Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Модель децентралізованого зберігання даних
для децентралізованого застосунку і смарт-контракту
у мережі Ethereum
(тема)

Виконав:

студент II курсу, групи СПм-23-2

Селіванов І.О.
(прізвище, ініціали)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Системне програмування
(повна назва освітньої програми)

Керівник: проф. Фесенко Т.Г.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Селіванову Івану Олексійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Модель децентралізованого зберігання даних для децентралізованого застосунку і смарт-контракту у мережі Ethereum

затверджена наказом по університету від “ 22 ” листопада 2024 р. № 1236 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 20 січня 2025 р.

3. Вхідні дані до роботи

1. загальні поняття, що пов’язані з децентралізованими додатками: блокчейн, мережа Ethereum, блоки, транзакції, смарт-контракти, механізми консенсусу, стандарти ERC;
2. мови програмування: TypeScript, Solidity;
3. фреймворк: React;
4. бібліотеки: Ethers, Web3, NFT.Storage, OpenZeppelin.

4. Перелік питань, що потрібно опрацювати у роботі

1. Дослідження наявних проблем в інформаційній безпеці публічних даних.
2. Аналіз технології блокчейн та її переваг.
3. Розробка моделі децентралізованого зберігання даних, алгоритм, методику реалізації для децентралізованого застосунку і смарт-контракту у мережі Ethereum
4. Розробка застосунку децентралізованого зберігання даних у мережі Ethereum.
5. Розробка смарт-контракту для зберігання даних у децентралізованій мережі.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) слайд-презентація – 15слайдів: титульна сторінка, мета роботи та завдання, існуючі варіанти забезпечення інформаційної безпеки, аналіз існуючих варіантів забезпечення інформаційної безпеки, основні властивості даних в блокчейн-системах, модель та алгоритм роботи децентралізованого додатку, вибір технологій для реалізації та функціональні можливості застосунку, висновки, апробація результатів кваліфікаційної роботи.

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	26.11.2024-30.11.2024	
2	Аналіз завдання та пошук літератури	01.12.2024-04.12.2024	
3	Аналіз технічних засобів для реалізації	05.12.2024-07.12.2024	
4	Розробка програмних модулів	08.12.2024-19.12.2024	
5	Відлагодження програмних модулів	19.12.2024-28.12.2024	
6	Опрацювання результатів дослідження	29.12.2024-04.01.2025	
7	Оформлення матеріалів кваліфікаційної роботи	05.01.2024-17.01.2025	

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Фесенко Т.Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 105 с., 22 рис., 2 дод., 40 джерел.

ДЕЦЕНТРАЛІЗАЦІЯ, ІНФОРМАЦІЙНА БЕЗПЕКА, РОЗПОДІЛЕНА СИСТЕМА, ДЕЦЕНТРАЛІЗОВАНА СИСТЕМА, БЛОКЧЕЙН, МЕРЕЖА ETHEREUM, БЛОК, ТРАНЗАКЦІЯ, СМАРТ-КОНТРАКТ, СТАНДАРТ ERC-721, NFT, TYPESCRIPT, REACT.

Метою кваліфікаційної роботи є дослідження нових методів забезпечення інформаційної безпеки публічних даних та розробка моделі децентралізованого застосунку на базі смарт-контракту в блокчейні Ethereum, що реалізовує децентралізоване зберігання даних.

У ході виконання кваліфікаційної роботи було проведено аналіз існуючих централізованих рішень інформаційної безпеки та їх актуальність. Розглянуто методи забезпечення безпеки на основі розподілених систем керування та блокчейну. Розроблено модель децентралізованого зберігання даних з інтеграцією можливостей блокчейн-технологій для безпечного зберігання публічних даних. Запропоновано алгоритм і методику реалізації децентралізованого застосунку на базі смарт-контракту із використанням таких додаткових технологій, як MetaMask для інтеграції гаманців, NFT.Storage для зберігання метаданих NFT, а також Etherscan для перевірки транзакцій у блокчейн-мережі Ethereum.

ABSTRACT

Master’s thesis: 105 pages, 22 figures, 2 appendices, 40 sources.

DECENTRALIZATION, INFORMATION SECURITY, DISTRIBUTED SYSTEM, DECENTRALIZED SYSTEM, BLOCKCHAIN, ETHEREUM NETWORK, BLOCK, TRANSACTION, SMART CONTRACT, ERC-721 STANDARD, NFT, TYPESCRIPT, REACT.

The objective of this qualification work is to explore new methods for ensuring the information security of public data and to develop a decentralized application based on a smart contract within the Ethereum blockchain that implements decentralized data storage.

In the course of the qualification work, an analysis of existing centralized information security solutions and their relevance was conducted. Additionally, methods for ensuring security based on distributed management systems and blockchain technology were examined. A model of decentralized data storage was developed, leveraging blockchain technology to ensure the secure storage of public data. An algorithm and methodology for implementing the decentralized application were proposed, utilizing smart contracts alongside additional technologies such as MetaMask for wallet integration, NFT.Storage for storing NFT metadata, and Etherscan for verifying transactions in the Ethereum blockchain network.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ABI (англ. Application Binary Interface) – інтерфейс прикладного бінарного програмування

API (Application Programming Interface) – інтерфейс прикладного програмування

dApp (decentralized application) – децентралізований застосунок

DDoS (Distributed Denial of Service) – розподілена атака на відмову в обслуговуванні

DeFi (decentralized finance) – децентралізовані фінанси

ERC (Ethereum Request for Comments) – набір стандартів для смарт-контрактів у блокчейні Ethereum

ETH (Ethereum) – нативна криптовалюта блокчейна Ethereum

EVM (Ethereum Virtual Machine) – віртуальна машина Ethereum

ID (Identifier) – ідентифікатор, унікальний номер або позначення

IDS (Intrusion Detection System) – система виявлення вторгнень

IPFS (InterPlanetary File System) – міжпланетна файлова система для розподіленого зберігання та доступу до даних

IPS (Intrusion Prevention System) – система запобігання вторгненням

JSON (JavaScript Object Notation) – формат обміну даними

NFT (Non-Fungible Token) – унікальний цифровий актив

PoS (Proof of Stake) – механізм консенсусу на основі доказу володіння долею

PoW (Proof of Work) – механізм консенсусу на основі доказу виконання роботи

SQL (Structured Query Language) – мова структурованих запитів для роботи з базами даних

URI (Uniform Resource Identifier) – уніфікований ідентифікатор ресурсу

ВСТУП

У сучасному світі, де кількість цифрових даних зростає експоненційно, питання їх зберігання та безпеки стає критичним. Традиційні централізовані системи зберігання даних мають значні недоліки: ризик крадіжки, залежність від довірених третіх сторін, а також можливість втрати інформації через технічні несправності чи зловмисні атаки [1]. Це робить необхідним пошук нових підходів до зберігання та захисту даних, особливо у сферах, де автентичність і цілісність інформації є визначальними.

Однією з таких сфер є державні реєстри, де інформація про права власності часто стає об’єктом маніпуляцій. У разі незаконних змін у реєстрах виникають конфлікти щодо власності, що може завдати значної шкоди як окремим особам, так і суспільству в цілому. Іншим прикладом є дані про державні закупівлі, які мають бути відкритими для громадськості, щоб забезпечити прозорість витрачання коштів і мінімізувати ризик корупції.

Проблема також стосується сфери цифрової інтелектуальної власності, такої як книги, музика, малюнки чи відеоконтент. У разі відсутності надійного механізму підтвердження авторства та захисту прав такі активи можуть бути викрадені або використані без згоди правовласника.

Технологія блокчейн відкриває нові можливості для вирішення цих питань. Децентралізовані системи забезпечують розподілене зберігання даних, що виключає залежність від єдиного вузла, забезпечує стійкість до зовнішніх атак і підвищує прозорість процесів. Смарт-контракти, що функціонують на блокчейні, автоматизують виконання угод, забезпечуючи додаткову надійність і безпеку.

Об’єктом дослідження є децентралізована система зберігання даних, заснована на блокчейн-технології.

Предметом дослідження є особливості функціонування смарт-контрактів та їх використання для створення моделі децентралізованого

зберігання даних у мережі Ethereum із застосуванням стандарту ERC-721.

Метою роботи є створення моделі децентралізованого зберігання даних для децентралізованого застосунку, яка забезпечує прозорість, безпеку та підтвердження автентичності цифрових активів із використанням блокчейну Ethereum і смарт-контрактів.

Для досягнення поставленої мети було визначено наступні завдання:

1. дослідити концепцію децентралізації та її значення у сучасних інформаційних системах;

2. проаналізувати основні загрози інформаційній безпеці та обґрунтувати необхідність використання блокчейн-технологій для їх зниження;

3. визначити функціональні можливості блокчейну Ethereum та його переваги для зберігання даних;

4. розробити модель децентралізованого зберігання даних, що відповідає вимогам прозорої та безпечної роботи з цифровими активами;

5. розробити децентралізований застосунок із інтеграцією смарт-контракту та забезпеченням роботи з NFT;

6. провести тестування розробленої системи, перевіривши її працездатність та відповідність вимогам децентралізації.

Таким чином, у роботі пропонується розглянути модель децентралізованого зберігання даних у мережі Ethereum, яка спрямована на розв'язання актуальних проблем безпеки даних та захисту інтелектуальної власності. Це дослідження має потенціал для створення надійної інфраструктури, здатної забезпечити прозорість та доступність інформації в сучасному цифровому середовищі.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Актуальність завдання

У контексті стрімкого розвитку інформаційних технологій, питання захисту даних стає все більш нагальним. З кожним роком все більше аспектів життя переходить у цифровий формат: комунікації, банківські послуги, електронна комерція, а також управління державними процесами – усе це вже міцно пов'язане з мережею Інтернет. Однак, поряд із цим зростає загроза витоків даних, маніпуляцій та шахрайства, що ставить під сумнів надійність традиційних централізованих систем зберігання даних.

Останніми роками технології блокчейн і смарт-контракти стали важливими інструментами для забезпечення безпеки даних завдяки своїй децентралізованій природі. Вони дозволяють уникнути концентрації контролю в одному вузлі, розподіляючи управління та зберігання даних між різними учасниками мережі [2]. Це забезпечує більший рівень захисту від атак, оскільки жоден вузол не може бути єдиною точкою збою. Наприклад, проєкт Filecoin продемонстрував ефективність децентралізованого зберігання файлів, дозволяючи користувачам зберігати дані на безлічі незалежних вузлів, що робить їх менш вразливими до атак та витоків [3].

Враховуючи швидке впровадження блокчейн-технологій у багатьох сферах, а також зростаючі загрози для безпеки даних у традиційних централізованих системах, тема децентралізованого зберігання даних є надзвичайно актуальною. Впровадження децентралізованих рішень у різних галузях, зокрема в управлінні публічними даними, сприяє підвищенню безпеки, прозорості та довіри до систем управління даними в теперішньому часі та в майбутньому.

1.2 Основні види загроз в інформаційній безпеці

Загрози інформаційній безпеці – це зловмисні дії, спрямовані на крадіжку або знищення конфіденційних даних, компрометацію комп'ютерних систем чи ідентифікаційних даних, порушення або саботаж бізнес-операцій та загалом порушення цифрового життя. До поширених типів загроз інформаційній безпеці належать шкідливе програмне забезпечення, програми-вимагачі, атаки типу «відмова в обслуговуванні» (DoS) та атаки із застосуванням SQL-ін'єкцій.

Інформаційна безпека є комплексом заходів, що спрямовані на захист даних від випадкових дій або навмисних загроз [4]. Метою таких дій може бути спричинення шкоди власникам інформації. Ця шкода може мати як матеріальний, так і моральний характер.

Основними принципами інформаційної безпеки є цілісність даних, їх доступність, конфіденційність та достовірність. Цілісність передбачає, що дані залишаються незмінними протягом їх зберігання або передачі. Доступність означає, що легальні користувачі повинні мати безперешкодний доступ до інформації. Конфіденційність забезпечує обмеження доступу до даних стороннім особам. Достовірність гарантує, що інформація надходить від надійного джерела.

Основні типи загроз включають:

- витік даних, який виникає, коли зловмисники отримують несанкціонований доступ до конфіденційної інформації, такої як особисті дані або комерційні секрети. Це може призвести до матеріальних та моральних збитків для організацій, а також до порушень стандартів і регуляцій, що тягне за собою правові наслідки;

- атака нульового дня. Цей тип атаки використовує невідомі вразливості у програмному чи апаратному забезпеченні. Оскільки про проблему не знають ані виробники, ані користувачі, зловмисники можуть вільно діяти, поки їх не виявлять;

- шкідливе програмне забезпечення – це програми, які виконують зловмисні дії на пристроях, таких як комп’ютери чи мережі. Воно може вражати систему, виконуючи специфічні завдання, наприклад, шпигунське програмне забезпечення або програми-вимагачі, які вимагають викуп за відновлення даних;

- програми-вимагачі, які блокують доступ до інформації через шифрування, а потім вимагають викуп за її розшифровку. Зазвичай зловмисники погрожують видалити чи розголосити дані, якщо викуп не буде сплачено. Такий тип атак може мати руйнівні наслідки, зокрема для бізнесу;

- атаки відмови в обслуговуванні (DoS). Ці атаки перевантажують ресурси системи, щоб вона не могла обробляти запити законних користувачів. Більш масштабна форма такої атаки – розподілена DoS (DDoS), що використовує кілька заражених пристроїв для повного виведення системи з ладу;

- SQL-ін’єкція – це атака, що дає зловмисникам можливість отримати несанкціонований доступ до бази даних, додаючи шкідливий код до SQL-запитів. Такі атаки дозволяють отримати доступ до конфіденційних даних, паролів або іншої важливої інформації;

- складні загрози. Атаки складного рівня спрямовані на проникнення в систему з метою отримання чутливих даних. Зазвичай їх підтримують кримінальні групи або держави для довготривалого доступу до важливих ресурсів та інформації.

