомогою компілятора Solidity (solc) в байт-код EVM, який містить інструкції, що нагадують машинний код, і ABI Definition, який містить метадані, такі як змінні і методи, що використовуються в смарт-контрактах.

Як і об'єктно-орієнтовані мови, такі як C++ та Java, типовий смарт-контракт має змінні, методи setter та getter, а також модифікатори доступу для управління доступом. Методи сеттерів тягнуть за собою зміну статусу даних в блокчейні, що призводить до плати за транзакцію, яка повинна бути сплачена в газі. Геттери є безкоштовними і дозволяють

отримати тільки ті дані, які зберігаються на всьому вузлі або на одноранговому вузлі.

2.3.1. Solidity

Для написання смарт-контрактів на Ethereum використовується Solidity – мова програмування, орієнтована на контракти повні за Тьюрингом. Це мова високого рівня, яка підтримує всі можливості сучасної мови програмування, включаючи успадкування, статичну типізацію та складні користувацькі типи даних. Це дозволяє нам створювати смарт-контракти, які можуть бути використані для створення розподілених додатків, таких як ведення обліку земельних ділянок, голосування, аукціони, букмекерські ставки та інші.

2.3.2. Remix

За допомогою браузерної IDE Remix можна створювати та тестувати смарт-контракти. У ряді відомих IDE та текстових редакторів також доступні плагіни та фреймворки для створення та тестування смарт-контрактів.

2.3.3. Концепції смарт-контракту

Стандартний смарт-контракт нагадує визначення класу в такій мові, як C++ або Java. Solidity пропонує кілька додаткових функцій для підвищення

зручності використання смарт-контрактів для вирішення різних варіантів використання, на додаток до загальновживаних об'єктно-орієнтованих концепцій, таких як конструктори, методи, змінні, модифікатори видимості, успадкування, масиви та цикли.

Перш ніж віртуальна машина на Ethereum-вузлі зможе отримати доступ і виконати транзакції і методи, зазначені в блокчейні, смарт-контракти повинні бути скомпільовані в байт-код EVM і визначення ABI. Цей компілятор включає в себе Remix і Truffle, дві утиліти Ethereum. Найперший рядок коду solidity містить вказівку на версію, яка була використана при створенні смарт-контракту.

У Solidity змінні мають статичну типізацію і повинні бути визначені тільки один раз. Якщо змінна оголошується у вихідному файлі більше одного разу, компілятор видає помилку. Деякі з найбільш використовуваних типів даних в Solidity описані в Таблиці 3.6.

Таблиця 2.6 – Типи даних у Solidity

Тип	Можливі значення	Приклад
boolean	True, False	True
integer	uint, int	8, -20
address	hex рядок	0x123
String	рядок символів	“foo”

Масиви бувають статичними та динамічними. Використання масивів, як статичних, так і динамічних, не рекомендується, оскільки вони можуть розростатися до величезних розмірів і споживати багато газу при виконанні простих завдань, таких як пошук даних. Статичний масив у наведеному нижче прикладі має розмір 7.

```
uint[] a = new uint[](7);
```

Рисунок 2.6 – Приклад використання масиву

Solidity заохочує використання mapping, а не масивів. Пара ключ-значення з будь-яким типом даних як для ключа, так і для значення називається mapping. Їх можна порівняти з хеш-таблицями, які при оголошенні ініціалізуються, включають всі можливі значення і спочатку відображаються в байтовому представленні тільки нулів. У відображенні зберігається тільки хеш ключа keccak-256, і він використовується для зіставлення відповідного значення ключа. Наступний приклад демонструє, як оголошувати та використовувати відображення.

```
//declare the mapping
mapping(address => uint) public balances;

//store value in a mapping
balances[addressSender] = newBalance;
```

Рисунок 2.7 – Приклад використання mapping

Структури використовуються для зберігання значень різних типів даних і є подібними до структур, що використовуються в мові програмування C. Структури широко застосовуються разом з mapping.

```
//struct to store the temperature, humidity, and timeStamp
struct SensorData{
    uint64 temperature;
    uint64 humidity;
    string dataStorageTime;
}
```

Рисунок 2.8 – Приклад структури

Функціональність конструкторів у соліді ідентична функціональності конструкторів в інших об'єктно-орієнтованих мовах програмування. На

прикладі смарт-контракту, що використовується в даній роботі, створено приклад, наведений нижче.

```
//Constructor for the Smart Contract
constructor() public {
    currentID = 1;
    createdBy = msg.sender;
}
```

Рисунок 2.9 – Приклад конструктора

Задекларовані в смарт-контракті методи можуть бути використані для зміни або отримання поточного стану блокчейну. Однак методи, які діють як сеттери, або змінюють стан блокчейну, не можуть давати вихідні дані. Такі методи повинні викликатися за допомогою web3, який надає тільки хеші їх транзакцій, а витрати на газ повинні оплачуватися в ефірі. методи, що не вносять змін в блокчейн, які тільки повертають статус значення, що зберігається в їх локальній копії блокчейна, без використання газу. Нижче продемонстровано метод, який використовується для встановлення значення.

```
//register IoT device
function registerDevice(address addressToAdd) ownerOnly public {
    if(devicePresent(addressToAdd)) {
        emit deviceEvent(addressToAdd,"DEVICE ALREADY REGISTERED");
    }
        trustedAddresses[addressToAdd] = true;
        emit deviceEvent(addressToAdd,"SUCESSFULLY REGISTERED");
}
```

Рисунок 2.10 – Приклад методу

Події – це унікальні методи, які смарт-контракт може видавати при виконанні певної умови. Наприклад, сигналізувати про завершення успішної транзакції або натякнути на помилку. Ці події можуть прослуховуватися

іншим смарт-контрактом або з інших джерел за допомогою web3. Приклад події наведено нижче.

```
//Transaction successful
event setFileHashEvent(
    address indexed _from,
    string _message
);

//Event triggered using emit keyword
emit setFileHashEvent(msg.sender, "FILEHASH TXN CALLED");
```

Рисунок 2.11 – Приклад івентів

Коли ми хочемо заблокувати певній групі користувачів доступ до блокчейну, ми використовуємо модифікатори. Використовуючи модифікатори, смарт-контракт може побудувати фундаментальний тип контролю доступу. До методів, які виконуються тільки в тому випадку, якщо вимоги в модифікації виконані, можуть бути застосовані ці модифікатори. Нижче наведено приклад модифікатора.

```
//Modify some functions to be executed only by Contract creator
modifier ownerOnly {
    require(msg.sender == createdBy);
    _;
}
```

Рисунок 2.12 – Приклад використання модифікатора

2.3.4. Web3

Розробник може створити клієнт (веб-додаток або мобільний додаток), який може безпосередньо взаємодіяти з блокчейном Ethereum, використовуючи фреймворк Web3, який був створений Ethereum Foundation. Функціонал, який пропонує web3, включає в себе передачу ефіру на інший вузол, доступ до адреси гаманця поточного вузла тощо. Крім того, він може бути використаний для підключення до смарт-контракту, який був створений на блокчейні, і виконання інструкцій, які він містить. Ці дії можуть включати що завгодно - від переміщення ефіру між сторонами до віддачі голосу за певного кандидата на виборах або передачі права власності на землю від однієї особи до іншої. Web3 підтримує Javascript, Python і багато інших мов. Найбільш відомим фреймворком, що дозволяє користувачам підключати веб-сторінки до деяких внутрішніх функцій смарт-контрактів, є, безумовно, Web3.js. Python та C добре підтримуються більшістю IOT-пристроїв. Для цієї роботи було обрано web3.py, а не web3.js в якості інтерфейсу між пристроями IoT та блокчейном Ethereum.

2.3.5. Truffle

Фреймворк під назвою Truffle дозволяє швидко створювати та впроваджувати смарт-контракти в Ethereum. Truffle також може бути використаний для інших завдань, таких як отримання адреси та визначення ABI смарт-контракту. Кілька інструкцій, які дозволяють Truffle створювати і запускати смарт-контракти, включені в Таблицю 3.7.

Таблиця 2.7 – Приклади команд Truffle

Команда	Функція
truffle compile	Компілює смарт-контракт у байт-код

truffle migrate	Розгортає смарт-контракт в мережі
truffle console	Отримує доступ до метаданих контракту

Таблиця 2.8 – Приклади команд Truffle console

Команда	Функція
ContractName.address	Повертає адресу смарт-контракту
ContractName.abi	Повертає abi метадані смарт-контракту

2.3.6. Ganache

Фреймворк під назвою Ganache може бути корисним для тестування смарт-контрактів під час їх розробки. Він пропонує завантажений приватний блокчейн з ефіром, який вже завантажений на тестові акаунти.

2.4. Альтернативні рішення децентралізованих сховищ

Зберігання даних в Ethereum можливе за допомогою змінних смарт-контрактів. Вартість такого зберігання висока, і вартість додавання транзакцій в блокчейн також зростає в міру накопичення обсягу даних.

Ще одним суттєвим недоліком використання Ethereum як ексклюзивного способу зберігання даних є те, що на змінні в Solidity накладаються вкрай жорсткі обмеження. Це зроблено навмисно, оскільки Ethereum не рекомендує зберігати великі обсяги даних в блокчейні, щоб запобігти зловживанням, які можуть задушити мережу і призвести до відмови

в обслуговуванні та інших видів мережевих атак. Всі дані повинні бути репліковані на всіх вузлах, однак наш варіант використання не вимагає такого рівня високої надмірності.

Дані можуть зберігатися в інших децентралізованих системах зберігання, таких як Inter Planetary File System або Ethereum Swarm, щоб обійти ці обмеження. У цій тезі ми порівняємо продуктивність при використанні обох цих стратегій. Siacoin [14], storj [14] та інші рішення для зберігання даних є додатковими варіантами, які знаходяться на різних стадіях розвитку.

Для створення децентралізованого рівня зберігання даних в Інтернеті з використанням безсерверної архітектури хостингу хорошими альтернативами є Swarm і IPFS. Вони також пропонують блокову архітектуру зберігання, в якій великі документи обслуговуються блоками і можуть бути отримані паралельно. Це пропонує рівень стимулювання для вузлів-учасників, щоб гарантувати, що матеріали, які зберігаються в них, не будуть стерті. Вони пропонують децентралізоване вирішення доменних імен і захист цілісності адресації контенту (файловий хеш).

2.4.1. Ethereum Swarm

Рішення для зберігання даних Ethereum Swarm [17] пропонує використовувати протокол Peer to Peer (P2P), подібний до BitTorrent, для децентралізованого зберігання даних на великій кількості вузлів. Він використовує багато технологій, розроблених фондом Ethereum, і є компонентом інструментарію Geth. Поточною реалізацією є версія 0.3.x, також відома як Proof of Concept 3.

Для доступу до даних, що зберігаються між вузлами, він пропонує HTTP-проксі. Файли swarm повертаються з хешем файлу, який унікально ідентифікує ресурс.

Його основна мета – дати dapps можливість ефективно зберігати дані для таких цілей, як обмін повідомленнями, потокова передача даних і змінні модифікації ресурсів. Swarm є гідним варіантом для цього аргументу, оскільки Ethereum Foundation явно підтримує його.