Вразливості можуть бути як випадковими, так і навмисними, та викликані зовнішніми або внутрішніми факторами.

Одним із найяскравіших прикладів масштабної атаки стало порушення безпеки компанії Equifax у 2017 році. В результаті цієї атаки було викрадено персональні дані понад 147 мільйонів осіб, зокрема їхні номери соціального страхування, адреси та фінансова інформація. Хоча система була захищена традиційними централізованими методами, злом став можливим через недоліки в оновленнях системи безпеки. Цей випадок показує вразливість

централізованих систем, які можуть стати одною точкою провалу у разі успішної атаки на сервери.

Крім цього, несанкціонований доступ до публічних реєстрів або маніпуляції в них можуть завдати серйозної шкоди, зокрема у сфері реєстрації майна або державних закупівель. У 2019 році було зафіксовано випадок фальсифікації даних у кадастрах землі в Україні, коли зміни в реєстрі дозволили привласнити ділянки землі. Це стало можливим через слабку безпеку централізованої системи.

Останнім прикладом масштабної кібератаки є злом реєстрів Міністерства юстиції України, що фактично паралізувало значну частину господарської діяльності в країні на декілька тижнів. У результаті атаки на центральні сервери тимчасово припинили роботу ключові державні реєстри. Також після потрапляння хакерів в інфраструктуру міністерства є ризик викрадення та видалення понад 1 мільярд рядків даних.

Ця атака також вплинула на роботу системи проведення державних закупівель, яка залежить від даних з реєстру Мін’юсту. Ця кібератака свідчить про критичну залежність сучасних економік від доступу до централізованих реєстрів, а також про високі ризики, пов'язані з їх безпекою.

Таким чином, сучасні інформаційні загрози можуть приймати різноманітні форми: від несанкціонованого доступу до публічних або конфіденційних даних до шахрайських схем із використанням соціальної інженерії. Основним елементом таких атак є вразливість до централізованого контролю над даними, що підкреслює важливість розробки та впровадження децентралізованих рішень, які зможуть підвищити рівень захисту та мінімізувати ризики компрометації інформації.

1.3 Наявні системи та методи забезпечення інформаційної безпеки

Методи забезпечення інформаційного захисту залежать від специфіки системи та умов її функціонування. Вони можуть включати правила

контролю доступу до інформаційних ресурсів, створення процедур для відновлення системи після збоїв, застосування методів приховування даних, а також регламенти, що визначають правила роботи з конфіденційною інформацією.

Методи забезпечення інформаційної безпеки можна поділити на чотири основні категорії: апаратні, адміністративні, правові та фізичні. Кожен з них виконує свою специфічну функцію у побудові багаторівневої стратегії безпеки, що спрямована на захист даних та інформаційних ресурсів від різноманітних загроз.

До основних інструментів належать фаєрволи, шифрування, антивірусне програмне забезпечення, мультифакторна аутентифікація, системи виявлення та запобігання вторгнень, постійний моніторинг і аудит безпеки та правові акти. Кожен із цих інструментів виконує особливу роль у забезпеченні захисту, а їхнє поєднання дозволяє створити комплексну систему кібербезпеки.

Програмні та апаратні фаєрволи здійснюють фільтрацію мережевого трафіку, блокуючи або дозволяючи підключення залежно від налаштованих правил. Вони створюють бар'єр між внутрішніми системами та зовнішнім трафіком, знижуючи ризики вторгнень [5].

Шифрування даних є одним із найбільш ефективних засобів забезпечення безпеки. За допомогою криптографічних алгоритмів, шифрування перетворює конфіденційні дані в незрозумілий текст, який може бути відновлений лише з використанням спеціального ключа [6]. Воно широко використовується для захисту персональних даних у хмарних сховищах і під час передачі даних через незахищені мережі.

Антивірусне програмне забезпечення допомагає виявляти, блокувати і видаляти шкідливі програми, такі як віруси, черв'яки і трояни. Використовується для виявлення, нейтралізації та видалення шкідливого програмного забезпечення. Постійне оновлення антивірусних баз дозволяє вчасно реагувати на нові загрози.

Використання мультифакторної аутентифікації (MFA) є ефективним способом боротьби з фішинговими атаками та зловживанням доступом, оскільки ускладнює процес авторизації для сторонніх осіб. Цей інструмент забезпечує доступ до системи, тільки якщо користувач підтвердить свою особу за допомогою декількох факторів, таких як пароль, біометричний або фізичний ключ. Цей метод значно підвищує рівень безпеки, адже навіть при компрометації одного фактора, система залишається захищеною завдяки додатковим перевіркам.

Системи виявлення та запобігання вторгнень (IDS та IPS) допомагають забезпечити постійний моніторинг мережі та активний захист від підозрілих дій [7]. Системи виявлення вторгнень пасивно аналізують мережевий трафік для виявлення потенційно шкідливої активності та негайно повідомляють про загрози. Вони ефективні для раннього виявлення загроз, дозволяючи адміністраторам вчасно вживати заходів. IDS часто застосовуються для моніторингу критичних мережевих сегментів та аналізу лог-файлів. В свою чергу системи запобігання вторгнень працюють на випередження, активно блокуючи підозрілі дії. Вони автоматично реагують на загрози, запобігаючи можливому вторгненню. IPS є дуже ефективним інструментом для запобігання атак у режимі реального часу, захищаючи систему до того, як загроза зможе завдати шкоди. Завдяки цьому IPS часто використовуються на периметрі мереж для забезпечення надійного захисту від зовнішніх загроз.

Мережевий моніторинг і аудит безпеки – ключові методи для забезпечення інформаційної безпеки, які допомагають підтримувати систему в актуальному стані та запобігати порушенням. Відслідковування мережевого трафіку в реальному часі дозволяє виявляти та аналізувати підозрілу активність, що може вказувати на потенційні загрози, такі як підозрілі з'єднання або зміни в обсягах трафіку. Цей інструмент особливо ефективний для запобігання атак, що розвиваються поступово або мають складний характер, наприклад, атак типу DDoS або мережевого сканування. Аудит інформаційної безпеки включає періодичну перевірку систем на

відповідність встановленим стандартам безпеки, тестування вразливостей та аналіз заходів захисту. Аудити допомагають виявляти прогалини у політиках безпеки, покращувати інфраструктуру захисту та знижувати ймовірність успішної атаки.

Правові та нормативні інструменти – це не менш важливі заходи для забезпечення інформаційної безпеки, ніж вищезазначені. Вони регулюють питання захисту даних на законодавчому рівні та допомагають організаціям дотримуватись обов’язкових стандартів і вимог, що значно знижує ризики витоків та порушень у сфері персональних даних. Існують різні міжнародні стандарти, що зазвичай містять детальні вимоги до обробки та захисту персональних даних. Вони зобов'язують організації впроваджувати комплексні заходи безпеки та відповідати високим стандартам обробки даних. Також уряди різних країн приймають нормативно-правові акти, що встановлюють для організацій певні стандарти безпеки даних. Наприклад, закон України «Про захист персональних даних» регулює порядок збору, зберігання та обробки персональної інформації громадян. Він визначає обов’язки організацій щодо захисту даних, за невиконання яких передбачені адміністративні санкції. Закон покликаний забезпечити прозорість обробки даних і безпеку персональної інформації [8].

Комплексний підхід до забезпечення інформаційної безпеки є необхідною умовою захисту інформаційних систем. Систематичне поєднання цих заходів дозволяє створити надійну багаторівневу систему, здатну ефективно виявляти, запобігати та реагувати на загрози безпеці, забезпечуючи стабільну роботу та конфіденційність даних.

1.4 Децентралізація та технологія блокчейн як варіанти вирішення проблеми забезпечення безпеки інформації та даних

Технологія блокчейн забезпечує значний захист даних завдяки своїй децентралізованій структурі. У централізованих системах дані зберігаються

на одному сервері або в єдиній точці, що створює вразливість для хакерських атак або фізичних пошкоджень серверів. У децентралізованих мережах, таких як блокчейн, дані розподіляються між численними вузлами, кожен із яких зберігає копію загального реєстру [9]. Це робить злом системи вкрай складним, оскільки для зміни даних необхідно контролювати більшість вузлів, що практично неможливо у великих мережах. Основні переваги, що характеризують блокчейн, як безпечну та децентралізовану технологію це – прозорість, конфіденційність, розподіленість і незмінність даних, можливість автоматизації процесів та відсутність цензури даних.

Однією з найважливіших переваг блокчейну є прозорість. Кожна транзакція записується у відкритий реєстр, доступний для перевірки всіма учасниками мережі [10]. Така прозорість сприяє зменшенню можливостей для шахрайства, оскільки будь-яка транзакція або зміна даних може бути перевірена сторонніми особами.

Смарт-контракти в блокчейні забезпечують автоматизацію виконання умов угод без залучення третіх сторін. Смарт-контракт – це програмний код, який автоматично виконує умови угоди, якщо виконані всі попередньо задані критерії. Це дозволяє зменшити ризик шахрайства, оскільки всі операції виконуються автоматично, що мінімізує ймовірність людських помилок або свідомого викривлення даних.

Додатковою важливою перевагою технології блокчейн є її розподіленість і незмінність даних. Така архітектура унеможливлює маніпуляції з даними, що особливо важливо для забезпечення довіри в децентралізованих системах [11]. Незмінність даних означає, що інформація, одного разу записана в блокчейн, не може бути змінена або видалена. Це досягається шляхом використання криптографічних хеш-функцій, що захищають кожен блок інформації.

Блокчейн також забезпечує цифрову ідентифікацію та конфіденційність, дозволяючи користувачам зберігати особисті дані в зашифрованому вигляді і контролювати доступ до них. Завдяки

децентралізованій структурі користувачі мають більше контролю над своїми даними і можуть самостійно керувати доступом до них, що робить блокчейн ефективним інструментом для захисту конфіденційності.

У децентралізованих мережах практично неможливо здійснити цензуру даних, адже інформація зберігається на багатьох вузлах, незалежних від єдиного контролюючого суб'єкта. Це означає, що жоден користувач чи організація не можуть одноосібно змінювати чи видаляти інформацію, що зберігається у блокчейні. Таким чином, децентралізація забезпечує захист від несанкціонованого втручання та додає надійності збереженню інформації.

Децентралізація блокчейн-технології забезпечує універсальний захист і стійкість даних до зовнішніх втручань, що робить її майже ідеальною для багатьох сучасних галузей, де необхідні довіра, безпека та прозорість у процесах обробки даних.

2 ДЕЦЕНТРАЛІЗАЦІЯ, ТЕХНОЛОГІЯ БЛОКЧЕЙН, МЕРЕЖА ETHEREUM ТА ЇЇ СТАНДАРТИ

2.1 Децентралізація в блокчейн-технологіях: визначення і ключові принципи

Децентралізація в блокчейн-технологіях – це процес розподілу обов'язків управління та зберігання даних між кількома вузлами мережі, без залежності від центрального органу [12]. У таких системах кожен вузол виконує роль самостійного учасника, який може підтверджувати транзакції та зберігати копії даних. Децентралізація забезпечує стійкість до атак, прозорість і високу надійність системи. Блокчейн Ethereum є одним із провідних прикладів децентралізованої технології. Його архітектура забезпечує функціонування смарт-контрактів, розподіляючи обчислення та зберігання даних між тисячами вузлів.

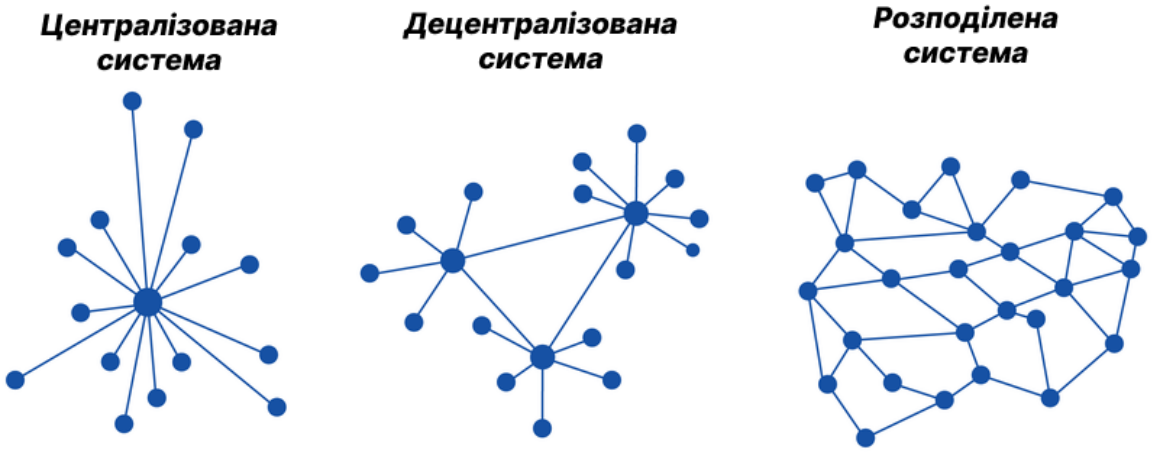

Рисунок 2.1 – Візуалізація різних типів управління в системі

Далі необхідно виокремити ключові принципи роботи децентралізації в блокчейн-системах:

- автономія учасників. Визначає, що кожен вузол мережі може функціонувати незалежно, забезпечуючи сталість процесів навіть за умови виходу з ладу інших вузлів;

- відсутність єдиної точки відмови. Усі дані зберігаються у розподіленій мережі, що значно підвищує її стійкість до атак або технічних проблем;

- рівноправність вузлів. Кожен учасник має рівний доступ до системи і можливість взаємодіяти з нею на рівних умовах;

- транспарентність. Усі транзакції фіксуються у відкритому реєстрі, доступному для перевірки. Це підвищує довіру користувачів;

- стійкість до цензури. Децентралізована система позбавлена контролю з боку будь-якої третьої сторони, що унеможливлює блокування або цензурування.