2.4.2. Inter Planetary File System

Міжпланетна файлова система (IPFS) [14] - це система зберігання даних, яка вирішує багато з тих же проблем, з якими стикається Ethereum Swarm. Однак вона значно простіша у налаштуванні та використанні, ніж Ethereum, і існує набагато довше.

Для інформації, що зберігається в мережі IPFS, він надає хеш файлу, подібний до Swarm. Він пропонує інший протокол для доступу до контенту, що зберігається в його вузлах, на відміну від Ethereum Swarm. Крім того, він пропонує користувальницький інтерфейс для графічного доступу до файлів, що зберігаються на вузлі.

2.4.3 Порівняння сховищ

Рисунок 2.13 показує просте порівняння між різними рішеннями для зберігання даних, включаючи звичайні бази даних.

Comparison	IPFS	Swarm	RDBMS	NoSQLDB
Decentralization	Yes	Yes	No	No
Distributed	Yes	Yes	Yes	Yes
Single Point of Failure	No	No	Yes	Yes
Is Production Ready	Yes	No	Yes	Yes
Privacy of Data	No	No	Yes	Yes
Speed	Slow	Slow	Fast	Fast
Flexibility of Data Format	Flexible	Flexible	Not Flexible	Flexible

Рисунок 2.13 – Порівняння різних сховищ даних

Рисунок 2.14 показує технічні відмінності між IPFS та Swarm.

Comparison	**IPFS**	**Swarm**
Storage Component	Distributed Hash Tables	Immutable Content Addressed chunkstore
Cloud Hosting Like Service	No	Yes
Integration with Ethereum	None	Full
Network Layer	libp2p	devp2p
Incentivation Layer	FileCoin	Ethereum

Рисунок 2.14 – Технічні відмінності IPFS і Swarm

2.5. Висновки до розділу 2

У даному розділі були дані визначення поняттям «блокчейн», «смарт-контракт», «ефір» та «майнінг», описано існуючі концепції блокчейнів загалом та конкретно Ethereum. Також було розглянуто існуючі рішення збереження даних.

Також для подальшої розробки системи було розглянуто ряд основних інструментів, які стануть корисними в роботі з блокчейном.

РОЗДІЛ 3. ОПИС ПРОГРАМНОГО ПРОДУКТУ

3.1. Використані інструменти

Під час розробки програмного продукту були використані наступні інструменти: Truffle, Remix, Ganache, Infura.

Truffle, покликаний спростити життя розробників, описується як "середовище розробки світового класу, тестовий фреймворк і конвеєр активів для блокчейнів, що використовують віртуальну машину Ethereum (EVM)".

Простіше кажучи, Truffle – це інтегроване середовище розробки, тестовий фреймворк і конвеєр активів. Він побудований на блокчейні Ethereum і призначений для того, щоб зробити створення DApps (розподілених додатків) простим і зрозумілим. За допомогою Truffle можна створювати інтерфейси для DApps, а також компілювати і розгортати смарт-контракти, вбудовувати їх у веб-додатки. Зараз Truffle є однією з найпопулярніших IDE для блокчейна Ethereum.

Звичайно, визначення - це перше, на що варто було б звернути увагу в інструкції до Remix IDE. Безсумнівно, це найбільш надійний і широко використовуваний компілятор для програмування смарт-контрактів Solidity. Однак, також важливо пам'ятати, що Remix - це більше, ніж просто компілятор. Remix - це інтегроване середовище розробки, або IDE, яке може допомогти в написанні, компіляції та налагодженні програм Solidity. Одним з найулюбленіших інструментів для розробки dApps та web3 є Remix IDE.

Remix IDE – це, по суті, програма з відкритим вихідним кодом з гнучким доступом до численних корисних плагінів та зручним графічним інтерфейсом. Використовуючи мову програмування Solidity, вона може бути незамінним помічником для девелоперів протягом усього життєвого циклу розробки смарт-контрактів.

Ganache взаємодіє з Truffle – це перше, що потрібно розуміти про Ganache. Насправді, Ganache є частиною фреймворку Truffle Suite, разом з Truffle і Drizzle. Truffle, заснований на віртуальній машині Ethereum, служить середовищем розробки, тестовим фреймворком і конвеєром активів. Drizzle, з іншого боку, надає вибір різних інтерфейсних бібліотек. Що таке Ganache в блокчейні? У відповіді підкреслюються можливості Ganache як висококласного інструменту розробки, який можна використовувати для управління власною локальною мережею блокчейн для створення децентралізованих додатків на Ethereum і Corda. Всі фази процесу розробки отримують значну користь від ganache, який пропонує ряд мислимих переваг.

Можливість створювати, тестувати і розгортати свої проекти смарт-контрактів і децентралізованих додатків в передбачуваному і безпечному середовищі - одна з головних переваг Ganache. Залежно від необхідної вам функціональності, ви можете вибрати між двома альтернативними версіями Ganache. Десктопна програма, яка може підтримувати завдання розробки Corda і Ethereum, називається Ganache UI. З іншого боку, у вас є інструмент командного рядка Ganache-CLI, який присвячений розробці Ethereum. Також варто відзначити доступність Ganache на Linux, Mac і Windows для обох версій.

Infura – це постачальник інфраструктури як послуги (IaaS) і бекенда Web3, який надає розробникам блокчейну доступ до різноманітних додатків та інструментів. Частиною цього є набір Infura API (Application Programming Interface - інтерфейс прикладного програмування). Ядром сервісу Infura Web3 є флагманський Ethereum API компанії. Втім, в планах - підключення міжпланетної файлової системи (IPFS) та Filecoin. Тим не менш, деякі замінники Infura на даний момент забезпечують більшу міжланцюгову зв'язність. Незважаючи на те, що Ethereum залишається найкращим програмованим блокчейном для розробки децентралізованих додатків (dApps), багато розробників блокчейнів вже шукають альтернативи Infura в

інших місцях. Це відбувається одночасно зі зростанням популярності Polygon Network та Binance Smart Chain (BSC) (раніше Matic Network).

Infura – це пакет для розробки блокчейн, який пропонує інтерфейси прикладного програмування (API) і інструменти для розробників, і він представлений вам відомою компанією ConsenSys, яка займається розробкою програмного забезпечення для блокчейн. Крім того, Infura надає розробникам швидкий і надійний доступ до мережі Ethereum, щоб вони могли створювати складні Web3-додатки і програмне забезпечення нового покоління, яке масштабується відповідно до попиту користувачів.

Крім того, як постачальник інфраструктури як послуги (IaaS) і внутрішньої інфраструктури Web3, Infura надає першокласні ресурси і документацію, щоб допомогти розробникам швидко створювати децентралізовані додатки (dApps). Це досягається за рахунок скорочення часу, необхідного для створення інфраструктури з нуля. Використовуючи розподілену мережу вузлів, розміщену в хмарі, Infura забезпечує інфраструктуру, яка підходить для підприємства. В результаті значно зменшуються складності, які виникають при створенні та володінні ексклюзивними обчислювальними засобами та засобами зберігання даних.

3.2. Запропонована архітектура система

Систему було побудовано, використовуючи блокчейн Ethereum та IPFS. IPFS використовується для збереження користувацьких файлів у розподіленій системі у той час, як блокчейн використовується для ведення реєстру існуючих у системі файлів, збереження їхніх метаданих, таких як хеші файлів, адреси власників та використовувані симетричні ключі кодування.

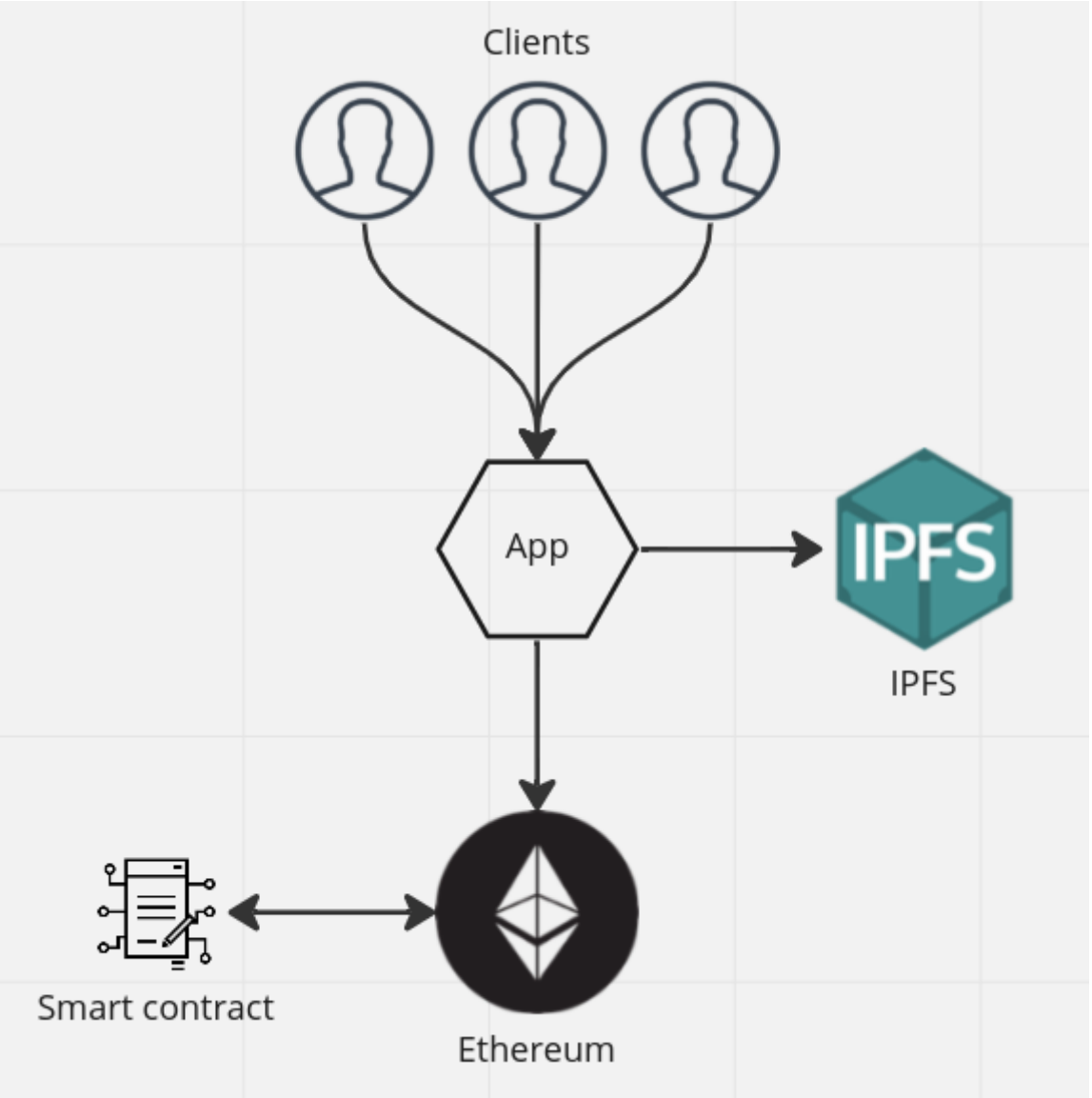

Рисунок 3.1 – Схема архітектури

3.3. Використані алгоритми криптографії

3.3.1. Криптографія з відкритим ключем

Шифрування даних здійснюється за допомогою криптографії з відкритим ключем, яка також слугує засобом перевірки автентичності повідомлень, з використанням ключової пари з відкритого та закритого ключів.