Централізовані системи, на відміну від децентралізованих, залежать від одного учасника системи, який керує всіма процесами. Прикладом централізованої системи може бути банківська мережа, де всі транзакції проходять через центральний сервер. У такій архітектурі є переваги, зокрема швидке прийняття рішень і простота в управлінні, але вона має серйозні недоліки: ризик єдиної точки відмови і таким чином уразливість до атак. У децентралізованих системах немає єдиного контролюючого вузла. Рішення приймаються за згодою учасників мережі. Це усуває ризик «єдиної точки відмови» і робить систему стійкішою до атак. Однак такі системи складніші в розробці та управлінні.

Децентралізовані системі складні в реалізації, що є одним із мінусів цього підходу. Також через необхідність консенсусу між вузлами обробка та виконання транзакції може займати більше часу, що виокремлює інший мінус децентралізації – менша швидкість транзакцій, порівняно із централізованими системами. Іншим недоліком є значні витрати на ресурси. Зберігання та обробка даних у розподілених мережах вимагає значних обчислювальних потужностей. Неявним недоліком децентралізації є

проблеми управління. Відсутність централізованого контролю ускладнює узгодження дій учасників. Хоча у децентралізованого підходу є певні недоліки, визначені переваги роблять його ключовою концепцією в розвитку сучасних блокчейн-технологій.

2.2 Блокчейн-технологія Ethereum

Блокчейн – це децентралізована, розподілена цифрова база даних (реєстр), яка дозволяє безпечне зберігання, обмін і верифікацію інформації між учасниками мережі без необхідності довіреного посередника [13]. Основою блокчейн-технології є послідовно пов’язані блоки даних, що захищені криптографічними алгоритмами (хеш). Кожен блок містить інформацію про транзакції, криптографічний хеш попереднього блоку і часову позначку, забезпечуючи незмінність записів та їх послідовність. Нижче продемонстровано загальну структуру блокчейн системи:

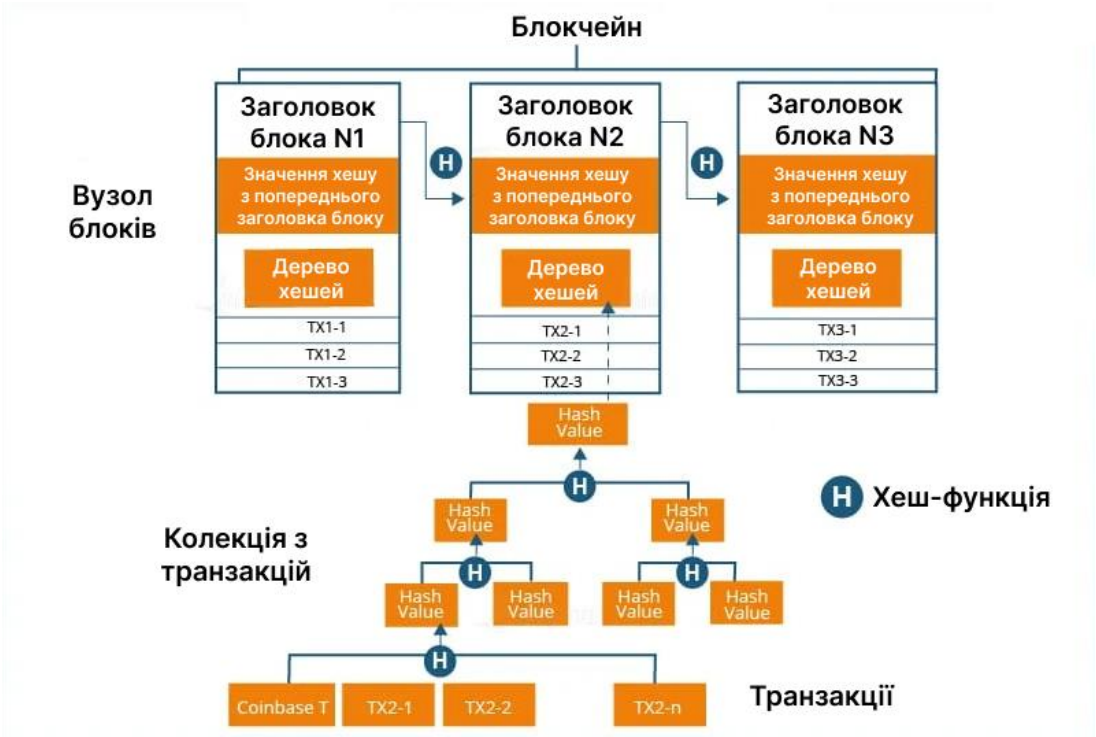

Рисунок 2.2 – Модель блокчейн системи

Однією з головних особливостей блокчейну є його розподілений характер: кожен вузол у мережі зберігає повну копію реєстру і всі зміни в ньому підтверджуються консенсусом між учасниками [14]. Така архітектура знижує залежність від центрального органу і підвищує стійкість до зовнішніх атак, адже дані неможливо змінити без згоди більшості вузлів.

Перша в світі успішно реалізована блокчейн-система є Bitcoin, представлена Сатоші Накамото в 2008 році. Ця система є яскравим прикладом децентралізованої цифрової валюти. У цій системі блокчейн слугує основою для зберігання даних про всі транзакції з біткоїнами. Головною інновацією Bitcoin стала можливість створення довіри між учасниками мережі без посередників завдяки використанню механізму консенсусу Proof of Work (PoW).

Блокчейн Bitcoin побудований таким чином, щоб забезпечити максимальну безпеку та незмінність даних, однак його функціонал обмежений виключно фінансовими операціями. Це створило передумови для розробки інших блокчейн-систем із ширшими можливостями.

Наступним дотепер успішним блокчейном є Ethereum, він був представлений у 2015 році та впродовж короткого часу став другим за популярністю блокчейном і ключовим прикладом розвитку блокчейн-технологій. Основна відмінність Ethereum від Bitcoin полягає у впровадженні смарт-контрактів – автоматизованих програм, що виконуються на блокчейні за визначених умов [15]. Завдяки цьому Ethereum перетворився на платформу для створення децентралізованих застосунків.

Ethereum також базується на принципі децентралізації. Його мережа має розподілений характер, де всі вузли беруть участь у перевірці та валідації транзакцій. Використання механізму консенсусу Proof of Stake (PoS), який поступово замінює PoW у мережі Ethereum, знижує енергоспоживання і підвищує ефективність роботи системи.

Блокчейн Ethereum працює за тими ж правилами як і будь-який

блокчейн, але він має декілька особливостей. Найголовніша особливість, про яку вже було зазначено, – це смарт-контракти. Друга – це те, що Ethereum використовує власну віртуальну машину (EVM), яка дозволяє виконувати смарт-контракти. EVM є ізольованим середовищем виконання, що забезпечує безпеку й універсальність роботи контрактів. Також на відміну від Біткоїна, який фокусується лише на фінансових транзакціях, Ethereum є платформою для загального призначення. Його блокчейн підтримує децентралізовані обчислення, що дозволяє запускати будь-які додатки, які відповідають правилам EVM. І останнє – це гнучкість стандартів токенів. Ethereum дозволяє створювати власні токени за допомогою стандартів, таких як ERC-20 та ERC-721. Це робить Ethereum платформою для токенізації будь-яких активів. Нижче на рисунку 2.3 представлено модель, за якою виконуються всі операції в мережі Ethereum.

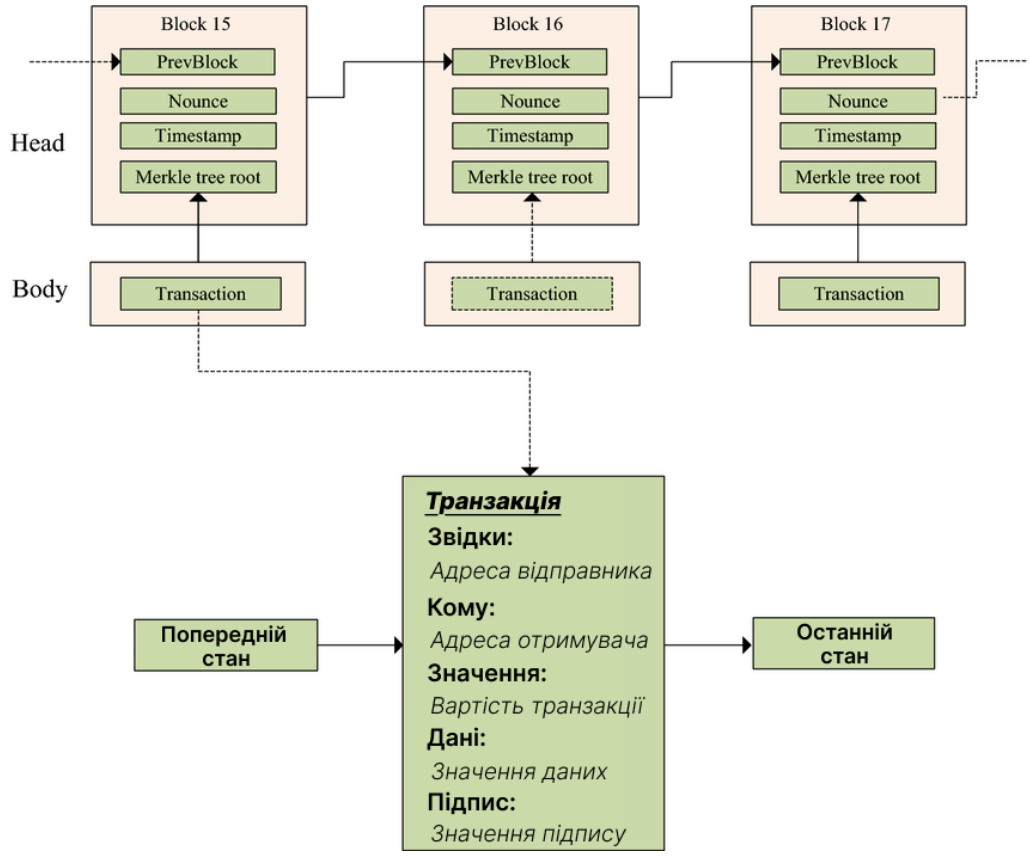

Рисунок 2.3 – Модель блокчейну Ethereum

На схемі моделі блокчейна Ethereum головним елементом цілісності та збереження інформації є блок. Кожен блок містить в собі набір транзакцій і має посилання на попередній блок через хеш, утворюючи таким чином ланцюг. Таким чином, при спробі замінити будь-які дані необхідно вносити зміни і в кожний наступний утворений ланцюг, що в теорії є можливим, але на практиці таких випадків не було.

Блок ділиться на дві частини: заголовок блоку (head), що містить мета інформацію про блок та тіло блоку (body), що зберігає список транзакцій [16]. Заголовок блоку включає кілька ключових елементів:

- prevBlock – хеш попереднього блоку. Він забезпечує зв’язок між блоками та гарантує, що дані є незмінними;

- nonce – унікальне число, яке призначається кожній транзакції від конкретної адреси користувача. Це забезпечує, що транзакції виконуються в правильному порядку та не можуть бути повторені чи підроблені. Простими словами nonce відображає кількість попередніх транзакцій, здійснених з конкретної адреси, що запобігає повторному використанню транзакцій;

- timestamp – час створення блоку, що відображає момент, коли валідатор згенерував цей блок. Timestamp дозволяє впорядковувати події в хронологічному порядку та забезпечує коректну роботу смарт-контрактів і транзакцій;

- merkle tree root – криптографічний хеш, який представляє всі транзакції блоку у вигляді дерева Меркла [17]. Ця структура дає змогу швидко перевіряти, чи міститься певна транзакція в блоці.

Тіло блоку містить список усіх транзакцій, підтверджених і включених у цей блок. Усі транзакції зберігаються у вигляді:

- from – унікальний ідентифікатор, що представляє криптографічну адресу користувача, з гаманця якого відправляється транзакція. Адреса є публічним ключем, створеним із приватного ключа користувача. Адреса відправника завжди повинна мати достатньо коштів (нативної валюти ETH) для покриття як суми переказу, так і комісії за транзакцію (Gas Fee);

- to – унікальний ідентифікатор, що вказує на кінцевого отримувача транзакції. Адреса може належати як звичайному гаманцю користувача, так і смарт-контракту, виконуючи тим самим виклик функцій;

- value – кількість ETH, яку відправник хоче перевести на адресу отримувача. Значення передається у найменшій одиниці виміру Ethereum – Wei (1 ETH = 10^{18} Wei). Якщо транзакція не містить переказу ETH (наприклад, лише виклик смарт-контракту), значення Value може дорівнювати нулю;

- data – поле, що може включати додаткову інформацію або інструкції, які передаються у рамках транзакції. Якщо транзакція виконує виклик смарт-контракту, у Data записуються дані про функцію, яку потрібно викликати, та її параметри. Також через це поле можна ініціювати передачу токенів ERC-20 або взаємодіяти зі смарт-контрактами NFT. Поле Data є необов'язковим;

- signature – криптографічний підпис, створений за допомогою приватного ключа відправника, що підтверджує ініціалізацію транзакції власником приватного ключа, асоційованого з адресою відправника, гарантує цілісність та автентичність.

2.3 EVM та смарт-контракти, принцип їх роботи: механізми та застосування

Ethereum Virtual Machine (EVM) – це децентралізоване обчислювальне середовище, що забезпечує виконання смарт-контрактів у мережі Ethereum [18]. EVM є ядром платформи Ethereum, яке забезпечує її функціональність і гнучкість, перетворюючи Ethereum на платформу загального призначення для створення децентралізованих додатків.

EVM працює як ізольоване середовище для виконання програмного коду, що забезпечує безпеку і стабільність системи. Основні характеристики EVM включають детермінізм, ізоляцію, використання мови програмування Solidity та концепцію «газу». Детермінізм гарантує однакове виконання коду

на кожному вузлі, ізоляція захищає вузли від шкідливих програм, а газ слугує механізмом контролю за використанням ресурсів мережі та стимулюванням ефективності коду. Solidity є мовою, що дозволяє створювати програми, які компілюються в байт-код для виконання в EVM.

Алгоритм роботи EVM (рисунок 2.4) передбачає, що кожен вузол мережі отримує байт-код контракту, виконує його у своєму ізольованому середовищі та обчислює результат, використовуючи різні системні ресурси мережі [19]. Після завершення виконання зміни записуються у блокчейн.

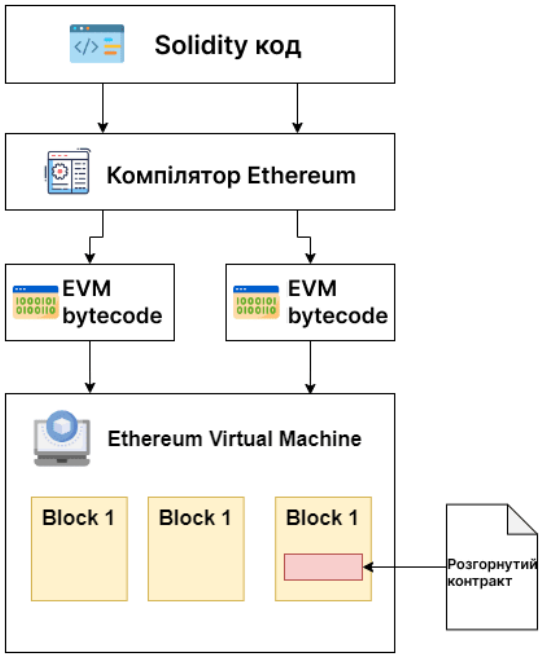

Рисунок 2.4 – Модель виконання операцій через EVM

Смарт-контракти є самовиконуваними програмами на блокчейні Ethereum, які забезпечують автоматизацію дій відповідно до заздалегідь визначених умов. Вони складаються з коду контракту, стану контракту та інтерфейсу для взаємодії з користувачами. Механізм роботи смарт-контрактів включає створення транзакції з кодом, виконання цього коду на вузлах мережі та оновлення стану контракту в разі успішного виконання [20].

Концепція смарт-контрактів бере свій початок із 1994 року, коли

вчений та інформатик Нік Сабо вперше запропонував ідею цифрових протоколів для виконання договорів. Основою цієї концепції є автоматизація договірних зобов’язань та виключення необхідності в довірених посередниках. Попри історичні витоки, реалізація смарт-контрактів у мережі Ethereum більше нагадує програмування універсального характеру, ніж класичні юридичні договори чи документи.

Смарт-контракти забезпечують ефективне управління токенізованими активами та правами доступу між кількома сторонами. Їх можна уявити як криптографічний механізм, що відкриває доступ або активує певну дію за умови виконання попередньо встановлених критеріїв. Попри свої переваги, такі як автоматизація, прозорість та незмінність даних, смарт-контракти мають обмеження. Помилки у коді можуть спричинити втрату коштів, а складність розробки вимагає високої кваліфікації програмістів. Крім того, висока вартість газу для складних операцій може обмежувати масштабування рішень.

EVM та смарт-контракти відкривають нові можливості для створення інноваційних децентралізованих рішень, розширюючи горизонти використання блокчейн-технологій у різних галузях, зокрема у фінансах, логістиці, мистецтві та управлінні даними.

2.4 Модель архітектури децентралізованих застосунків (dApp)

Цифрові застосунки можна поділити на дві категорії: централізовані та децентралізовані. Кожен із цих підходів має унікальну архітектуру, яка визначає спосіб взаємодії користувачів із системою, а також впливає на продуктивність, безпеку та масштабованість додатків. На рисунку 2.5 представлено різницю в архітектурі двох різних підходів:

Рисунок 2.5 – Порівняння двох видів архітектур функціонування додатків

Централізовані застосунки базуються на єдиному сервері, який виконує всі основні функції, такі як зберігання даних, обробка запитів та управління доступом. Клієнтські пристрої, які взаємодіють із централізованим сервером через інтернет-провайдерів (ISP), фактично є залежними від цього центрального вузла [21]. Якщо сервер недоступний або виникає збій, робота додатка переривається для всіх користувачів. Така архітектура, хоча і проста для управління та розробки, але має значні ризики, включаючи вразливість до атак і втрату даних через централізовану точку відмови.

На противагу цьому, децентралізовані додатки функціонують через розподілену мережу, де дані та обчислювальні ресурси розміщені на багатьох вузлах [22]. У цій моделі кожен вузол може взаємодіяти з іншими без необхідності в центральному сервері. Завдяки цьому, навіть якщо один із вузлів виходить із ладу, система продовжує працювати. Така архітектура забезпечує високу стійкість до збоїв, а також більшу прозорість і безпеку.

Детальна модель децентралізованих додатків включає кілька ключових компонентів, які забезпечують їх функціонування. Користувач взаємодіє з dApp через браузер, який відображає інтерфейс програми. Фронтенд, побудований із використанням технологій TypeScript, React, HTML та CSS,

відповідає за взаємодію користувача з застосунком. Цей фронтенд підключається до веб-сервера, але головна особливість dApps полягає в їхній інтеграції з блокчейном через смарт-контракти.

Фронтенд взаємодіє зі смарт-контрактами через спеціальні бібліотеки, такі як Web3.js або Ethers.js. Ці бібліотеки забезпечують зв'язок між інтерфейсом користувача та блокчейном. На практиці взаємодія dApp працює наступним чином: користувач надсилає запит через інтерфейс додатку, фронтенд обробляє його і перетворює на виклик функції смарт-контракту. Цей виклик передається до блокчейну через вузли, які перевіряють і далі виконують код смарт-контракту. Вузли відповідають за верифікацію транзакції та її додавання до блокчейну. Після успішного виконання транзакції смарт-контракт повертає результат, який фронтенд може відобразити у вигляді оновлених даних або повідомлення для користувача. Схематично ця взаємодія зображена на рисунку 2.6.

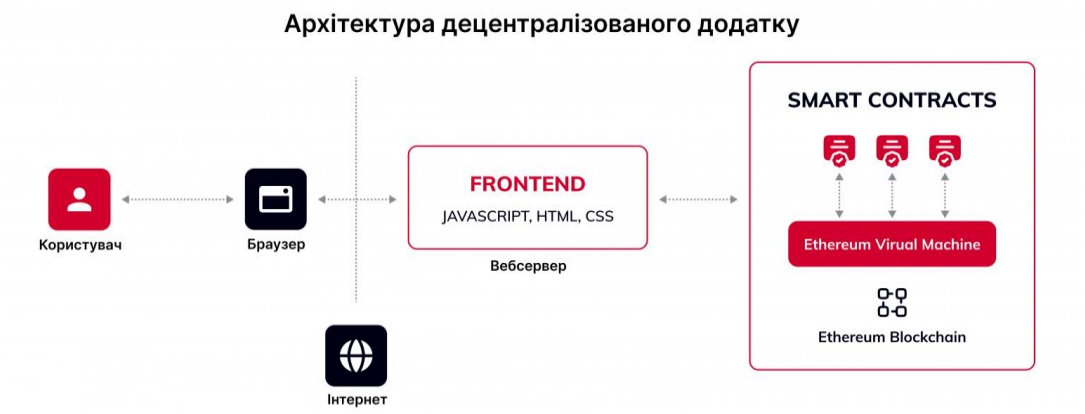

Рисунок 2.6 – Схема взаємодії компонентів у децентралізованих додатках

Додатково децентралізовані застосунки можуть використовувати децентралізовані сховища, такі як IPFS, для зберігання файлів або великих обсягів даних, які не ефективно зберігати безпосередньо в блокчейні. Це дозволяє оптимізувати роботу dApps, розділяючи обробку даних між блокчейном і зовнішніми сховищами.

2.5 Використання стандартів ERC у смарт-контрактах та dApps

Стандарти ERC (Ethereum Request for Comments) є набором технічних специфікацій, які визначають правила та функціонал для смарт-контрактів у блокчейні Ethereum. Вони забезпечують взаємодію між різними застосунками, спрощуючи інтеграцію та використання смарт-контрактів у децентралізованих додатках. Стандарти ERC є основою для створення токенів, що відповідають певним узгодженим критеріям і забезпечують передбачувану поведінку в мережі [23].

Ці набори технічних специфікацій необхідні в першу чергу для того, щоб забезпечити сумісність між додатками та платформами, дозволяючи працювати з різними токенами без перешкод. Вони встановлюють єдині правила для розробників, що сприяє швидшому впровадженню інновацій та знижує ризик виникнення помилок. Крім того, стандарти надають готову архітектуру для створення токенів, спрощуючи процес розробки, а перевірені методи мінімізують ймовірність вразливостей у коді, забезпечуючи його надійність.

Стандарт ERC-20 – це один із перших і найпоширеніших стандартів на платформі Ethereum, який встановлює набір правил для створення та управління взаємозамінними токенами [24]. Взаємозамінність означає, що кожен токен одного типу є ідентичним іншому за властивостями та вартістю. Іншими словами, один токен ERC-20 завжди можна замінити на інший такого ж типу без втрати цінності чи зміни функціоналу. Наприклад, є банкнота номіналом 100 гривень і незалежно від того, якою конкретною банкнотою ви володієте, її вартість залишається однаковою. Є можливим обміняти одну банкноту на іншу такого ж номіналу, і ця дія не змінить функціональності чи вартості цих грошей. Аналогічно працюють токени ERC-20.

Стандарт ERC-20 визначає наступні базові функції:

- balanceOf(address) – показує баланс токенів для вказаної адреси;

- transfer(address, uint256) – переводить токени з одного гаманця на інший;

- approve(address, uint256) – дозволяє іншій адресі витрачати певну кількість токенів;

- transferFrom(address, address, uint256) – передає токени від третьої сторони на основі попереднього дозволу;

- totalSupply() – відображає загальну кількість випущених токенів.

ERC-20 токени часто використовуються для створення криптовалют і цифрових активів у децентралізованих фінансових застосунках (DeFi). Наприклад, популярні криптовалюти, такі як USDT (Tether), LINK (Chainlink) і UNI (Uniswap), створені відповідно до стандарту ERC-20. Усі ці токени можуть взаємодіяти між собою та з іншими смарт-контрактами на платформі Ethereum завдяки їхній взаємозамінності й дотриманню єдиного стандарту.

Наступний не менш важливий стандарт ERC-721, розроблений для створення невзаємозамінних токенів (NFT). Кожен токен, який створений за цим стандартом має наступні ключові характеристики: цілісність, невзаємозамінність, змогу бути переданими або бути у власності [25]. Токени стандарту ERC-721 не можуть бути поділені на частини на відміну від ERC-20.

Найкращим чином можна зрозуміти характеристики та функціональність цього стандарту по аналогії з реальним життям. Наприклад, оригінальна картина відомого художника Казимира Малевича «Чорний супрематичний квадрат» не може бути замінена іншою картиною, навіть якщо вона здається ідентичною за написанням. Ця картина також не може бути розрізана або поділена, оскільки в такому випадку вона втратить свої властивості й цінність. Отже, кожна картина має свою унікальність, історію та художню цінність, які визначають її відмінність від інших. Так само й токени ERC-721: вони використовуються для представлення унікальних за своєю природою цифрових чи фізичних активів, зокрема, творів мистецтва, ігрових предметів, нерухомості або колекційних предметів.

Стандарт ERC-721 має наступні базові функції:

- ownerOf(uint256 tokenId) – визначає власника конкретного токена;
- transferFrom(address from, address to, uint256 tokenId) – передає токен від одного користувача до іншого;
- approve(address to, uint256 tokenId) – дозволяє іншій адресі виконати передачу токена;
- getApproved(uint256 tokenId) – повертає адресу, яка має дозвіл на управління токеном;
- totalSupply() – показує загальну кількість токенів.

Інтерфейс ERC-721 використовує стандартні контракти IERC721 та IERC721Metadata, які визначають функції управління токенами та зберігання їх метаданих. Як правило, метадані токенів зберігаються в URI, що дозволяє мати доступ до пов’язаної інформації поза блокчейном. Також важливо зазначити, що велика кількість операцій з NFT може перевантажувати мережу Ethereum, збільшуючи вартість газу.

ERC-721 працює в dApps через інтеграцію з фронтендом. Інтерфейс сайту побудований на JavaScript, використовує бібліотеки, такі як Web3.js чи Ethers.js, для взаємодії зі смарт-контрактами. Виклики функцій контракту виконуються через API, який з’єднує користувача з блокчейном через провайдери. Смарт-контракти обробляють запити від користувачів, такі як купівля чи передача NFT, а дані про токени записуються в блокчейн, забезпечуючи їхню незмінність. Додаткові дані про токени, наприклад, зображення чи атрибути, зберігаються поза блокчейном у таких системах, як IPFS, а їхній URI додається до токена.

У цьому розділі було детально розглянуто основи децентралізації, блокчейн-технології, особливості мережі Ethereum та її стандарти. Децентралізація визначена як ключовий принцип сучасних блокчейн-систем, що забезпечує прозорість, надійність та захищеність даних завдяки розподіленій архітектурі. Було визначено, що хоча децентралізовані системи мають свої недоліки, такі як складність управління та високе споживання

ресурсів, їх переваги, включаючи стійкість до цензури та відсутність єдиної точки відмови, превалюють над обмеженнями та ризиками.

Блокчейн Ethereum, завдяки впровадженню смарт-контрактів та універсальній віртуальній машині, став платформою для створення децентралізованих застосунків і токенізації активів. Його гнучкість, підтримка стандартів токенів ERC-20 та ERC-721, а також перехід до механізму консенсусу Proof of Stake підвищують ефективність та енергетичну сталість мережі.