Алгоритм RSA був створений Роном Рівестом, Аді Шаміром та Леонардом Адлеманом і використовує комбінацію відкритого та закритого ключів для шифрування та дешифрування даних. Крім того, він може використовуватися для підтвердження підписів.

Приватний ключ одержувача може бути використаний для розшифрування зашифрованих даних після того, як вони були зашифровані за допомогою відкритого ключа одержувача на пристрої, який їх згенерував. Він зазвичай використовується для доставки симетричних ключів шифрування через обмеження розміру корисного навантаження.

3.3.2. Симетричне шифрування

Дані шифруються і розшифровуються за допомогою симетричного шифрування з використанням одного і того ж секретного ключа. Великі обсяги даних шифруються або розшифровуються за допомогою симетричного шифрування з використанням симетричного ключа, що, як правило, швидше, ніж криптографія з відкритим ключем. Недоліком є те, що секретний ключ потрібно постійно змінювати та надійно зберігати.

Фактичні дані шифруються за допомогою вдосконаленого стандарту шифрування (Advanced Encryption Standard, AES), який використовує надійний секретний ключ. Цей алгоритм використовується для шифрування та зберігання даних в IPFS або Swarm. Криптографія з відкритим ключем була взята до уваги для цієї дипломної роботи, оскільки запропонована система вимагає, щоб багато пристроїв використовували один і той самий ключ.

3.3.3. Хешування

Хешування передбачає створення групи символів, здатних конкретно ідентифікувати вхідні дані.

Вони, як правило, використовуються в криптографії для створення підпису для великих вхідних даних і шифрування за допомогою приватного ключа відправника, який може бути розшифрований тільки за допомогою відкритого ключа відправника, що підтверджує легітимність відправника. Дайджест повідомлень (серія MD), безпечні алгоритми хешування (серія SHA) та код автентифікації хеш-повідомлень (HMAC) - це деякі з алгоритмів хешування, які часто використовуються.

У цій роботі дані аутентифікуються на стороні отримання та підписуються на стороні виробництва за допомогою хеш-методу (HMAC). Той самий процес, який використовується для передачі ключа AES, може також використовуватися для аутентифікації користувачів та перевірки їх підписів за допомогою секретного спільного ключа.

3.4. Аналіз отриманих даних

Продуктивність системи буде оцінюватися з точки зору кількості транзакцій в секунду (TPS), вартості транзакції (CPT), використання процесора і пам'яті після того, як мережа буде сконфігурована. Для тестування TPS створюється набір даних з 5 000 записів.

Як Ethereum, так і Swarm/IPFS використовуються в трьох типах експериментів:

1. Таймінги запису записуються і зберігаються в Swarm/IPFS. Для вимірювання продуктивності читання хеші файлів зберігаються в текстовому файлі.
2. Отриманий хеш файлу передається в Ethereum за допомогою функції смарт-контракту, а дані зберігаються в Swarm/IPFS. Фіксується час читання і запису.
3. Swarm/IPFS зберігає дані в зашифрованому вигляді. Отриманий хеш файлу потім зберігається і використовується за необхідності. У всіх тестах використовується алгоритм шифрування AES-CBC з підкладкою PKCS. Розмір ключа становить 256 біт.

Ми порівняємо час, необхідний для завантаження і читання цих записів з шифруванням і без нього, а також витрати, які несе рахунок, що подає транзакції.

3.4.1. Витрати на впровадження смарт-контрактів

На вартість смарт-контрактів впливають численні фактори. Крім фіксованої вартості, необхідно сплачувати комісію за зберігання та обчислення. На вартість розгортання дещо впливає рівень оптимізації.

У Truffle повинен бути розгорнутий міграційний контракт і реальний. Власник смарт-контракту, що розгортається, та часова мітка останнього успішного розгортання визначаються за допомогою міграційного контракту. Код для реєстрації, запису та зчитування даних з блокчейну є частиною FileContract.

Вартість розгортання смарт-контракту в приватних блокчейнах становить 20 гвей за одиницю використаного газу. Тільки в тому випадку,

якщо ми хочемо розгорнути ідентичний контракт в основну мережу Ethereum, ця вартість перевіряється.

Сумарно розгортання смарт-контрактів обійшлося в 0,0232 Ефіру.

3.4.2. Порівняння ефективності

І Swarm, і IPFS використовують тип хешування для зберігання даних, тому читання або запис в систему лінійно масштабується з часом. В цілому, читання і запис в IPFS або Swarm займає приблизно однаковий час. Швидкість запису часто повільніша, ніж швидкість читання. Оскільки тестування проводилося через мережу, мережеві затримки читання і запису маскують будь-які відмінності між реальними операціями читання і запису в ці системи зберігання даних.

У цьому розділі порівнюється час читання та запису для різних систем зберігання даних, які ми використовували, в тому числі з шифруванням, дешифруванням та перевіркою підпису, та без них.

Таблиця 3.1 відображає різні показники, які були зафіксовані під час тестів на запис для 5000 транзакцій у запропонованій системі. Якщо вартість виражена в ефірі, то витрачений час вимірюється в секундах.

Таблиця 3.1 – Результати для тестів на запис для 5000 транзакцій

Тест	Час	TPS	Витрати	Витрати на транзакцію	Середнє використання ЦП %	Середнє використання ОЗП %
Swarm	149 с.	33.55	-	-	4.6	57.2
Swarm + Encryption	150 с.	33.33	-	-	6.3	68.9

IPFS	2755 с.	1.81	-	-	90.1	5.6
IPFS + Encryption	3777 с.	1.32	-	-	91.4	5.8
Ethereum + Swarm	895 с.	5.58	0.46	0.000093975	48.7	65.6
Ethereum + IPFS	4039 с.	1.23	0.46	0.000093975	90.4	27.3
Ethereum + Swarm + Encryption	920 с.	5.43	0.46	0.000093975	56.8	50.3
Ethereum + IPFS + Encryption	5695 с.	0.87	0.46	0.000093975	99.8	27

Таблиця 3.2 відображає різні метрики, отримані в результаті тестів на читання для 5000 транзакцій на запропонованій системі.

Таблиця 3.2 – Результати тестів на читання для 5000 транзакцій

Тест	Час	TPS	Середнє використання ЦП %	Середнє використання ОЗП %
Swarm	151 с.	33.11	40.1	21.5
Swarm + Decryption	156 с.	32.05	45.4	45.7
IPFS	379 с.	13.19	23.9	5.7
IPFS + Decryption	426 с.	11.73	26.7	5.8
Ethereum + Swarm	556 с.	8.99	32.3	26.8
Ethereum + IPFS	456 с.	10.96	37	25.9

Ethereum + Swarm + Decryption	622 с.	8.03	34.5	29
Ethereum + IPFS + Decryption	783 с.	6.28	39.9	26.7

3.4.3. Різниця в продуктивності між Swarm та IPFS

При повній реалізації запропонованої нами системи (Ethereum + Swarm) ми змогли завершити тест на запис зі швидкістю близько 5 TPS і тест на читання зі швидкістю близько 9. Це легко досяжно, враховуючи наше передбачуване використання однієї транзакції на пристрій щохвилини. Цей метод буде зручним у використанні для систем, які відправляють транзакції на низькій або середній частоті.

Оскільки завантаження процесора для IPFS під час тесту на запис було дуже високим у порівнянні з Swarm, використання IPFS замість Swarm призвело до набагато гіршої продуктивності. Ми маємо TPS 16 (5000/306) для запису і 71 (5000/70) для читання.

3.4.4. Вартість запису

У порівнянні з використанням як Swarm, так і блокчейну Ethereum, зберігання даних виключно за допомогою Swarm було неймовірно швидким. Додавання шифрування мало вплинуло на продуктивність. Витрати на продуктивність при зберіганні тестового набору даних з використанням лише

Swarm, IPFS, Ethereum та Swarm, а також Ethereum та IPFS показано на рисунку 6.14.

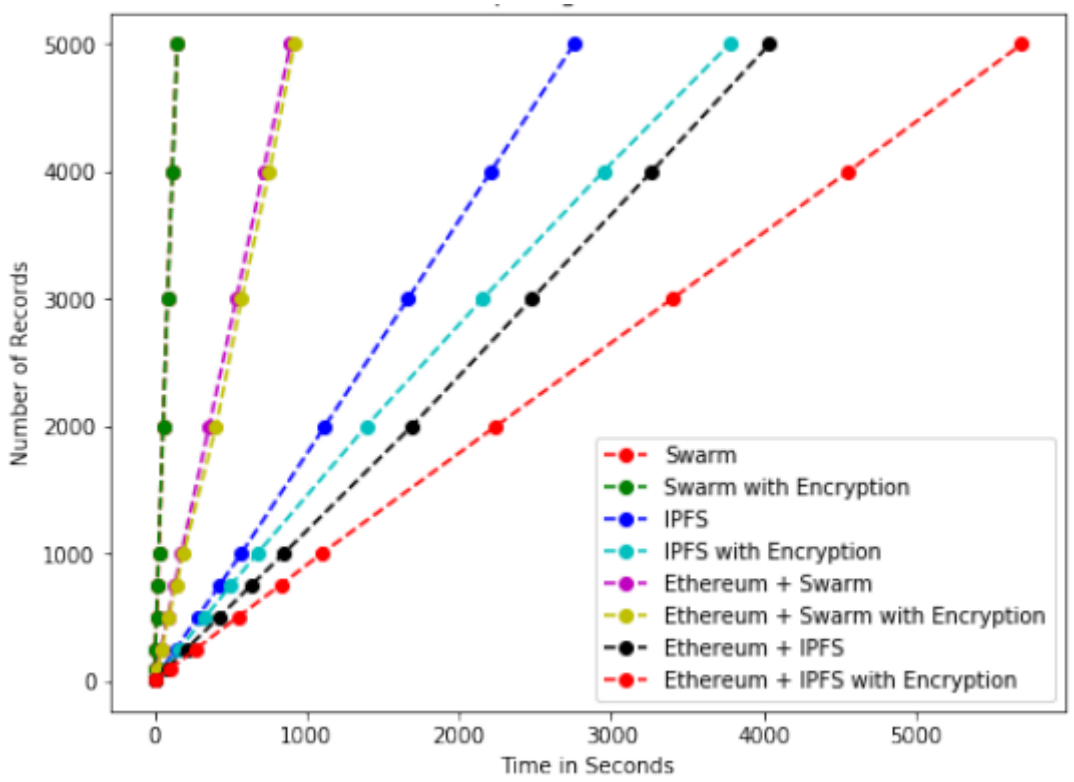

Рисунок 3.2 – Порівняння часу запису

3.4.5. Вартість читання

У порівнянні з використанням блокчейну Ethereum і технології Swarm разом, читання даних Swarm було неймовірно швидким. При додаванні шифрування продуктивність дещо знизилася, оскільки отримувач повинен згенерувати і підтвердити підпис. Однак при використанні IPFS і блокчейна Ethereum погіршення продуктивності значно гірше. Вартість продуктивності при використанні просто Swarm, IPFS, Ethereum і Swarm, а також Ethereum і IPFS для зчитування тестового набору даних показана на малюнку 6.16.

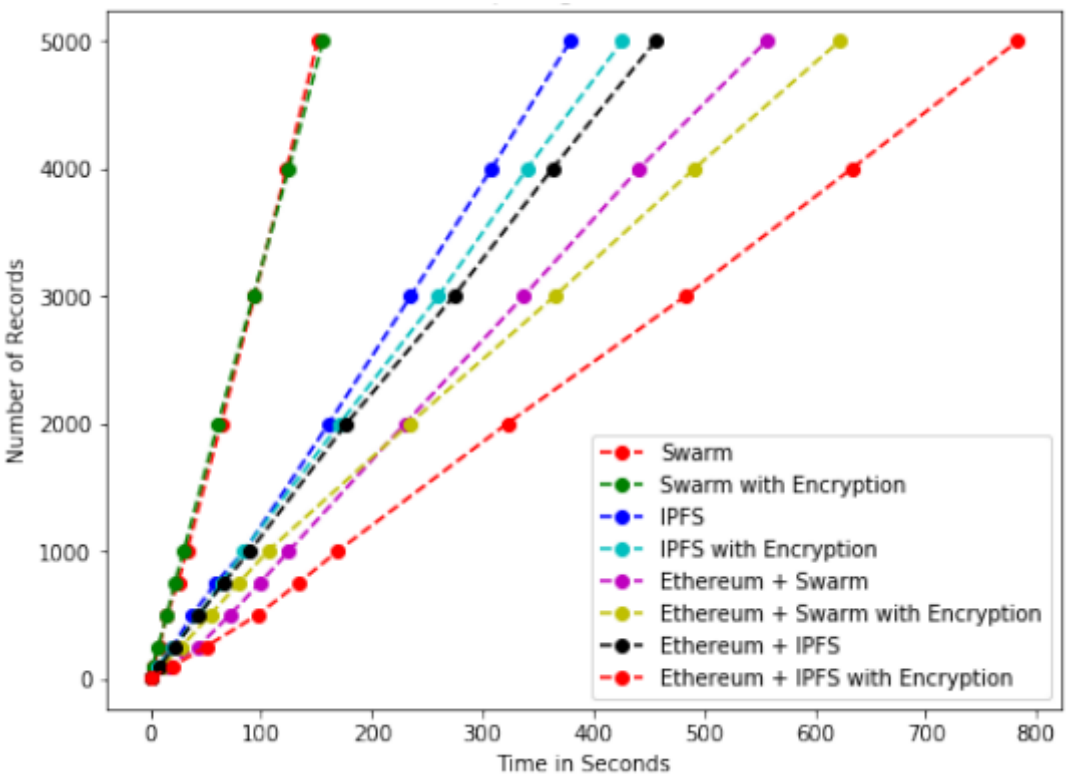

Рисунок 3.3 – Порівняння часу читання

3.5. Висновки до розділу 3

У цьому розділі було розглянуто, протестовано та порівняно декілька варіацій запропонованої системи. Розглянуто і реалізовано можливість шифрування даних для їх безпечного збереження.

З точки зору продуктивності, IPFS перевершила комбінація Ethereum (для підтримки порядку записів) і Swarm (для реального зберігання).

Та ж сама техніка зберігання даних може застосовуватися для інших випадків використання, включаючи безпечне ведення записів в таких галузях, як установи державного правління, освіти, охорони здоров'я і т.д., а також розширення ємності сховища в міру необхідності без необхідності тимчасово припиняти надання послуг для технічного обслуговування.

Однак, Swarm і IPFS все ще знаходяться у процесі розробки, і ще не готові до використання в режимі реального часу обробки великих даних.

Спростити та убезпечити процес налаштування та використання систем допомогла б розробка методів шифрування даних, що зберігаються у відповідних системах, що дозволило б ще більше спростити та убезпечити процес налаштування та використання цих систем. У той час як Ethereum та IPFS пропонують ефективні сервіси моніторингу для відстеження продуктивності мережі, Swarm ще не реалізував засоби для перегляду мережевого трафіку, статусу однорангових мереж тощо.розробляють свої власні шари стимулювання [16], подібні до Ethereum.

РОЗДІЛ 4 РОЗРОБКА СТАРТАП-ПРОЕКТУ

Останні роки довели, що стартапи сильніші, ніж більшість думала. Гнучкий та інноваційний підхід були ключовими, коли світ зіштовхнувся з такими проблемами, як віддалена робота, масштабні зміни в галузі та ринку, а також абсолютно нова реальність.

Здатність стартапів швидко змінюватися та адаптуватися і є ключовою властивістю даного виду бізнесу, не кажучи вже про те, що багато стартапів продовжували рости та розширювати свої команди.

Стартап — це бізнес-структура, що розвивається та працює на основі інновацій, створена для вирішення проблеми шляхом надання нової пропозиції в умовах надзвичайної невизначеності.

Власне, стартап – це бізнес, який:

- швидко росте;
- порушує ринок або галузь (щось нове, що змушує конкурентів удосконалюватись);
- вирішує проблему;
- працює в умовах надзвичайної невизначеності.

Багато підприємців і відомих бізнес-магнатів визначають стартап як культуру та менталітет побудови бізнесу на основі інноваційної ідеї для вирішення критичних проблем.

Одне, що відрізняє стартапи від інших компаній — це зв’язок між їхнім продуктом та його попитом. Стартапи мають продукти, орієнтовані на невикористаний ринок. Підприємці-початківці знають ідеальну стратегію, щоб створити продукт, який хоче ринок, а також охопити й обслуговувати їх усіх. Це викликає швидке зростання. [19].

1.1 Опис ідеї проекту

В межах підпункту було проаналізовано і подано у вигляді таблиць:

- зміст ідеї;
- можливі напрямки застосування;
- можливі напрямки застосування;
- основні вигоди, що може отримати користувач товару;
- чим відрізняється від існуючих аналогів та замінників.

Перші три пункти подані у вигляді таблиці (таблиця 4.1) і дають цілісне уявлення про зміст ідеї та можливі базові потенційні ринки, в межах яких потрібно шукати групи потенційних клієнтів.

Таблиця 4.1 — Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Дана комплексна система дозволяє розвязати проблему видачі контекстуальних рекомендацій.	Видача рекомендацій користувачу	Збільшення прибутку шляхом покращення якості рекомендацій
	Генерування динамічних вкладень графу взаємодій	Збільшення об’єму клієнської бази

Аналіз потенційних техніко-економічних переваг ідеї (чим відрізняється від існуючих аналогів та замінників) порівняно із пропозиціями конкурентів передбачає:

- визначення переліку техніко-економічних властивостей та характеристик ідеї;
- визначення попереднього кола конкурентів (проектів-конкурентів) або товарів-замінників чи товарів-аналогів, що вже існують на

ринку, та проводиться збір інформації щодо значень техніко-економічних показників для ідеї власного проекту та проектів-конкурентів відповідно до визначеного вище переліку;

- проводиться порівняльний аналіз показників: для власної ідеї визначаються показники, що мають:
  - гірші значення (W, слабкі);
  - аналогічні (N, нейтральні) значення;
  - кращі значення (S, сильні) (табл. 4.2).

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Технікоекономічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Adobe Target	Amazon Personalize			
1	Форма виконання	Надавання послуг	Надавання послуг	Надавання послуг		+	
2.	Собівартість	Низька	Висока	Висока			+
3.	Функціонал	Широкий	Широкий	Широкий	+		

Визначений перелік слабких, сильних та нейтральних характеристик та властивостей ідеї потенційного товару є підґрунтям для формування його конкурентоспроможності.

1.2 Технологічний аудит ідеї проекту

В межах даного підрозділу було проведено аудит технології, за допомогою якої можна реалізувати ідею проекту (технології створення товару). Визначення технологічної здійсненності ідеї проекту передбачає аналіз таких складових (таблиця 4.3):

1. За якою технологією буде виготовлено товар згідно ідеї проекту?
2. Чи існують такі технології, чи їх потрібно розробити/доробити?
3. Чи доступні такі технології авторам проекту?

Таблиця 4.3 – Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Створення системи генерування рекомендацій користувачу за даними покупок користувачів	Використання мови програмування Python	Наявна	Доступна
		Використання мови програмування C#	Відсутні	Недоступні

		Використання мови С++	Відсутні	Недоступні
Обрана технологія реалізації ідеї проекту: мова програмування Python.				

За результатами аналізу таблиці зроблено висновок щодо можливості технологічної реалізації проекту. Технологічним шляхом реалізації проекту було обрано такі технології, як Python через доступність та безкоштовність.

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Спочатку було проведено аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку (таблиця 4.4).

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	50
2	Загальний обсяг продаж, грн/ум.од	10000
3	Динаміка ринку	Зростає
4	Наявність обмежень для входу	Немає
5	Специфічні вимоги до стандартизації та сертифікації	Немає

6	Середня норма рентабельності в галузі, %	20%

За результатами аналізу таблиці 4.4 було зроблено висновок, що ринок є привабливим для входження.

Надалі були визначені потенційні групи клієнтів, їх характеристики та сформовано орієнтовний перелік вимог до товару для кожної групи (табл. 4.5).

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проекту

Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
Точна та швидка генерація рекомендацій цільовій аудиторії	Власник бізнесу	Велика кількість даних	Простота використання, висока точність
Підбір рекомендацій	Кінцевий користувач	Цікавить ростота у використанні, низька ціна підтримки системи	Швидкість створення, низька ціна

Після визначення потенційних груп клієнтів було проведено аналіз ринкового середовища: складено таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (табл. 4.6, 4.7).

Таблиця 4.6 – Фактори загроз

№ п/ п	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренція	Вихід на ринок продуктів з кращими характеристиками	Передбачити додаткові переваги власного програмного продукту (ПП) для того, щоб повідомити про них саме після виходу на ринок конкурентів. Вдосконалення технічних моментів власного продукту. Обрати нову цільову аудиторію і зосередитися на ній: зниження цін.
2	Зміна потреб користувачів	Користувачам необхідний сервіс з більшим/новим функціоналом.	Розроблення гнучкої архітектури програмного забезпечення для легшого впровадження нового функціоналу

Таблиця 4.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Гнучкі ціни	Зменшення ціни товару задля збільшення попиту	Введення власних гнучких цін
2	Поява нових методів квантування зображення	З’являться нові методи, що будуть швидше, та більш точно квантувати зображення	Покращити ПП додаванням нового функціоналу, розширення можливостей

Надалі було проведено аналіз пропозиції: визначили загальні риси конкуренції на ринку (таблиця 4.8).

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства
1. Тип конкуренції- чиста	Існує величезна кількість конкурентів на ринку.	Якісно провести рекламу.
2. За рівнем конкурентної боротьби - міжнародний	Компанії-конкуренти з інших країн	Створити основу ПП таким чином, щоб можна було легко переробити даний ПП для використання у галузях інших країн.
3. За галузевою ознакою - міжгалузева	Продукт може використовуватись для різних галузей	Постійне вдосконалення продукту, що не має прив’язки до сфери
4. Конкуренція за видами товарів: - товарно-видова	Конкуренція між видами ПП, їх особливостями.	Створити ПП, враховуючи недозображення конкурентів
5. За характером конкурентних переваг - нецінова	Вдосконалення технології створення ПП, щоб собівартість була нижчою	Удосконалення моделі. Використання більш дешевих технологій для розробки, ніж використовують конкуренти, але тільки якщо ці технології відповідають необхідним вимогам якості.