Розглянуті механізми Ethereum демонструють важливість використання децентралізованих технологій для розробки інноваційних рішень у різних сферах. Таким чином, мережа Ethereum є універсальною платформою для децентралізованих задач завдяки своїй прозорості, безпеці та гнучкості. Її використання дозволяє створювати ефективні та надійні додатки, які відповідають сучасним вимогам щодо безпеки даних. мережа Ethereum [26].

3 РОЗРОБКА МОДЕЛІ ДЕЦЕТРАЛІЗОВАНОГО ЗБЕРІГАННЯ ДАНИХ, СМАРТ-КОНТРАКТУ ТА КОМПОНЕНТІВ ЗАСТОСУНКУ

3.1 Обґрунтування вибору середовища програмної реалізації

Розробка смарт-контракту здійснювалася з використанням середовища для розробки Visual Studio Code. Основною мовою програмування для реалізації функціоналу смарт-контракту була обрана Solidity, яка є стандартом у сфері розробки децентралізованих додатків на базі Ethereum.

Solidity – це високорівнева об’єктно-орієнтована мова програмування, спеціально створена для розробки смарт-контрактів на платформі Ethereum. Вона дозволяє розробникам створювати смарт-контракти, які виконуються у середовищі блокчейну [27]. Solidity має потужні можливості для написання ефективного та надійного коду.

Синтаксис Solidity схожий із такими мовами програмування, як JavaScript і C++, що робить цю мову зрозумілою для досвідчених розробників. Вона побудована на об’єктно-орієнтованій моделі, яка забезпечує структуровану організацію коду та використовує принципи інкапсуляції, поліморфізму і спадкування. Завдяки механізму спадкування можна створювати базові контракти з універсальними функціями та властивостями, які легко розширювати в дочірніх контрактах. Крім того, мова підтримує інтерфейси, що дозволяють ділити логіку на автономні модулі, спрощуючи подальшу підтримку та розвиток програмного коду.

Оскільки Solidity є мовою зі статичною типізацією, кожна змінна повинна бути явно оголошена розробником. Це дозволяє компілятору перевіряти коректність використання змінних і виявляти потенційні помилки ще на етапі розробки [28]. Статична типізація сприяє підвищенню безпеки смарт-контрактів, оскільки зменшує ймовірність випадкових помилок, пов'язаних із неправильним типом даних, що особливо важливо у фінансових

застосунках, де навіть незначна помилка може призвести до серйозних наслідків.

Solidity також забезпечує високий рівень безпеки завдяки вбудованим механізмам, які допомагають уникати вразливостей і гарантувати стабільну роботу смарт-контрактів. Наприклад, обмеження на споживання газу зменшує ризик його неконтрольованого використання. Крім того, Solidity виконує функції бібліотеки для перевірки типів та управління даними, що значно підвищує надійність контрактів.

Visual Studio Code – це популярне середовище розробки, яке широко використовується для написання смарт-контрактів і супутнього програмного забезпечення [29]. Завдяки своїй універсальності, середовище підтримує мову програмування Solidity через спеціальні розширення, які додають функціональність для написання, компіляції та налагодження смарт-контрактів. Це середовище дозволяє організовувати код у великих проєктах, ефективно управляти залежностями та інтегруватися з різними інструментами, такими як Hardhat або Truffle, що забезпечують комплексний підхід до розробки.

Однією з головних переваг Visual Studio Code є його модульність. Встановлюючи розширення, можна додати такі функції, як автозаповнення коду, динамічний аналіз помилок, підсвічування синтаксису Solidity та інтеграція з Git для контролю версій.

Додатково було використано React у поєднанні з бібліотеками Web3 для створення інтерактивного фронтенду, що взаємодіє зі смарт-контрактами в реальному часі. Використання мови програмування TypeScript забезпечило типізацію, що підвищило безпеку та надійність коду. Для зручної роботи з формами застосовано бібліотеку React Hook Form, а спеціалізовані TypeScript-бібліотеки значно полегшили інтеграцію та тестування функціоналу смарт-контрактів.

TypeScript – це мова програмування з підтримкою статичної типізації, яка є надбудовою над JavaScript. Її основною перевагою є можливість

визначення типів, що дозволяє уникати багатьох помилок під час розробки та забезпечує надійність коду [30]. TypeScript широко використовується у проєктах, які вимагають складної логіки та інтеграції з різноманітними інструментами. У межах цього проєкту TypeScript був обраний для створення фронтенду завдяки його сумісності з React та екосистемою Web3.

React – це популярний фреймворк для створення інтерфейсів, що дозволяє розробляти динамічні та інтерактивні застосунки [31]. У цьому проєкті React став основним інструментом для розробки фронтенд-частини додатка. Його компонентна структура дозволила розділити код на модулі, що спрощує підтримку та розширення функціональності. Крім того, React надає високий рівень продуктивності завдяки віртуальному DOM, що мінімізує кількість оновлень на сторінці.

Для інтеграції з смарт-контрактами у проєкті було використано бібліотеку Web3.js, яка забезпечує взаємодію фронтенду з блокчейном. Web3.js дозволяє виконувати транзакції, підключатися до криптогаманців і зчитувати дані зі смарт-контрактів. У поєднанні з TypeScript ця бібліотека забезпечила чітке визначення типів для взаємодії з контрактами, що значно підвищило безпеку та стабільність застосунку.

React Hook Form був обраний як бібліотека для роботи з формами завдяки її простоті у використанні та високій продуктивності. Вона забезпечила зручний спосіб створення, валідації та обробки даних із форм, що є важливим аспектом у додатках, які вимагають взаємодії з користувачем. Завдяки можливостям React Hook Form вдалося скоротити кількість коду та покращити зручність користувацького інтерфейсу.

Додатково, у проєкті застосовувалися спеціалізовані бібліотеки для роботи зі смарт-контрактами, такі як ethers.js. Ця бібліотека дозволила ефективно працювати з контрактами, виконувати виклики функцій і обробляти транзакції. У поєднанні з React та TypeScript ethers.js забезпечила гнучкість і надійність інтеграції.

Також у цьому проєкті застосовано бібліотеки OpenZeppelin та

NFT.Storage для надійної реалізації основних функцій смарт-контрактів. Зокрема, бібліотека OpenZeppelin забезпечує перевірені реалізації стандартів ERC, що дозволяє значно спростити розробку, підвищити безпеку та відповідність стандартам блокчейну. Бібліотека NFT.Storage дозволяє зберігати файли і метадані у децентралізованій файловій системі.

OpenZeppelin – це популярна бібліотека для розробки смарт-контрактів на платформі Ethereum, яка забезпечує перевірену та безпечну реалізацію стандартів, таких як ERC-20, ERC-721 та інших [32]. Вона дозволяє значно зменшити обсяг коду, який розробники повинні писати вручну, і мінімізує ризик помилок завдяки використанню готових модулів, які пройшли аудит безпеки.

Однією з основних переваг OpenZeppelin є її модульна структура, яка дозволяє розробникам легко розширювати функціональність контрактів. Наприклад, базові класи, такі як ERC20 чи ERC721, містять уже реалізовані функції для роботи з токенами, такі як передача, затвердження чи перевірка балансу. Це дозволяє швидко реалізувати необхідний функціонал, забезпечуючи відповідність контрактів стандартам Ethereum.

NFT.Storage – це спеціалізована бібліотека для роботи з NFT, яка забезпечує простий спосіб зберігання медіафайлів і метаданих у децентралізованій файловій системі IPFS (InterPlanetary File System) із гарантією довготривалого збереження через мережу Filecoin [33]. Бібліотека створена для розробників, які прагнуть використовувати децентралізоване зберігання, щоб забезпечити надійність і доступність даних своїх NFT.

Однією з ключових переваг NFT.Storage є її здатність автоматично завантажувати файли до IPFS. Всі файли, завантажені через бібліотеку, отримують унікальний ідентифікатор, який використовується для доступу до контенту. Це дозволяє легко інтегрувати збережені дані у смарт-контракти та забезпечувати відповідність метаданих стандартам, наприклад, ERC-721 чи ERC-1155. Бібліотека також підтримує простий API, який дозволяє розробникам завантажувати або отримувати результати файлів чи метаданих

у форматі JSON.

У якості інтерфейсу для обробки транзакцій було використано криптогаманець MetaMask. Hardhat був задіяний для локального тестування та емуляції блокчейн-середовища, що дозволило ефективно перевіряти логіку роботи смарт-контракту до його розгортання. Перевірка роботи смарт-контракту та ручне тестування виконувалися за допомогою блокчейн-експлорера Etherscan, який дозволяє детально аналізувати та моніторити транзакції.

Metamask є криптовалютним гаманцем, який функціонує як розширення для веббраузерів і мобільний додаток. Він забезпечує зручний спосіб взаємодії з блокчейнами, смарт-контрактами та децентралізованими застосунками без необхідності встановлення додаткового програмного забезпечення [34].

Однією з ключових особливостей Metamask є підтримка EVM блокчейнів та L2 рішень, зокрема Ethereum, Binance Smart Chain, ZK Sync Era, Polygon та інших. Користувачі можуть додавати нові мережі, працювати з токенами різних стандартів і виконувати транзакції між ними. Це робить Metamask універсальним інструментом для управління криптоактивами та взаємодії з децентралізованими додатками чи NFT-маркетплейсами.

У рамках цього проєкту він був обраний для здійснення транзакцій та тестування смарт-контрактів.

3.2 Модель та алгоритм роботи децентралізованого застосунку

Децентралізований застосунок є складною системою, де всі компоненти взаємопов’язані та залежать один від одного. Важливою частиною роботи цієї системи є інтеграція з зовнішньою розподіленою файловою системою IPFS, яка дозволяє зберігати дані децентралізовано. На рисунку 3.1 зображено модель роботи децентралізованого додатку.

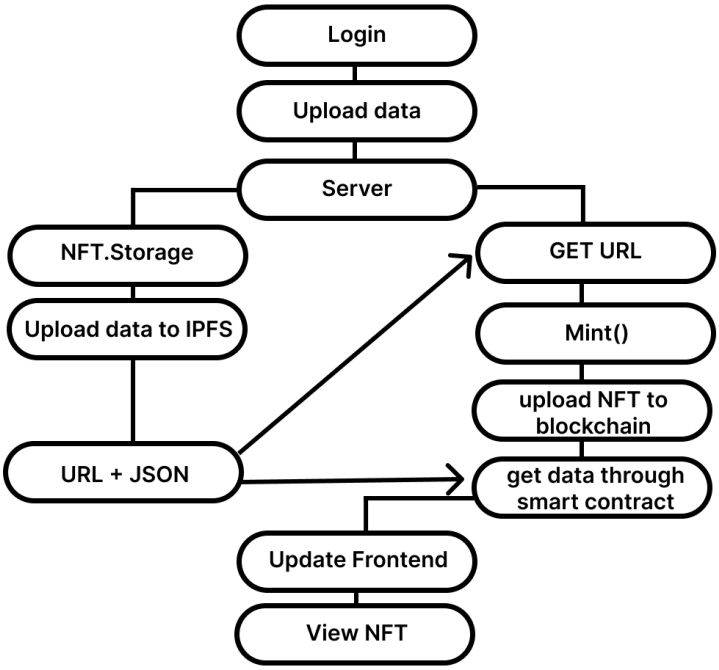

Рисунок 3.1 – Модель децентралізованого додатку

Технологія IPFS забезпечує високу доступність, захист від цензури та масштабованість, що критично важливо для зберігання метаданих NFT. Взаємодія блокчейну з IPFS відбувається через передачу унікального URL, що вказує на місце збереження даних у IPFS. Цей URL зберігається у смарт-контракті на блокчейні, забезпечуючи постійний зв'язок між токеном і його метаданими.

Уся система складається з наступних компонентів: користувач, фронтенд, сервер, IPFS, смарт-контракт та блокчейн Ethereum.

Взаємодія всіх компонентів працює за певним алгортимом, що необхідно для коректної роботи застосунку.

Алгоритм роботи додатку необхідний для структурування процесів, що виконуються під час взаємодії з децентралізованим застосунком. Він забезпечує чітке розуміння кожного етапу роботи системи: від початкового введення даних до створення NFT і відображення результату. На рисунку 3.2 продемонстровано алгоритм роботи застосунку.