6. За інтенсивністю - не марочна	Бренд присутній, але його роль незначна	Реклама, участь у конференціях, семінарах.

Було проведено аналіз конкуренції у галузі за моделлю М. Портера (табл. 4.9).

Таблиця 4.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товаризамінники
	Навести перелік прямих конкурентів	Визначити бар'єри входження в ринок	Визначити фактори сили постачальників	Визначити фактори сили споживачів	Фактори загроз з боку замінників
	Amazon Personalize	Наявність вже існуючих рішень	-	Контроль якості продукту	Наявність більш широкого функціоналу, зручнішого інтерфейсу та авторитет
Висновки:	Досить інтенсивна конкурентна боротьба з	Є можливості виходу на ринок, але є і конкуренти.	-	Клієнти диктують умови роботи на ринку:	Необхідно випускати ПЗ не гірше, ніж у конкурентів та розширяти функціонал.

	іншими гравцями			зручний інтерфейс	

За результатами аналізу було зроблено висновок про можливість роботи на ринку з огляду на конкурентну ситуацію.

Цей висновок був врахований при формулюванні переліку факторів конкурентоспроможності у наступному пункті. На основі аналізу конкуренції, проведеного в таблиці, а також із урахуванням характеристик ідеї проекту (табл. 4.2), вимог споживачів до товару (табл. 4.5) та факторів маркетингового середовища (табл. 4.6, 4.7) визначається та обґрунтовується перелік факторів конкурентоспроможності.

Аналіз оформлено у (табл. 4.10).

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування
1	Ціна	Один із факторів для вибору продукту клієнтом.
2	Якість	Один із факторів для вибору продукту клієнтом.
3	Зручність роботи з програмою	Дозволяє користувачу легко працювати з програмою

За визначеними факторами конкурентоспроможності (табл. 4.10) проведено аналіз сильних та слабких сторін стартап-проекту (табл. 4.11).

Таблиця 4.11 – Порівняльний аналіз сильних та слабких сторін проекту

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні						
			–3	–2	–1	0	+1	+2	+3
1	Ціна	15					*		
2	Якість	10			*				
3	Зручність роботи з програмою	15					*		

Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу (табл. 4.12) на основі виділених ринкових загроз та можливостей, та сильних і слабких сторін (таблиця 4.11).

Перелік ринкових загроз та ринкових можливостей було складено на основі аналізу факторів загроз та факторів можливостей маркетингового середовища. Ринкові загрози та ринкові можливості є наслідками впливу факторів, і, на відміну від них, ще не є реалізованими на ринку та мають певну ймовірність здійснення.

Таблиця 4.12 – SWOT-аналіз стартап-проекту

Сильні сторони: Якість Простота використання Висока швидкодія	Слабкі сторони: Дуже насичений ринок, мала кількість функціоналу, відсутня кросплатформеність.
Можливості: насичення ринку новим підходом до прогнозування; різноманітна клієнтура, вдосконалення системи	Загрози: Конкуренція

На основі SWOT-аналізу було розроблено альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок.

Визначені альтернативи були проаналізовані з точки зору строків та ймовірності отримання ресурсів (таблиця 4.13).

Таблиця 4.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	PR, просування бренду	50%	6 місяців
2	Перехід на безкоштовне розповсюдження	75%	3 місяців
3	Партнерство для об’єднання продукції	65%	2 місяці

Після аналізу було обрано альтернативу №2.

1.3 Аналіз ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: було проведено опис цільових груп потенційних споживачів (таблиця 4.14).

Таблиця 4.14 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Підприємці	Висока	Високий	Сильна	Просто
2	Великі компанії	Середня: велика конкуренція і можливість власних веб-відділів	Високий	Сильна	Складно
3	Маленькі компанії.	Низька	Низький	Слабка	Середня
Які цільові групи обрано:1,2,3					

За результатами аналізу потенційних груп споживачів було обрано цільові групу, для яких буде запропоновано даний товар, та визначено стратегію охоплення ринку - стратегію диференційованого маркетингу (компанія працює з декількома сегментами).

Для роботи в обраних сегментах ринку сформовано базову стратегію розвитку (таблиця 4.15).

Таблиця 4.15 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Постійне оновлення і покращення продукту	Ринкове позиціонування на індивіуальних користувачів	Швидкодія, якість продукту	Концентрований маркетинг

Наступним кроком обрано стратегію конкурентної поведінки (таблиця 4.16).

Таблиця 4.16 – Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
1	Ні.	Компанія буде шукати нових споживачів та забирати існуючих у конкурентів	Буде копіювати, удосконалювати та створювати свої унікальні пропозиції	Зайняття конкурентної ніші

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії) та до продукту (табл. 4.5), а також в залежності від обраної

базової стратегії розвитку (табл. 4.15) та стратегії конкурентної поведінки (таблиця 4.16) розроблено стратегію позиціонування (таблиця 4.17), що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 4.17 – Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартаппроекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту
1	Легкість розуміння, зручний інтерфейс, надійний, швидкий, точний та достовірний ПП для генерації рекомендацій.	Стратегія диференціації	Позиція на основі порівняння фірми з товарами конкурентів; Відмінні особливості споживача	Економія часу; Зручність застосування; Практичність та точність результату

Результатом виконання підрозділу стала узгоджена система рішень щодо ринкової поведінки стартап-компанії, яка визначає напрями роботи стартап- компанії на ринку.

1.4 Розроблення маркетингової програми стартап-проекту

Сформовано маркетингову концепцію товару, який отримає споживач. Для цього підсумовано результати попереднього аналізу конкурентоспроможності товару (таблиця 4.18). Концепція товару – письмовий опис фізичних та інших характеристик товару, які сприймаються споживачем, і набору вигод, які він обіцяє певній групі споживачів.

Таблиця 4.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Швидкість отримання результату	Швидка видача рекомендацій наступної пропозиці	Необхідно покращити швидкість навчання моделі для видачі рекомендацій наступної пропозиці
2	Зручність застосування	Нативна підтримка мови програмування Python	Розробка зручного прикладного програмного інтерфейсу
3	Практичність та точність результату	Користувач отримує точні (з малою похибкою розбіжності) результати рекомендацій.	Користувач на виході роботи ПП отримує модель та прогноз, котрі відповідають необхідним показникам варіативності та точності.

Розроблено трирівневу маркетингову модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання (таблиця 4.19).

1-й рівень. При формуванні задуму товару вирішується питання щодо того, засобом вирішення якої потреби і / або проблеми буде даний товар, яка його основна вигода. Дане питання безпосередньо пов’язаний з формуванням технічного завдання в процесі розробки конструкторської документації на виріб.

2-й рівень. Цей рівень являє рішення того, як буде реалізований товар в реальному/ включає в себе якість, властивості, дизайн, упаковку, ціну.

3-й рівень Товар з підкріпленням (супроводом) – додаткові послуги та переваги для споживача, що створюються на основі товару за задумом і товару в реальному виконанні (гарантії якості , доставка, умови оплати та ін).

Таблиця 4.19 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Генерація палітри кольорів зображення за допомогою інтелектуальних систем		
II. Товар реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх/Тл/Е/Ор
	1. Індивідуальний підхід. 2. Низька ціна. 3. Простота у використанні.	1.Нм 2.Нм 3.Нм	1.Технологічна 2.Економічна 3.Технологічна
	Якість: тестування фірмами аудиторами		
	Пакування: відсутнє		
	Марка: ROSO		

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар (остаточне визначення ціни відбувається під час фінансово-економічного аналізу проекту), яке передбачає аналіз ціни на товари-аналоги або товари

субститути, а також аналіз рівня доходів цільової групи споживачів (таблиця 4.20). Аналіз проведено експертним методом.

Таблиця 4.20 – Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	2000$	3700$	У всіх трьох груп високий рівень доходів	900$--

Наступним кроком є визначення оптимальної системи збуту, в межах якого було прийняте рішення (таблиця 4.21).

Таблиця 4.21 – Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Канал нульового рівня	Продаж	0(напряму)	Власна

Останньою складової маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (таблиця 4.22).

Таблиця 4.22 – Концепція маркетингових комунікацій

№ п/ п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
	Інтеграція API у клієнтській системі	Інтернет	Низька ціна, простота використання, універсальність	Показати переваги рішення над конкурентами, виділити ключові особливості	Створення сайту продукту, розповсюдження інформації про продукт на спеціалізованих ресурсах.

Було визначено, що придбання продукту буде проводитись через мережу Інтернет або при безпосередньому спілкуванні із представниками компанії. Розповсюдження інформації про продукт буде проводитись виключно через Інтернет, адже аудиторія даного продукту активно користується всесвітньою мережею.

Результатом підрозділу стала ринкова (маркетингова) програма, що включає в себе концепції товару, збуту, просування та попередній аналіз можливостей ціноутворення, спирається на цінності та потреби потенційних клієнтів, конкурентні переваги ідеї, стан та динаміку ринкового середовища, в межах якого впроваджено проект, та відповідну обрану альтернативу ринкової поведінки.

1.5 Висновки

В даному розділі проведено підготовчий аналіз для впровадження розробленої системи в якості стартап проекту. Досліджено аналогічні конкурентні системи, встановлено сильні та слабкі сторони системи в порівнянні з ними. Також було досліджено можливі шляхи розповсюдження продукту та його ймовірну аудиторію, рівень доходів та ймовірну ціну продукту, що розробляється.

Було проведено аналіз потенційних ризиків і можливостей, а також розраховані основні фінансово-економічні показники проекту. Отримані результати кажуть про те, що реалізація проекту є доцільною. Було визначено сильні сторони проекту: зручність у використанні, ціна, якість та широкий функціонал. Серед слабких варто виділити повільне навчання. Варто відмітити можливість реклами продукту на спеціалізованих ресурсах із зазначенням сильних сторін проекту.

ВИСНОВКИ

У даній дисертації було запропоновано запровадження нової системи для безпечного збереження великих даних, яка використовує блокчейн технології.

Було розроблено систему, яка використовує Ethereum для реєстру файлів і зберігання посилань на дані, що зберігаються в IPFS або Swarm.

Працездатність та ефективність даної моделі була перевірена експериментально шляхом тестування продуктивності. Результати показали, що розроблена система є життєспроможною для умов використання при помірній кількості запитів на секунду, здатна обробляти великі об’єми даних. Проте, аби досягти обробки великих даних близької до реального часу, використання публічних мереж Ethereum, Swarm чи IPFS не є доречним. Для подібних цілей слід розгортати приватну мережу блокчейну і Swarm, оскільки він показав кращі результати за IPFS.

Слід зазначити, що подібна техніка зберігання даних може застосовуватися для інших випадків використання, включаючи безпечне ведення записів в таких галузях, як установи державного правління, освіти, охорони здоров'я і т.д., а також розширення ємності сховища в міру необхідності без необхідності тимчасово припиняти надання послуг для технічного обслуговування.

Практичне значення отриманих результатів пов’язане з гарантією цілісності та безпеки даних, оскільки дані зберігаються у зашифрованому виді, модифікувати дані неможливо, а доступ до них має лише власник.

Найбільш наочним є приклад впровадження даної системи у будь-яку сферу, де дані є суто конфіденційними. Кращий приклад такої системи – це сфера медицини. Оскільки дані про медичний стан пацієнта є надзвичайно важливими, по-перше, необхідні гарантії цілісності даних, тобто що жодний

файл не буде втрачений, оскільки це може бути певні цінні результати аналізу, від якого залежить життя пацієнта. По-друге, також необхідно гарантувати безпечність збереження даних – що доступ до них має лише сам пацієнт і ніхто більше, оскільки дана тема є питанням приватного життя.

Також можна зазначити, що розроблена система є можливим аналогом відомих хмарних сховищ, таких як гугл диск, та може використовуватись звичайними користувачами для тих самих цілей, наприклад, для збереження архіву власних приватних фотографій. У створеній системі користувачі можуть бути впевненими, що ніхто ніколи не отримає до них доступ, як це було з айклаудом, коли група анонімних хакерів отримала доступ і опублікувала приватні плівки ряду відомих особистостей, спричинивши світовий скандал.

ПЕРЕЛІК ПОСИЛАНЬ

1. BlockChain [Електронний ресурс] – Режим доступу до ресурсу: https://www.investopedia.com/terms/b/blockchain.asp.
2. PoS [Електронний ресурс] – Режим доступу до ресурсу: http://www.marketch.ru/marketing_dictionary/marketing_terms_p/point_of_sale/.
3. NiceHash [Електронний ресурс] – Режим доступу до ресурсу: https://www.nicehash.com/.
4. Атака 51% [Електронний ресурс] – Режим доступу до ресурсу: https://academy.binance.com/ru/articles/what-is-a-51-percent-attack.
5. Proof-of-work [Електронний ресурс] – Режим доступу до ресурсу: https://forklog.com/chto-takoe-proof-of-work-i-proof-of-stake/.
6. Ethereum – глобальна платформа [Електронний ресурс] – Режим доступу до ресурсу: https://ethereum.org/.
7. Хмарне сховище даних [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Cloud_storage.
8. Storj [Електронний ресурс] – Режим доступу до ресурсу: https://storj.io/.
9. Mediacoin [Електронний ресурс] – Режим доступу до ресурсу: https://mediacoin.pro/.
10. IPFS [Електронний ресурс] – Режим доступу до ресурсу: https://ipfs.io/.
11. Axel [Електронний ресурс] – Режим доступу до ресурсу: https://www.axel.org/.
12. Peer-to-peer [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Peer-to-peer.
13. Elliptic Curve Digital Signature Algorithm [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm.

14. Хешування [Електронний ресурс] – Режим доступу до ресурсу: https://ssl.com.ua/blog/hashing-coding-encryption/.
15. Стійкість до колізії [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Collision_(computer_science).
16. Merkle Tree [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/455260/.
17. Фільтр Блума [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/otus/blog/541378/
18. BitTorrent [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/BitTorrent.
19. Transmission Control Protocol [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/docs/ru/aix/7.1?topic=management-transmission-control-protocolinternet-protocol
20. Схема Шнорра [Електронний ресурс] – Режим доступу до ресурсу: https://forklog.com/chto-takoe-podpisi-shnorra-chto-takoe-taproot/.
21. Tracker [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Tracker.
22. BitTorrent Peer Protocol [Електронний ресурс] – Режим доступу до ресурсу: https://wiki.theory.org/BitTorrentSpecification.
23. Algorithm choke [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/figure/Implementation-of-the-choking-algorithm_fig3_223808116
24. Моделі управління блокчейном [Електронний ресурс] – Режим доступу до ресурсу: https://cryptor.net/kriptovalyuty/modeli-upravleniya-v-blokcheyn-protokolah

ДОДАТОК А. ЛІСТИНГ ПРОГРАМНОГО КОДУ

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

contract Migrations {
  address public owner = msg.sender;
  uint public last_completed_migration;

  modifier restricted() {
    require(
      msg.sender == owner,
      "This function is restricted to the contract's owner"
    );
    _;
  }