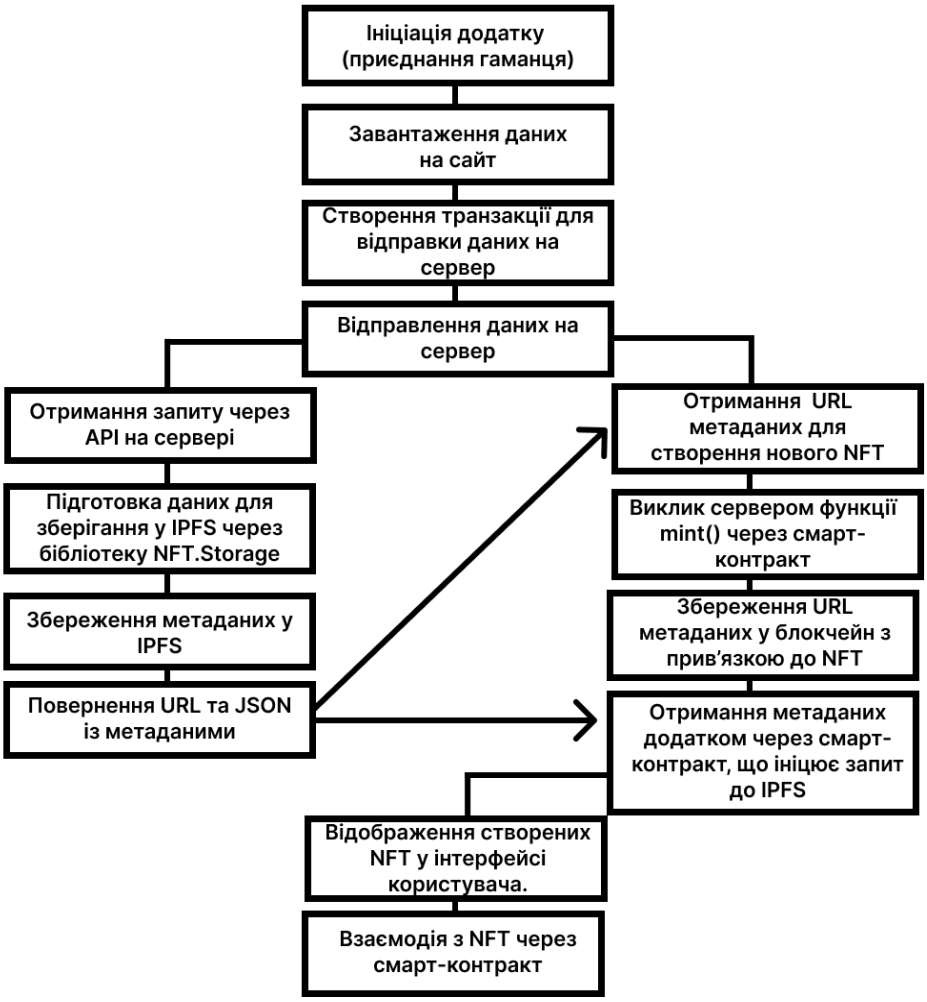

Рисунок 3.2 – Алгоритм роботи децентралізованого додатку

Отже, згідно з алгоритмом роботи додаток має наступні етапи роботи:

1. авторизація – користувач підключає свій криптогаманець до застосунку для отримання доступу до функціоналу;

2. завантаження даних – користувач заповнює форму, де вказує назву, опис NFT і завантажує файл;

3. відправлення даних на сервер. Дані передаються на сервер через API-запит, що обробляє форму користувача;

4. обробка даних на сервері. Сервер конвертує зображення у байтовий формат і формує JSON-об'єкт метаданих;

5. збереження метаданих у IPFS. Метадані надсилаються до IPFS

через бібліотеку NFT.Storage. У відповідь сервер отримує унікальний URL метаданих;

6. виклик смарт-контракту. Сервер ініціює транзакцію у блокчейні Ethereum, викликаючи функцію mint() смарт-контракту. URL метаданих передається як параметр для створення NFT;

7. збереження даних у блокчейні. Смарт-контракт створює NFT і зберігає його унікальний ідентифікатор разом із URL метаданих у блокчейні;

8. відображення результатів. Сервер повертає користувачу ID створеного токена. Застосунок оновлює інтерфейс, звертаючись до IPFS для відображення даних NFT.

3.3 Загальна структура децентралізованого застосунку

Проєкт містить чотири головні папки – contracts, scripts, src та server.

Репозиторій contracts містить лише один файл смарт-контракту із назвою DistributedStorage.sol, який реалізує логіку для створення і управління NFT токенами. Логіку контракту більш детально буде розглянуто пізніше.

Наступна папка scripts включає у себе два ключових файли проєкта. Загалом ця папка необхідна для автоматизації процесів роботи з блокчейном. Перший файл deploy.ts містить скрипт для розгортання смарт-контракту на блокчейні за допомогою фреймворку Hardhat. Цей файл автоматизує процес завантаження смарт-контракту в мережу блокчейну, що є важливим кроком під час реалізації блокчейн-застосунків. Другий файл з назвою mint.ts необхідний для автоматизації процесу створення токенів стандарту ERC-721. Скрипт у цьому файлі завантажує данні в децентралізоване сховище і створює відповідні NFT, прив'язані до метаданих цих файлів.

В папці src зберігаються файли, які реалізовують основну логіку роботи програми та інтерфейс користувача, включаючи компоненти, модулі та допоміжні утиліти. Структуру папки src можна оглянути на рисунку 3.3.

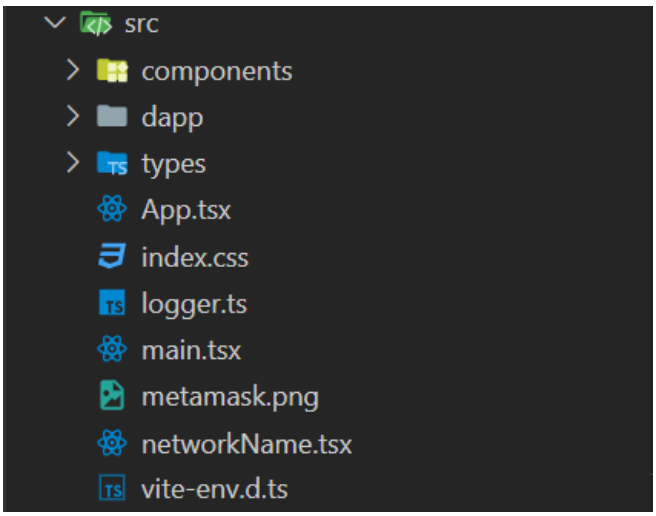

Рисунок 3.3 – Структура папки src

Детальніше, для чого потрібна кожна з папок:

- папка components містить в собі окремі модулі та компоненти, які є базовими елементами інтерфейсу чи логіки роботи програми. Ці компоненти багаторазово використовуються у проєкті;

- папка dapp містить файли, необхідні для інтеграції з блокчейн-мережами. В ній містяться файли, які забезпечують підключення програми до блокчейну;

- папка types використовується для зберігання типізації, яка допомагає коректно працювати з кодом TypeScript і забезпечує безпомилкове використання змінних та функцій;

- файл App.tsx – це головний компонент додатку, який об’єднує всі частини програми в єдине ціле;

- файл main.tsx є вхідною точкою програми, яка ініціалізує основний рендеринг компонентів React;

- index.css – це файл стилів, який використовується для оформлення інтерфейсу застосунку.

Останньою важливою папкою проєкту є server, у ній зберігаються файли, які реалізують серверну частину проєкту, включаючи налаштування серверу, маршрути для роботи API, набір для проведення тестів та

конфігураційні файли. Структуру папки server можна детально оглянути на рисунку 3.4.

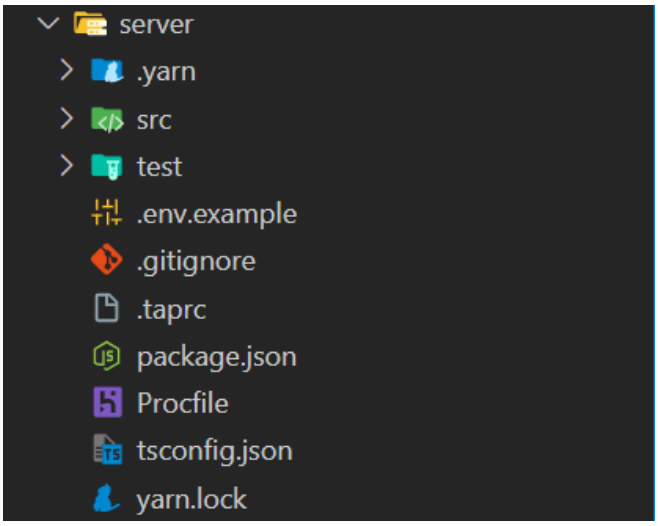

Рисунок 3.4 – Структура папки server

Якщо цю папку розглядати більш детально, то вона має наступну структуру:

- папка src містить основні файли логіки серверу, які забезпечують функціонування API та взаємодію з клієнтською частиною;
- папка test забезпечує тестування серверної логіки, включаючи API-маршрути;
- файл .env.example відповідає за конфігурацію змінних середовища, необхідних для коректної роботи серверу;
- файл package.json описує залежності серверної частини проєкту, а також містить скрипти для запуску і тестування.

3.4 Реалізований смарт-контракт та опис його роботи

У рамках розробки додатку був створений смарт-контракт, який реалізує функціональність для роботи з NFT відповідно до стандарту ERC-721. Для спрощення роботи з контрактом використовуються бібліотеки

OpenZeppelin, які надають перевірені рішення для основних функцій, таких як зберігання метаданих та управління правами власності.

Контракт реалізовано у файлі DistributedStorage.sol з використанням парадигми об'єктно-орієнтованого програмування та він має наступні ключові можливості:

- створювати NFT – за цей процес відповідає функція mint(). Це дозволяє генерувати унікальні токени та прив’язувати до них метадані;
- зберігати метадані у вигляді URI (унікальний ідентифікатор ресурсу);
- спалювати або знищувати NFT за допомогою функції burn();
- управляти дозволом на власність токеном. У контракті використовуються методи для визначення власника NFT та передачі між користувачами;
- вести облік кількості створених токенів за допомогою бібліотеки Counters.

Завдяки набору імпортів із бібліотеки OpenZeppelin (лістинг 3.1) написаний смарт-контракт має додаткову функціональність. Імпорт файлу ERC721.sol із смарт-контрактом містить базову реалізацію стандарту ERC-721 для створення NFT, що надає основні функції, такі як створення, передача та управління токенами [35]. Імпортований файл ERC721URIStorage.sol представляє розширення для зберігання URI (метаданих) для кожного токена. Це дозволяє додавати інформацію про токени у вигляді метаданих, які можна використовувати для зберігання даних про зображення, опис або інші характеристики токенів.

Лістинг 3.1 – Набор імпортів із бібліотеки OpenZeppelin

```
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import
"@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorag
e.sol";
```

Завдяки чітко організованій файловій структурі інтеграція смарт-контракту з додатком забезпечує зручну взаємодію між фронтендом та блокчейном. Як було зазначено вище, код контракту знаходиться в папці contracts, а згенеровані типи та інтерфейси для інтеграції з ним зберігаються в папці за маршрутом – src\types. Це дозволяє коректно зв'язати фронтенд з блокчейн-логікою контракту.

Підключення контракту до клієнтської частини реалізовано через файл DistributedStorage.ts, в якому містяться автоматично згенеровані типи функцій. Це спрощує взаємодію з контрактом через бібліотеку Ethers.js.

Згенеровані типи та інтерфейси для взаємодії з контрактами знаходяться в папці src\types\factories, що дозволяє забезпечити типізацію та зручність у роботі з контрактами на всіх етапах розробки. Підключення до контрактів здійснюється через фабричний клас, згенерований для кожного контракту. Цей клас забезпечує доступ до функцій контракту.

Користувачі застосунку можуть створювати токени, завантажуючи дані через інтерфейс додатка та переглядати адресу власника токенів за допомогою функцій ownerOf та balanceOf, а також передавати NFT між акаунтами за допомогою методів transferFrom або safeTransferFrom.

3.4.1 Інтерфейс смарт-контракту

Смарт-контракт складається з набору змінних, які описують його поточний стан, та функцій, що реалізують логіку. Інтерфейс, що описаний у файлі DistributedStorage.ts забезпечує доступ до основних функцій та подій смарт-контракту, які дозволяють взаємодіяти з токенами, змінювати їхній стан та отримувати інформацію про них.

Інтерфейс смарт-контракту успадковує функціональність базового класу utils.Interface (лістінг 3.2). Цей клас дозволяє працювати з функціями та подіями та використовується для взаємодії з ABI смарт-контракту.

3.4.2 Основні функції смарт-контракту

Функції у смарт-контрактах – це визначені частини коду, які виконують певну логіку або операцію та доступні для виклику користувачем або іншим контрактом [36]. Як було зазначено вище, на основі інтерфейсу створюється контракт, який реалізує описані функції. Інші контракти можуть використовувати інтерфейс для виклику функцій, не знаючи деталей реалізації. У лістингу 3.2 показано основні функції смарт-контракту за допомогою інтерфейсу.

Лістинг 3.2 – Імпорт функцій базового класу та їх опис через інтрефейс

```
export interface DistributedStorage Interface extends
utils.Interface {
  functions: {
    "approve(address,uint256)": FunctionFragment;
    "balanceOf(address)": FunctionFragment;
    "burn(uint256)": FunctionFragment;
    "currentCounter()": FunctionFragment;
    "freeMint(address,string)": FunctionFragment;
    "owner()": FunctionFragment;
    "ownerOf(uint256)": FunctionFragment;
    "symbol()": FunctionFragment;
    "renounceOwnership()": FunctionFragment;
    "transferFrom(address,address,uint256)": FunctionFragment;
    "transferOwnership(address)": FunctionFragment;
  };
}
```

Перелік цих функцій:

- approve() дозволяє власнику токена надати дозвіл іншій адресі керувати певним токеном. Оскільки ця функція є зовнішньою, її використання впливає на стан контракту, додаючи відповідний запис до реєстру дозволів. В результаті вона оновлює таблицю дозволів для вказаного точена;

- balanceOf() повертає баланс токенів, що належать вказаній адресі. Це зовнішня функція, яка не змінює стан контракту, а лише зчитує дані з таблиці

балансу. Результатом виконання функції є поточна кількість токенів на зазначеній адресі;

- burn() використовується для знищення токенів. Вона змінює стан контракту, видаляючи токен з обігу. Після виконання функції кількість токенів у системі зменшується, а стан відповідної таблиці оновлюється;

- currentCounter() повертає значення лічильника токенів, який відображає загальну кількість випущених токенів. Це зовнішня функція, яка виконує лише зчитування даних і не змінює стан контракту;

- freeMint() дозволяє безкоштовно створити новий токен для вказаної адреси та зберегти метадані токена. Ця функція змінює стан контракту, додаючи новий запис до таблиці токенів, а також збільшує значення лічильника;

- owner() повертає адресу власника контракту. Як і попередня функція, вона виконує лише зчитування і не змінює стан контракту;

- ownerOf() повертає адресу власника токена за його унікальним ідентифікатором. Ця функція є зовнішньою і лише отримує дані про власника з таблиці токенів;

- renounceOwnership() дозволяє власнику контракту відмовитися від прав власності. Це змінює стан контракту, оновлюючи значення змінною власника на порожнє значення;

- transferFrom() здійснює передачу токена між двома адресами без додаткової перевірки. Виконання функції змінює стан контракту, оновлюючи записи про власників у таблиці токенів;

- transferOwnership() дозволяє передати права власності на контракт новій адресі. Вона змінює стан контракту, оновлюючи змінну власника.

3.4.3 Основні події смарт-контракту

Події у смарт-контрактах – це механізм реєстрації змін стану контракту. Події не зберігають дані в самому смарт-контракті, але

записуються у спеціальні лог транзакції, які зберігається у блокчейні [37]. Завдяки цьому події ефективні для моніторингу змін без додаткових витрат на зберігання. Вони виконують роль тригерів і використовуються для фіксації важливих дій, таких як переказ токенів, зміна власника чи виклик певної функції. У лістингу 3.3 продемонстровано основні події контракту.

Лістинг 3.3 – Імпорт подій та їх опис через інтрефейс

```
export interface DistributedStorage Interface extends
utils.Interface {
  events:{
    "Approval(address,address,uint256)": EventFragment;
    "ApprovalForAll(address,address,bool)": EventFragment;
    "BatchMetadataUpdate(uint256,uint256)": EventFragment;
    "MetadataUpdate(uint256)": EventFragment;
    "OwnershipTransferred(address,address)": EventFragment;
    "Transfer(address,address,uint256)": EventFragment;
  };}
```

Деякі функції та події в смарт-контрактах мають схожий механізм визначення, проте їхній спосіб використання відрізняється. Через це немає сенсу детально пояснювати принципи дії таких подій, як Transfer(), Approval() та OwnershipTransferred().

Стосовно інших подій, то BatchMetadataUpdate() використовується для реєстрації змін у метаданих групи токенів. Вона дозволяє масштабно оновлювати метадані без необхідності окремої обробки кожного токена.

Подія MetadataUpdate() викликається при зміні метаданих окремого токена. Це корисно для оновлення інформації про певний актив, наприклад, зображення або опису.

3.4.4 Функція конструктор

Функція-конструктор є спеціальною функцією в програмуванні смарт-контрактів, яка виконується один раз – під час створення екземпляра

контракту. Вона служить для початкової ініціалізації контракту, налаштування його стану та визначення важливих параметрів, таких як змінні, ролі або дозволи [38].

У Solidity конструктор є важливою складовою, яка дозволяє встановлювати значення за замовчуванням, створювати необхідні структури даних або виконувати інші дії, які визначають початкову логіку контракту (лістинг 3.4).

Лістинг 3.4 – Ініціалізація фабрики контракту через конструктор

```
export class DistributedStorage__factory extends
ContractFactory{
constructor(...args: DistributedStorage ConstructorParams) {
    if (isSuperArgs(args)) {
      super(...args);
    } else
   {
      super(_abi, _bytecode, args[0]);
    }}
}
```

В проекті функція-конструктор фабрики DistributedStorage__factory.ts відіграє ключову роль у налаштуванні механізму для створення і управління екземплярами смарт-контрактів. Фабрика контрактів – це інструмент, що дозволяє розгортати нові екземпляри контрактів, підключатися до вже існуючих і взаємодіяти з ними через визначений інтерфейс.

Конструктор приймає аргументи у фабриці, які можуть включати підписувача або повний набір параметрів, необхідних для базового класу ContractFactory. Завдяки умовній перевірці визначається, який підхід до ініціалізації використовувати: через ABI, байткод і підписувача, або через усі необхідні параметри для розширеного налаштування. Це забезпечує гнучкість у використанні фабрики та робить її універсальним інструментом для управління контрактами.

Функція-конструктор фабрики також налаштовує ключові методи, такі

як розгортання контрактів, генерація транзакцій для їх створення, підключення до існуючих екземплярів за адресою, та взаємодія через підписувача. Усі ці методи ґрунтуються на коректній роботі конструктора, який гарантує, що фабрика готова до повноцінної роботи з контрактом.

3.4.5 Функція розгортання контракту та створення NFT

Два файли, deploy.ts та mint.ts містять скрипти для взаємодії з контрактами на Ethereum за допомогою Hardhat.

Файл deploy.ts використовується для розгортання смарт-контракту на блокчейні. Спочатку у файл імпортуються необхідні бібліотеки, зокрема Hardhat, і ініціалізується з'єднання з мережами через hre (Hardhat Runtime Environment). Далі за допомогою методу getContractFactory() скрипт отримує фабрику для створення екземпляра контракту DistributedStorage і викликає метод deploy(), щоб розгорнути цей контракт в мережі Ethereum.

Після успішного розгортання скрипт виводить в консоль адресу контракту та власника контракту. У лістингу 3.5 продемонстровано роботу цього методу.

Лістинг 3.5 – Робота функції deploy()

```
const DistributedStorage = await
hre.ethers.getContractFactory("DistributedStorage");
const logo = await DistributedStorage.deploy();
await logo.deployed();
console.log("DistributedStorage deployed to:", logo.address);
console.log("owner", await logo.owner());
```

Файл mint.ts використовується для створення NFT на основі файлів зображень, які зберігаються у NFT.Storage. NFT.Storage – це бібліотека на базі IPFS і Filecoin для зберігання даних, яка використовується для завантаження файлів і метаданих у систему.

Також у цьому файлі імпортується пакет p-queue для обмеження

паралельних запитів при обробці великих обсягів даних. Функція uploadFile() відповідає за завантаження файлів у NFT.Storage. Вона зберігає зображення і його метадані та повертає метадані, зокрема URL для доступу до файлів.

Функція mintNft() обробляє файли, що зберігаються в папці assets, завантажує їх на NFT.Storage, а потім створює NFT за допомогою функції safeMint() контракту DistributedStorage.

Після виконання транзакції на блокчейні виводиться хеш блоку, в якому була підтверджена транзакція. Далі у лістингу 3.6 приведено основну частину коду із файлу mint.ts.

Лістинг 3.6 – Робота функції mint()

```
const file = readFileSync(filePath)
const metaData = await uploadFile({
  file: new File([file.buffer], name, {
    type: "image/png",}),
  name,
  description,});
const mintTx = await logo.safeMint(OWNER, metaData?.url);
const tx = await mintTx.wait();
```

3.5 Загальна взаємодія компонентів

Процес створення NFT включає взаємодію між інтерфейсом сайту, сервером та зовнішнім сервісом NFT.Storage.

Файл AddItemModal.tsx відповідає за інтерфейс користувача, який дозволяє створювати нові NFT. У цьому компоненті реалізовано форму, де користувач може ввести назву NFT, його опис і завантажити зображення. Після заповнення форми і натискання кнопки «Add», відбувається виклик функції onSubmit(). Ця функція упаковує введені користувачем дані та надсилає їх у вигляді POST-запиту на сервер. Запит виконується за допомогою API, і у ньому передаються дані у форматі JSON, включаючи файл зображення.

На сервері запит обробляється у файлі index.ts що знаходиться у

директорії серверу. В цьому файлі дані запиту приймаються за допомогою req.body, а потім перевіряється наявність усіх необхідних полів: name, description та file. Якщо дані валідні, сервер використовує бібліотеку NFT.Storage для створення метаданих. Спочатку зображення конвертується у набір байтів за допомогою методу toBuffer(). Потім метадані зберігаються за допомогою бібліотеки NFT.Storage. Код, що оброблює метаданні показано у лістингу 3.7.

Лістинг 3.7 – Завантаження метаданих через NFT.Storage

```
const metadata = await client.store({
  name: name.value,
  description: description.value,
  image: file ? new File([await file.toBuffer()], name.value, {
type: file.mimetype }) : null,});
```

Цей код зберігає файл у IPFS, повертаючи URL метаданих та IPNFT (унікальний ідентифікатор). Ці дані сервер надсилає у відповідь користувачу.

Після того, як метадані NFT успішно створені та збережені, сервер може ініціювати взаємодію з контрактом для додавання нового NFT у блокчейн. Функція mint(), яка відповідає за це, викликається через Ethers.js. У вже попередньо описаному файлі DistributedStorage.sol знаходиться смарт-контракт, де викликається функція mint(), яка приймає URL метаданих і створює новий NFT.

Ця функція додає новий токен у блокчейн, прив’язуючи до нього унікальний ідентифікатор і URL метаданих [39]. Після успішного виконання транзакції сервер повертає користувачу ID створеного токена та підтвердження про успішність операції.

Останній етап, за який відповідає компонент Nft.tsx, – це відображення створених NFT на сайті. Цей компонент отримує ID токена та URL метаданих, звертається до IPFS, щоб отримати зображення та іншу інформацію, і показує її користувачу.

Отже, запропоновано модель і алгоритм реалізації децентралізованого застосунку із використанням таких додаткових технологій, як NFT.Storage для зберігання метаданих NFT, а також OpenZeppelin для безпечної та ефективної реалізації смарт-контрактів із дотриманням стандарту ERC-721. Реалізована модель передбачає чітку інтеграцію між інтерфейсом користувача, блокчейн-мережею та зовнішніми сервісами для забезпечення стабільної роботи та безпеки даних.

Розроблений децентралізований застосунок має чітку структуру, що дозволяє ефективно інтегрувати роботу з блокчейн-мережами, реалізовувати логіку управління NFT та забезпечувати взаємодію між інтерфейсом користувача та сервером. Реалізований смарт-контракт відповідає стандарту ERC-721 та забезпечує базову функціональність створення, зберігання, передачі та знищення токенів, що робить його доступним і безпечним у використанні. Наступним етапом є демонстрація роботи додатку та розгортання контракту в мережі блокчейн.

4 ТЕСТУВАННЯ РОБОТИ ДЕЦЕНТРАЛІЗОВАНОГО ЗАСТОСУНКУ ТА РОЗГОРТАННЯ СМАРТ-КОНТРАКТУ

Невід'ємним етапом розробки децентралізованого додатку є тестування, яке необхідне для перевірки коректного функціонування компонентів сайту, безпеки системи та відповідності початковим вимогам. Тестування є частиною процесу розробки, оскільки дозволяє виявити помилки на ранніх етапах та гарантувати стабільність взаємодії різних платформ і бібліотек. У цьому розділі буде продемонстровано результат розробки та зроблено тести взаємодії між користувачем та застосунком. Також буде розгорнуто смарт-контракт, який є обов'язковим для забезпечення функціонування блокчейн-компонентів програми.

До початку тестування застосунку необхідно створити новий гаманець, з якого будуть проходити різні взаємодії з додатком та контрактом. Адреса створеного гаманця – 0x49Bc41F026f9832475aAEe8c1D5CE966694d9AeD, її можна побачити в скороченому вигляді на рисунку 4.1. для взаємодії гаманця та додатку в якості інтерфейсу було використано Metamask.

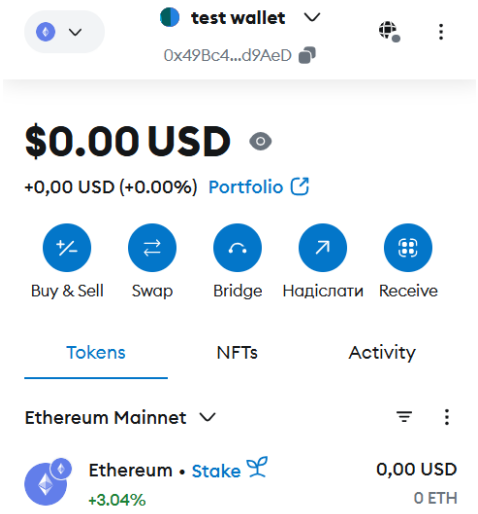

Рисунок 4.1 – Новостворена адреса гаманця за допомогою Metamask

Далі, запустивши застосунок на локальному хостингу, можна побачити інтерфейс головної сторінки сайту та ознайомитися з його основними функціями. Головна сторінка додатку (рисунок 4.2) є центральним місцем для взаємодії користувача з застосунком. На ній розташована кнопка для підключення гаманця – «Connect wallet» та кнопка додавання NFT – «Add NFT». Без підключеного гаманця кнопка додавання NFT є неактивною. У верхній частині сторінки знаходяться елементи інтерфейсу, що відображають поточну мережу, у якій працює додаток, адресу підключеного гаманця та його баланс. Додатково в лівому верхньому куті є індикатор, що відображає підключення гаманця до сайту. Цей індикатор має три стани: помаранчевий інформує, що гаманець ще не підключений, зелений – підключення було успішне та червоний – сталося помилка при підключенні.

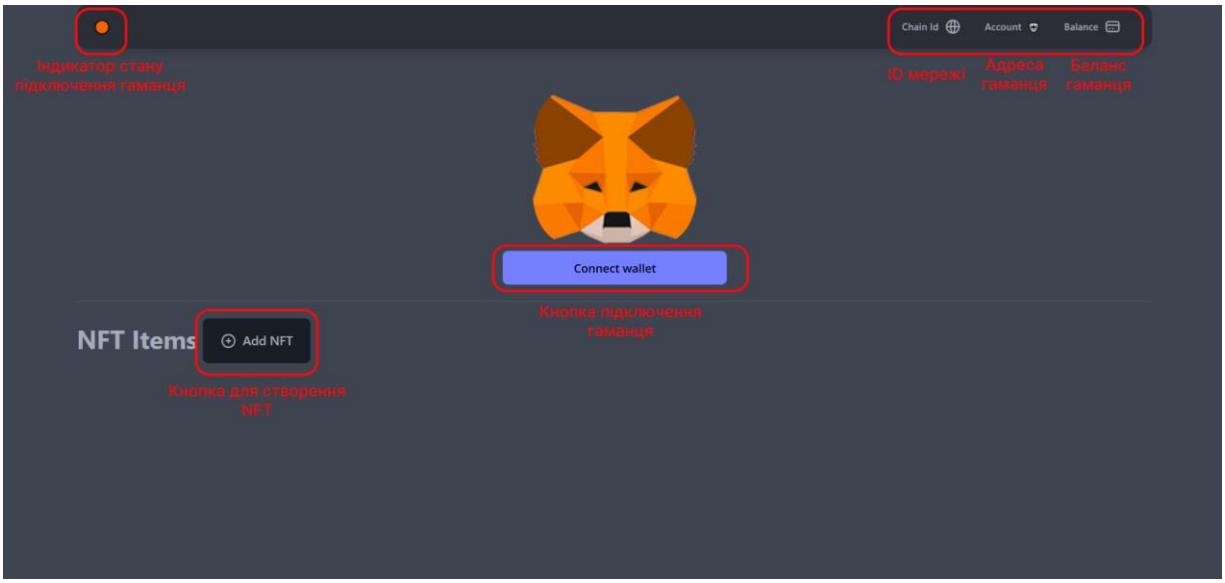

Рисунок 4.2 – Інтерфейс головної сторінки сайту

Спочатку користувач має підключити свій гаманець через розширення Metamask. Для цього необхідно натиснути кнопку підключення гаманця, після чого відкриється спливаюче вікно Metamask. Користувач повинен надати дозвіл на взаємодію між сайтом і гаманцем. Процес підключення гаманця зображено на рисунку 4.3.