  function setCompleted(uint completed) public restricted {
    last_completed_migration = completed;
  }
}
```

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

contract Irremissibile {
    string public name = "Irremissibile";
```

```
uint256 public fileCount = 0;
mapping(uint256 => File) public files;

struct File {
    uint256 fileId;
    string filePath;
    uint256 fileSize;
    string fileType;
    string fileName;
    address payable uploader;
}

event FileUploaded(
    uint256 fileId,
    string filePath,
    uint256 fileSize,
    string fileType,
    string fileName,
    address payable uploader
);

function uploadFile(
    string memory _filePath,
    uint256 _fileSize,
    string memory _fileType,
    string memory _fileName
) public {
    require(bytes(_filePath).length > 0);
    require(bytes(_fileType).length > 0);
```

```
        require(bytes(_fileName).length > 0);
        require(msg.sender != address(0));
        require(_fileSize > 0);

        fileCount++;

        files[fileCount] = File(
            fileCount,
            _filePath,
            _fileSize,
            _fileType,
            _fileName,
            payable(msg.sender)
        );

        emit FileUploaded(
            fileCount,
            _filePath,
            _fileSize,
            _fileType,
            _fileName,
            payable(msg.sender)
        );
    }
}
```

```
import { create, IPFSHTTPClient } from "ipfs-http-client";
```

```
import { IFile } from "../interfaces/ifile.interface";
import { PersssistFile } from "../interfaces/persssist-file.interface";

// here we handle the ipfs functionality.
export class AppStorage {

  ipfs: IPFSHTTPClient;
  untar: any;

  constructor() {
    const projectId = '2IyXllHLa4dN8DRWuq5lMgxeUr2';
    const projectSecret = '47eb8727407951d9f9b0c8ad5d0ad17a';


                const auth = 'Basic ' + Buffer.from(projectId + ':' +
projectSecret).toString('base64');

    this.ipfs = create({
      host: 'ipfs.infura.io',
      port: 5001,
      protocol: 'https',
      headers: {
        authorization: auth,
      }
    });
    this.initialize();
  }

  private async initialize() {
```

```
    if(typeof window === "undefined") return;
    this.untar = await require("js-untar");
}

async upload(file: IFile) {
    if(!file || !file.buffer) throw 'file not provided';
    if(!this.ipfs) throw 'ipfs not initialized';
    const blob = new Blob([file.buffer], { type: file.type });
    const result = await this.ipfs.add(blob);
    return result;
 }

async download(file: PersssistFile) {
    if(typeof window === "undefined") return;
    const iterable = this.ipfs.get(file.filePath);
    var chunks: Uint8Array[] = [];
    for await (const b of iterable) {
        chunks.push(b);
    }
    const tarball = new Blob(chunks, { type: 'application/x-tar' })
    const tarAsArrayBuffer = await tarball.arrayBuffer();
    const result = await this.untar(tarAsArrayBuffer);
    const resultFile = new Blob([result[0].buffer], { type: file.fileType })
    var url = window.URL.createObjectURL(resultFile);
    this.downloadURL(url, file.fileName);
}

private downloadURL(data: any, fileName: string) {
    var a;
```

```
        a = document.createElement('a');
        a.href = data;
        a.download = fileName;
        document.body.appendChild(a);
        a.click();
        a.remove();
    };

    listFiles() { }

}


// here we will handle the blockchain (Web3 library) functionality.

import Web3 from "web3";
import Persssist from '../../public/abis/Persssist.json'
import PersssistLocal from '../../abis/Persssist.json'
import { NetwokIds } from "../constants/networks";
import { PersssistFile } from "../interfaces/persssist-file.interface";


export class AppBlockchain {
    contract: any;
    web3: Web3 | undefined;
    initialized = false;

    supportedNetworks = [NetwokIds.goerli];
```

```
constructor() {}

private async ensureInitialized() {
    if(!this.initialized) {
        await this.initialize();
    }
}

async uploadFileMetadata(
    path: string,
    size: number,
    type: string,
    name: string,
    account: string,
    onSuccess: (hash: string) => void,
    onError: (e: any) => void,
) {
    await this.ensureInitialized();
    return this.contract.methods
        .uploadFile(path, size, type, name)
        .send({ from: account })
        .on('transactionHash', onSuccess)
        .on('error', onError);
}

async getFilesMetadata(): Promise<PersssistFile[]> {
    const methods = this.contract.methods;
    const filesCount = await methods.fileCount().call();
    const filesMetadata: PersssistFile[] = [];
```

```
    for (var i = filesCount; i >= 1; i--) {
      const file = await methods.files(i).call()
      filesMetadata.push({
        fileId: file.id,
        fileName: file.fileName,
        filePath: file.filePath,
        fileSize: file.fileSize,
        fileType: file.fileType,
        uploader: file.uploader
      });
    }
    return filesMetadata;
}

async requestAccounts() {
    if(typeof window === "undefined") return;
    return window
      .ethereum?.request({ method: "eth_requestAccounts" })
      .catch((err: any) => console.log(err));
}

async fetchAccounts() {
    if(typeof window === "undefined") return;
    return window
      .ethereum?.request({ method: "eth_accounts" })
      .catch((err: any) => console.log(err));
}

async detectAccountChanged(onChanged: (acc: string[]) => void) {
```

```
        window.ethereum.on('accountsChanged', onChanged);
    }

    async detectNetworkChanged(onError: (err: any) => void) {
        window.ethereum.on('networkChanged', (networkId: number) => {
            this.initializeContract(onError);
        });
    }

    private async initialize() {
        if(typeof window === "undefined") return;
        this.initializeWeb3();
        if (this.web3) {
            console.log('initializing contract')
            await this.initializeContract((err) => {
                throw err;
            });
        }
        this.initialized = true;
    }

    private initializeWeb3() {
        if(typeof window === "undefined") return;
        if (window.ethereum) this.web3 = new Web3(window.ethereum)
                        else if (window.web3) this.web3 = new
Web3(window.web3.currentProvider);
    }

    private async initializeContract(onError: (err: any) => void) {
```

```
    if (process.env.NEXT_PUBLIC_MODE === 'DEV') {
      return this.initializeContractLocal().catch(onError);
    }
    if (process.env.NEXT_PUBLIC_MODE === 'PROD') {
      return this.initializeContractRemote().catch(onError);
    }
  }

  private async initializeContractLocal() {
    if(!this.web3) throw 'Web3 not initialized';
    const networkId = await this.web3.eth.net.getId();
    const networkData = (PersssistLocal as any).networks[networkId];
    if (networkData) {
      this.contract = new this.web3.eth.Contract(
        (PersssistLocal as any).abi,
        networkData.address
      )
    }
  }

  private async initializeContractRemote() {
    if(!this.web3) throw 'Web3 not initialized';
    const networkId = await this.web3.eth.net.getId();
    if(!this.supportedNetworks.includes(networkId)) {
      throw {
        title: 'Network not supported',
        msg: 'Please make sure to connect to the Goerli Network'
      }
    } else {
```

```
            this.contract = new this.web3.eth.Contract(
                (Persssist as any).abi,
                process.env.NEXT_PUBLIC_CONTRACT
            );

        }
    }

    async contractSubscription(onData: () => any) {
        await this.ensureInitialized();
        if(!this.contract) return;
        this.contract.events.FileUploaded()
            .on('data', (event: any) => onData())
            .on('changed', (changed: any) => console.log(changed))
            .on('error', (err: any) => console.log(err))
            .on('connected', (str: any) => onData())
    }
}


import { createSlice } from '@reduxjs/toolkit';
import { IFile } from '../../../interfaces/ifile.interface';
import { PersssistFile } from '../../../interfaces/persssist-file.interface';
import { AppBlockchain } from '../../../lib/blockchain';
import { AppStorage } from '../../../lib/storage';
import { connectAccount } from '../accounts';

const appBlockchain = new AppBlockchain();
const appStorage = new AppStorage();
```

```
export const blockchain = createSlice({
name: 'blockchain',
initialState: {
        filesMetadata: [] as PersssistFile[],
        filesCount: 0,
},
reducers: {
        setFilesMetadata: (state, action) => {
                state.filesMetadata = action.payload;
                state.filesCount = action.payload.length;
        }
},
});

export const { setFilesMetadata } = blockchain.actions;
export default blockchain.reducer;

export const fetchFilesMetadata = () => async (dispatch: any) => {
const filesMetadata = await appBlockchain.getFilesMetadata();
dispatch(setFilesMetadata(filesMetadata));
}

export const subscribeToEvents = (errorCallback: (e: any) => void) =>
(dispatch: any) => {
const onAccountChanged = () => dispatch(connectAccount(errorCallback));
const onContractUpdated = () => dispatch(fetchFilesMetadata());

appBlockchain.detectAccountChanged(onAccountChanged)
```

```
appBlockchain.detectNetworkChanged(errorCallback)
appBlockchain.contractSubscription(onContractUpdated)
        .catch(errorCallback);
}

export const uploadFile = async (
file: IFile | null | undefined,
account: string,
successCallback: (hash: string) => void,
errorCallback: (e: any) => void
) => {
if(!file) return errorCallback('File not provided');
const addResult = await appStorage.upload(file);
return appBlockchain.uploadFileMetadata(
        addResult.path,
        addResult.size,
        file.type,
        file.name,
        account,
        successCallback,
        errorCallback
);
}


import { createSlice } from '@reduxjs/toolkit';
import { ConnectionState } from '../../../enums/connection-state';
import { AppBlockchain } from '../../../lib/blockchain';
```

```
const blockchain = new AppBlockchain();

export const accounts = createSlice({
   name: 'accounts',
   initialState: {
      list: [] as string[],
      connectionState: ConnectionState.DISCONNECTED,
   },
   reducers: {
      setAccounts: (state, action) => {
         state.list = [...action.payload];
      },
      setConnectionstate: (state, action) => {
         if(typeof window === "undefined") return;
         if(window.ethereum) {
            state.connectionState = action.payload;
         }else {
            state.connectionState = ConnectionState.UNAVAILABLE;
         }
      },
   },

});

export const { setAccounts, setConnectionstate } = accounts.actions;
export default accounts.reducer;

const connectionState = (accounts: string[]) => {
   if(typeof window === "undefined") return;
```

```
        if(window.ethereum) {
                return accounts.length > 0 ? ConnectionState.CONNECTED :
ConnectionState.DISCONNECTED;
        } else {
          return ConnectionState.UNAVAILABLE;
        }
      }

      export const checkExistingAccounts = () => (dispatch: any) => {
        blockchain.fetchAccounts().then((acc: string[]) => {
          if(acc && acc.length > 0) {
            dispatch(setAccounts(acc));
            dispatch(setConnectionstate(connectionState(acc)));
          }
        });
      }

      export const connectAccount = (errorCallback: (e: any) => void) =>
(dispatch: any) => {
        blockchain.fetchAccounts().then((acc: string[]) => {
          if(acc && acc.length > 0) {
            dispatch(setAccounts(acc));
            dispatch(setConnectionstate(connectionState(acc)));
          } else {
            blockchain.requestAccounts()
              .then((acc: string[]) => {
                if(!acc || acc.length === 0) {
                  return errorCallback({
                    title: 'No Account detected',
```

```
                    msg: 'Please make sure to connect a Metamask account'
                })
            }
            dispatch(setAccounts(acc));
            dispatch(setConnectionstate(connectionState(acc)));
        });
    }
  });
}




export const truncateAddress = (addr: string) => {
  return addr.slice(0, 4) + '...' + addr.slice(addr.length - 4, addr.length);
}

export const truncateName = (name: string) => {
  if (name.length > 16) {
    return name.slice(0, 14) + '...';
  }
  return name;
}

export const bytesToSize = (bytes: number) => {
  var sizes = ['Bytes', 'KB', 'MB', 'GB', 'TB'];
  if (bytes == 0) return '0 Byte';
  var i = Math.floor(Math.log(bytes) / Math.log(1024));
  return Math.round(bytes / Math.pow(1024, i)) + ' ' + sizes[i];
```

```
}


/**
 * Use this file to configure your truffle project. It's seeded with some
 * common settings for different networks and features like migrations,
 * compilation and testing. Uncomment the ones you need or modify
 * them to suit your project as necessary.
 *
 * More information about configuration can be found at:
 *
 * trufflesuite.com/docs/advanced/configuration
 *
  * To deploy via Infura you'll need a wallet provider (like @truffle/hdwallet-provider)
 * to sign your transactions before they're sent to a remote public node. Infura accounts
 * are available for free at: infura.io/register.
 *
 * You'll also need a mnemonic - the twelve word phrase the wallet uses to generate
 * public/private key pairs. If you're publishing your code to GitHub make sure you load this
 * phrase from a file you've .gitignored so it doesn't accidentally become public.
 *
 */

require('dotenv').config()
```

```
const HDWalletProvider = require('@truffle/hdwallet-provider');
const infuraProjectId = process.env.INFURA_PROJECT_ID
const accountAddr = process.env.ACCOUNT_ADDRESS;
module.exports = {

 contracts_build_directory: "./abis",
 networks: {
  development: {
   host: "localhost",
   port: 7545,
   network_id: "*" // Match any network id
  },
  kovan: {
   provider: () => {
    const privateKey = process.env.PRIVATE_KEY;
    return new HDWalletProvider(
     privateKey,
     `https://kovan.infura.io/v3/${infuraProjectId}`
    )
   },
   from: accountAddr,
   network_id: 42, // Kovan's id
   networkCheckTimeoutnetworkCheckTimeout: 10000,
   timeoutBlocks: 200,
  },
  rinkeby: {
   provider: () => {
    const privateKey = process.env.PRIVATE_KEY;
    return new HDWalletProvider(
```

```
        privateKey,
        `wss://rinkeby.infura.io/ws/v3/${infuraProjectId}`
      )
    },
    from: accountAddr,
    network_id: 4, // Rinkeby's id
    networkCheckTimeoutnetworkCheckTimeout: 10000,
    timeoutBlocks: 200,
  },
  goerli: {
    provider: () => {
      const privateKey = process.env.PRIVATE_KEY;
      return new HDWalletProvider(
          privateKey,
          `https://goerli.infura.io/v3/${infuraProjectId}`
      )
    },
    from: accountAddr,
    network_id: 5,  // Goerli's id
    networkCheckTimeoutnetworkCheckTimeout: 10000,
    timeoutBlocks: 200,
  }
},

// Set default mocha options here, use special reporters etc.
mocha: {
  // reporter: 'eth-gas-reporter',
  // reporterOptions: {
  //   excludeContracts: ['Migrations']
```

```
  // }
 },

 test_directory: './src/test',

 // Configure your compilers
 compilers: {
  solc: {
    version: "0.8.11",    // Fetch exact version from solc-bin (default: truffle's version)
     // docker: true,        // Use "0.5.1" you've installed locally with docker (default: false)
     // settings: {          // See the solidity docs for advice about optimization and evmVersion
     //  optimizer: {
     //    enabled: false,
     //    runs: 200
     //  },
     //  evmVersion: "byzantium"
     // }
  }
 },

 // Truffle DB is currently disabled by default; to enable it, change enabled:
 // false to enabled: true. The default storage location can also be
 // overridden by specifying the adapter settings, as shown in the commented code below.
 //
```

```
    // NOTE: It is not possible to migrate your contracts to truffle DB and you should
    // make a backup of your artifacts to a safe location before enabling this feature.
    //
    // After you backed up your artifacts you can utilize db by running migrate as follows:
    // $ truffle migrate --reset --compile-all
    //
    // db: {
      // enabled: false,
      // host: "127.0.0.1",
      // adapter: {
      //   name: "sqlite",
      //   settings: {
      //     directory: ".db"
      //   }
      // }
    // }
    plugins: ["solidity-coverage"]
};
```

```
import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
```

```
# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/2.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '<SECRET-KEY>'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
ALLOWED_HOSTS = []

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'mysite.dapp',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
```

```
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'mysite.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'mysite.wsgi.application'


# Database
# https://docs.djangoproject.com/en/2.0/ref/settings/#databases

DATABASES = {
    'default': {
```

```
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}


# Password validation
#
https://docs.djangoproject.com/en/2.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
                                                    'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
                                                    'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
                                                    'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
                                                    'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

```
# Internationalization
# https://docs.djangoproject.com/en/2.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'America/New_York'
USE_I18N = True
USE_L10N = True
USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.0/howto/static-files/

STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')



from django.http import JsonResponse
from django.shortcuts import render
from web3 import Web3, HTTPProvider
from .contract_abi import abi
from .form import getMsg
import ipfsapi

def index(request):
    return render(request, 'index.html',{})
```

```
def ipfsLink(request):
    if request.POST:
        if 'msg' in request.POST:
            msg = request.POST['msg']
            api = ipfsapi.Client("https://ipfs.infura.io", 5001)
            ipfs_hash = api.add_str(msg)
            d = {'hash':ipfs_hash,'link':"https://ipfs.io/ipfs/{}".format(ipfs_hash)}
            return JsonResponse(d)
        else:
            file = request.FILES['file']
            api = ipfsapi.Client("https://ipfs.infura.io", 5001)
            ipfs_dic = api.add(file)
                                                                    d =
{'hash':ipfs_dic['Hash'],'link':"https://ipfs.io/ipfs/{}".format(ipfs_dic['Hash']),
'Name': ipfs_dic['Name']}
            return JsonResponse(d)
    else:
        form = getMsg()
    return render(request, 'ipfsLink.html',{})

def ipfsLinkRecent(request):
    if request.POST:
                                                                w3 =
Web3(HTTPProvider('https://goerli.infura.io/v3/ad6c33a7fa6e4cf1b21f344139b3c
d7c'))
                                                            contract_address =
w3.toChecksumAddress('0x5101712E59E91A91e19b3F6B909b9b5553a3C96A')
        contract = w3.eth.contract(address=contract_address, abi=abi)
```

```
            userAcc = request.POST['userAccount']

            count = 2
            arr = []
                                    (hashSender, hashString, hashTimestamp) =
contract.functions.findHash(1).call()
            while (hashString != ""):
                if hashSender.lower() == userAcc.lower():
                    arr.append([hashString, hashTimestamp])
                                        (hashSender, hashString, hashTimestamp) =
contract.functions.findHash(count).call()
                    count += 1
            return JsonResponse({"arr":arr})
        else:
            return render(request, 'ipfsLink.html', {})
```

```
#!/usr/bin/python3

## To import from interfacer_modules which is one directory above
import os
import sys
sys.path.append("..")

from web3 import Web3

## Custom modules
```

```
        from interfacer_modules.blockchain.smartcontract import
SmartContractCaller
        import project_settings

        ## blockchain url and addresses
        smart_contract_address = project_settings.smart_contract_address
        eth_blockchain_url = project_settings.eth_blockchain_url
        abi_filename = os.path.abspath("../abi/contract_abi.json")

        ## Smart Contract Setup
        smart_contract_instance = SmartContractCaller(smart_contract_address,
eth_blockchain_url)

        ## Load the ABI file
        smart_contract_instance.load_abi(abi_filename)

        ## Create a smart contract obj to use
        smart_contract_instance.create_smartcontract_obj()

        ## Enter the address to register in contract
        print("Enter address to register in Smart Contract:")
        address_to_deregister = Web3.toChecksumAddress(input())

        res = smart_contract_instance.deregister_device(address_to_deregister)
        print()
        if res["_message"] == "SUCESSFULLY DEREGISTERED":
            print("INFO: Successfully removed the device")
        elif res["_message"] == "DEVICE NOT REGISTERED":
            print("ERROR: Device was already deregistered")
```

```
        else:
            print("ERROR: Failed to register device")

        #!/usr/bin/python3

        ## To import from interfacer_modules which is one directory above
        import os
        import sys
        sys.path.append("..")

        from web3 import Web3

        ## Custom modules
        from interfacer_modules.blockchain.smartcontract import
SmartContractCaller
        import project_settings

        ## blockchain url and addresses
        smart_contract_address = project_settings.smart_contract_address
        eth_blockchain_url = project_settings.eth_blockchain_url
        abi_filename = os.path.abspath("../abi/contract_abi.json")

        ## Smart Contract Setup
        smart_contract_instance = SmartContractCaller(smart_contract_address,
eth_blockchain_url)

        ## Load the ABI file
        smart_contract_instance.load_abi(abi_filename)
```

```
## Create a smart contract obj to use
smart_contract_instance.create_smartcontract_obj()

## Enter the address to register in contract
print("Enter address to register in Smart Contract:")
address_to_register = Web3.toChecksumAddress(input())

res = smart_contract_instance.register_device(address_to_register)
#print(res)
print()
if res["_message"] == "SUCESSFULLY REGISTERED":
   print("INFO: Successfully added new device")
elif res["_message"] == "DEVICE ALREADY REGISTERED":
   print("ERROR: Device was already registered")
else:
   print("ERROR: Failed to register device")



#!/usr/bin/python3

import os
import datetime
import time
import sys
sys.path.append("..")
import project_settings
import requests
import json
```

```
        ## Custom modules
        from interfacer_modules.blockchain.smartcontract import
SmartContractCaller

        ## Create payload to store in swarm
        def create_payload(temperature, humidity):
            """
                Data to store as payload in swarm
                1. Temperature
                2. Humidity
                3. Temperature Units (Celsius)
                4. Humidity Units (%)
                5. Timestamp
                6. Device type
                7. Device ID
                8. Device IP
                9. Sensor Type
            """
            ## Get current time
            current_time = str(datetime.datetime.now())

            payload = {
                "Temperature": temperature,
                "Humidity": humidity,
                "TemperatureUnits": "Celsius",
                "HumidityUnits": "%",
                "Timestamp": current_time,
                "DeviceType": "Raspberry Pi 3B+",
```

```
            "DeviceID": "IoTProducer1",
            "DeviceIP": "192.168.0.16",
            "SensorType": "DHT11"
        }
        return json.dumps(payload)

    ## IOT sensor values
    dht_version = project_settings.dht_version
    time_recheck_reading = project_settings.time_recheck_reading
    dht_GPIO = project_settings.dht_GPIO

    ## blockchain url and addresses
    smart_contract_address = project_settings.smart_contract_address
    eth_blockchain_url = project_settings.eth_blockchain_url
    abi_filename = os.path.abspath("../abi/contract_abi.json")

    ## Smart Contract Setup
    smart_contract_instance = SmartContractCaller(smart_contract_address,
eth_blockchain_url)

    ## Load the ABI file
    smart_contract_instance.load_abi(abi_filename)

    ## Create a smart contract obj to use
    smart_contract_instance.create_smartcontract_obj()

    ## Get readings
    current_temperature,current_humidity = 23,44
```

```
        ## Create a payload tot store in Swarm
        swarm_store = create_payload(current_temperature, current_humidity)

        ## Send POST request to swarm to store the payload
        r = requests.post(project_settings.swarm_blockchain_url, data=swarm_store,
headers={'Content-Type': 'text/plain'})

        # Store the received filehash for swarm
        filehash = r.text

        ## Save the filehash in blockchain
        res = smart_contract_instance.set_filehash_blockchain(filehash)

        if res["_message"] == "DEVICE NOT REGISTERED":
            print("ERROR: Device not registered")

        else:
            ## Printing Result to console
            print(res["_message"])
             print("Temp: ",current_temperature," and Humidity: ",current_humidity,"
set at: ", str(datetime.datetime.now()))



        #!/usr/bin/python3

        import os
        import datetime
        import time
        import project_settings
```

```
import requests
import json
import sys
sys.path.append("..")

## Custom modules
from interfacer_modules.blockchain.smartcontract import SmartContractCaller

## blockchain url and addresses
smart_contract_address = project_settings.smart_contract_address
eth_blockchain_url = project_settings.eth_blockchain_url
abi_filename = os.path.abspath("../abi/contract_abi.json")

time_recheck_reading = project_settings.time_recheck_reading

## Smart Contract Setup
smart_contract_instance = SmartContractCaller(smart_contract_address, eth_blockchain_url)

## Load the ABI file
smart_contract_instance.load_abi(abi_filename)

## Create a smart contract obj to use
smart_contract_instance.create_smartcontract_obj()

## Helper functions to convert f to c and c to f
def fahrenheit_to_celsius(temperature):
    return round((temperature * 9/5) + 32)
```

```
        def celsius_to_fahrenheit(temperature):
            return round(temperature * 1.8 + 32)

        print("Latest Readings: ")
        ## Run indefinitely
        while True:
            try:
                ## Get filehash from swarm
                filehash_swarm = smart_contract_instance.get_filehash_latest()
                if filehash_swarm == "ERROR":
                    print("ERROR: Device not registered")
                    break

                ## Get payload stored in filehash from swarm
                                                                    res =
requests.get(project_settings.swarm_blockchain_url+filehash_swarm+"/")
                payload_res = json.loads(res.text)

                ## Get the temperature, hulidity and timestamp stored
                temp_in_f = celsius_to_fahrenheit(payload_res["Temperature"])
                humidity = payload_res["Humidity"]
                time_captured = payload_res["Timestamp"]

                ## Print to console
                 print("Temperature: ", str(temp_in_f), " - Humidity: "+ str(humidity)+ "
- Timestamp: "+time_captured)
                #print(payload_final)
```

```
        ## Sleep and restart
        time.sleep(time_recheck_reading)

    except KeyboardInterrupt:
        print("Closed by user")
        break



#!/usr/bin/python3

import time
from web3 import Web3, HTTPProvider
import json
import datetime

class SmartContractCaller:
    def __init__(self, smart_contract_address, eth_blockchain_url):
        self.smart_contract_address = smart_contract_address
        self.eth_blockchain_url = eth_blockchain_url



    def load_abi(self, abi_filename):
        ## load the abi
        with open(abi_filename, "r") as abi_file:
            try:
                self.abi_ = json.load(abi_file)
            except:
                print('Failed to open the ABI')
```

```
        def create_smartcontract_obj(self):
            ## make a connection to the blockchain node
            self.w3 = Web3(HTTPProvider(self.eth_blockchain_url))

            ## Convert address to checksummed address
                                        self.chk_smart_contract_address =
Web3.toChecksumAddress(self.smart_contract_address)

            ## get address of the node
            self.executor_address = self.w3.eth.accounts[0]

            ## connect to the smart contract
                                                self.sensor_contract =
self.w3.eth.contract(address=self.chk_smart_contract_address, abi=self.abi_)


        ## DEPRECATED function to send temp and humidity to blockchain
                def set_tempandhumidity_blockchain(self, temp, humidity,
dataStorageTime):
            set_fn_txn_hash = self.sensor_contract.functions.setSensorData(temp,
humidity, dataStorageTime).transact({"from":self.executor_address})
            res = self.w3.eth.waitForTransactionReceipt(set_fn_txn_hash)
            return res

        ## DEPRECATED function to get the current temp and humidity
        def get_tempandhumidity_latest(self):
                                                temp_and_humidity =
self.sensor_contract.functions.getSensorDataLatest().call({"from":self.executor_ad
dress})
```

```
            return temp_and_humidity

        ## DEPRECATED function to get the current temp and humidity from ID
        def get_tempandhumidity_fromID(self, ID):
                                                    temp_and_humidity =
self.sensor_contract.functions.getSensorDataByID(ID).call({"from":self.executor_
address})

            return temp_and_humidity

        ## Test function to send temp and humidity and the full payload to
blockchain
        def set_blockchain(self, temp, humi, tempunits, humiunits, timestamp,
devicetype, deviceid, deviceip, sensortype):
            set_fn_txn_hash = self.sensor_contract.functions.setSensorData(temp,
humi, tempunits, humiunits, timestamp, devicetype, deviceid, deviceip,
sensortype).transact({"from":self.executor_address})
            #res = self.w3.eth.waitForTransactionReceipt(set_fn_txn_hash)
            #return res

        ## function to save swarm filehash to blockchain
        def set_filehash_blockchain(self, filehash):
                                                    set_fn_txn_hash =
self.sensor_contract.functions.setSensorData(filehash).transact({"from":self.execut
or_address})
            tx_receipt = self.w3.eth.waitForTransactionReceipt(set_fn_txn_hash)
                                                    result =
self.sensor_contract.events.setFileHashEvent().processReceipt(tx_receipt)
```

```
        return result[0]['args']

    ## function to save swarm filehash to blockchain. This sends only txn, doesnt wait for receipt
    def set_filehash_blockchain_test(self, filehash):
                                                    set_fn_txn_hash = self.sensor_contract.functions.setSensorData(filehash).transact({"from":self.executor_address})

    ## function to get latest swarm filehash
    def get_filehash_latest(self):
                                                    filehash = self.sensor_contract.functions.getSensorDataLatest().call({"from":self.executor_address})
        return filehash

    ## function to get swarm filehash using ID
    def get_filehash_id(self, id_req):
                                                    filehash = self.sensor_contract.functions.getSensorDataByID(id_req).call({"from":self.executor_address})
        return filehash

    ## function to get current ID from blockchain
    def get_current_BCID(self):
                                                    req_id = self.sensor_contract.functions.getCurrentID().call({"from":self.executor_address})
        return req_id
```

```
        ## Owner only Functions
        ## Register the IoT device
        def register_device(self, address_to_register):
                                                                res_hash =
self.sensor_contract.functions.registerDevice(address_to_register).transact({"from"
:self.executor_address})
            tx_receipt = self.w3.eth.waitForTransactionReceipt(res_hash)
                                                                result =
self.sensor_contract.events.deviceEvent().processReceipt(tx_receipt)
            return result[0]['args']

        ## De-Register the IoT device
        def deregister_device(self, address_to_deregister):
                                                                res_hash =
self.sensor_contract.functions.deregisterDevice(address_to_deregister).transact({"f
rom":self.executor_address})
            tx_receipt = self.w3.eth.waitForTransactionReceipt(res_hash)
                                                                result =
self.sensor_contract.events.deviceEvent().processReceipt(tx_receipt)
            return result[0]['args']
```