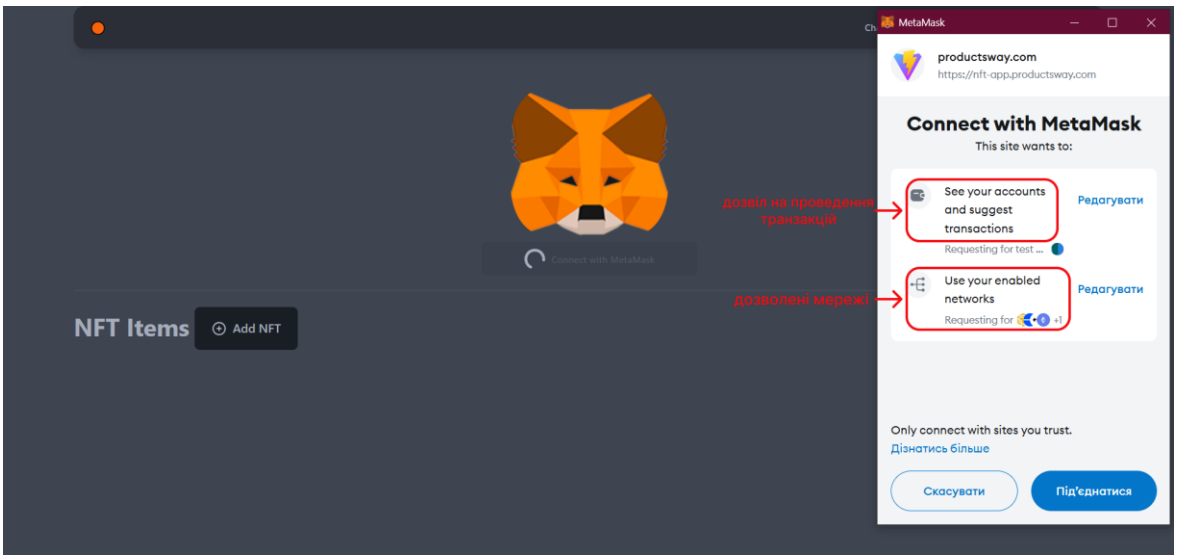

Рисунок 4.3 – Процес підключення гаманця до сайту через Metamask

Після підключення, гаманець надає сайту доступ до інформації про адресу підключеного гаманця, дозволяє ініціювати транзакції та працювати у дозволених мережах. Про вдале підключення свідчить зміна індикатору стану підключення з помаранчевого на зелений.

Також користувач повинен підписати запит від застосунку, детальніше про це на рисунку 4.4. Цей підпис підтверджує авторизацію користувача та забезпечує безпечну передачу даних.

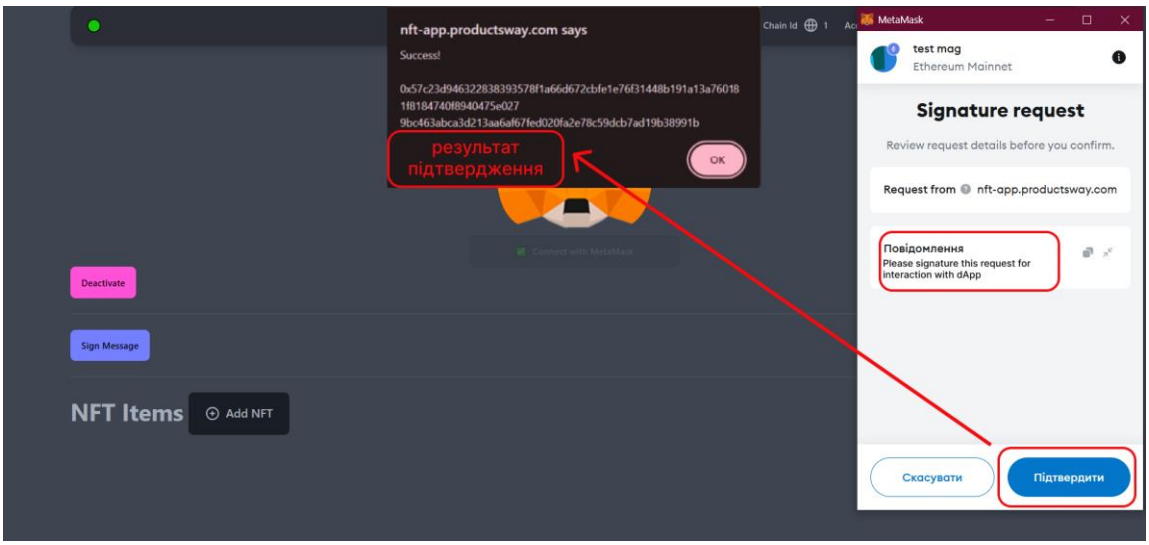

Рисунок 4.4 – Підпис запиту від додатку та результат підпису

Підпис потрібен для перевірки автентичності користувача та підтвердження, що взаємодія здійснюється від імені власника гаманця. Результат підпису можна побачити у вигляді повідомлення про успішне з'єднання.

Після підключення гаманця користувач переходить до завантаження зображення на сайт, яке має стати NFT. Для цього необхідно обрати файл на локальному пристрої, вказати його назву та додати опис. Для прикладу (рисунок 4.5) було обрано файл nure.png, поле Name було заповнено як «NureLogo», а поле Description – «logotype». Після заповнення полів користувач натискає кнопку «Add». У цей момент на сервер надсилається запит із зазначеними даними, а зображення завантажується для подальшої обробки.

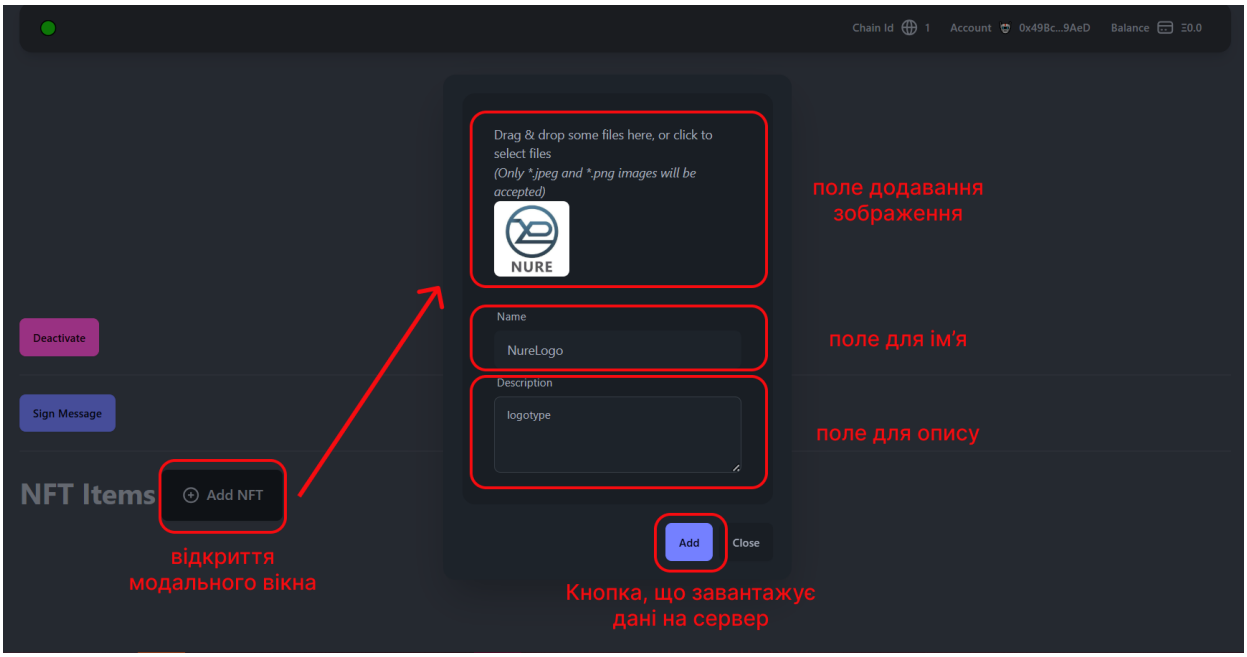

Рисунок 4.5 – Завантаження файлу-зображення на сервер

На цьому етапі виконується транзакція (рисунок 4.6), яка включає передачу метаданих на сервер та їх збереження у IPFS з використанням бібліотеки NFT.Storage.

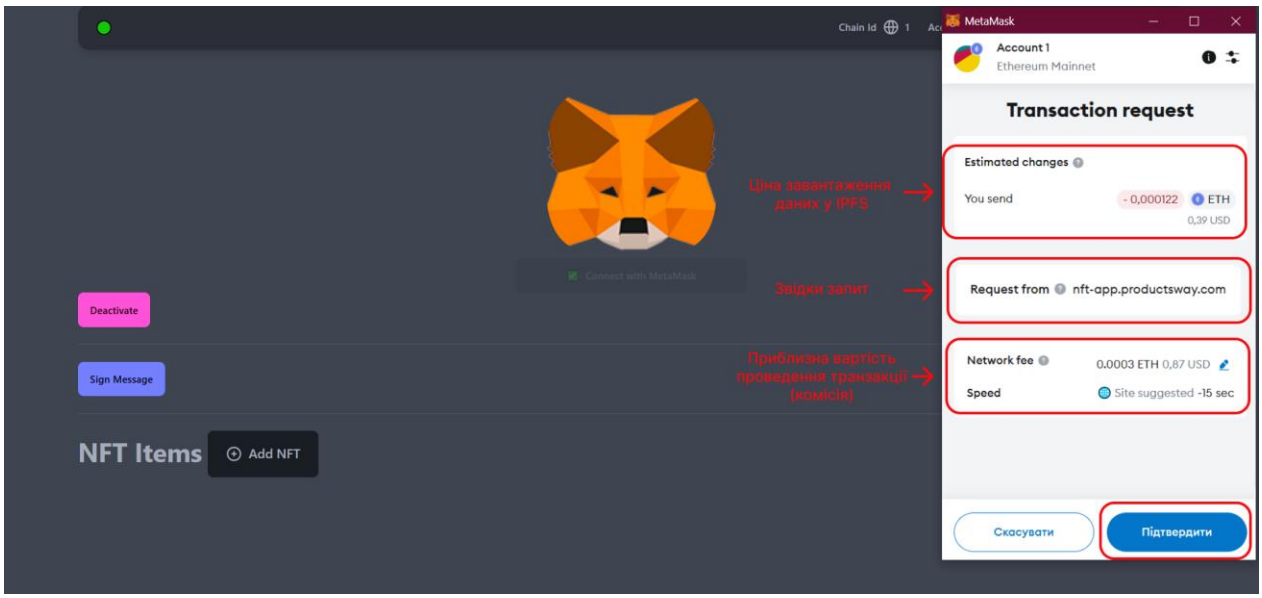

Рисунок 4.6 – Підписання транзакції, що завантажує данні у IPFS через бібліотеку NFT.Storage

Результатом завантаження є створення JSON-файлу, який містить метадані NFT, включаючи посилання на зображення, його назву та опис. Цей JSON-файл можна переглянути у додатку разом із відображенням введених даних. На рисунку 4.7 можна побачити вміст JSON-файлу, що підтверджує успішне виконання транзакції.

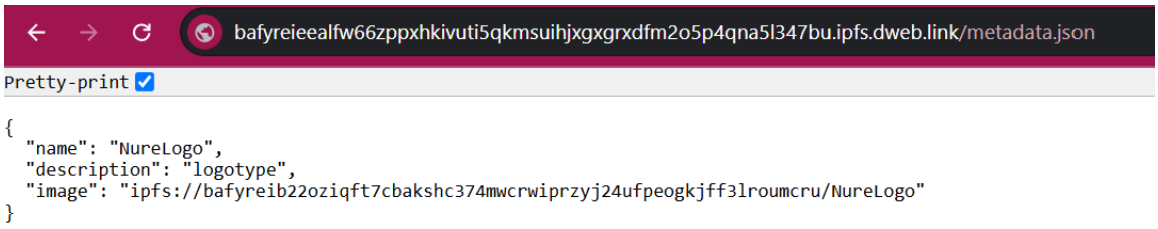

Рисунок 4.7 – Створений JSON-файл із метаданими NFT

Наступний етап передбачає розгортання смарт-контракту. Цей смарт-контракт слугує своєрідним «сховищем» для NFT. Додані NFT формують колекцію, створену за допомогою смарт-контракту та доступ до цієї колекції забезпечується через адресу розгорнутого смарт-контракту.

Додатково для цього було завантажено ще одне зображення із назвою «NureLogoUA». Для того, щоб ініціювати розгортання контракту, користувач має натиснути на кнопку «Deploy». Після цього автоматично генерується смарт-контракт, який містить усі завантажені зображення у вигляді NFT. Перед розгортанням контракту користувач має підтвердити транзакцію через MetaMask. Це включає перевірку вартості газу та підпис транзакції для її відправлення в мережу Ethereum. Вся інформація наявна на рисунку 4.8.

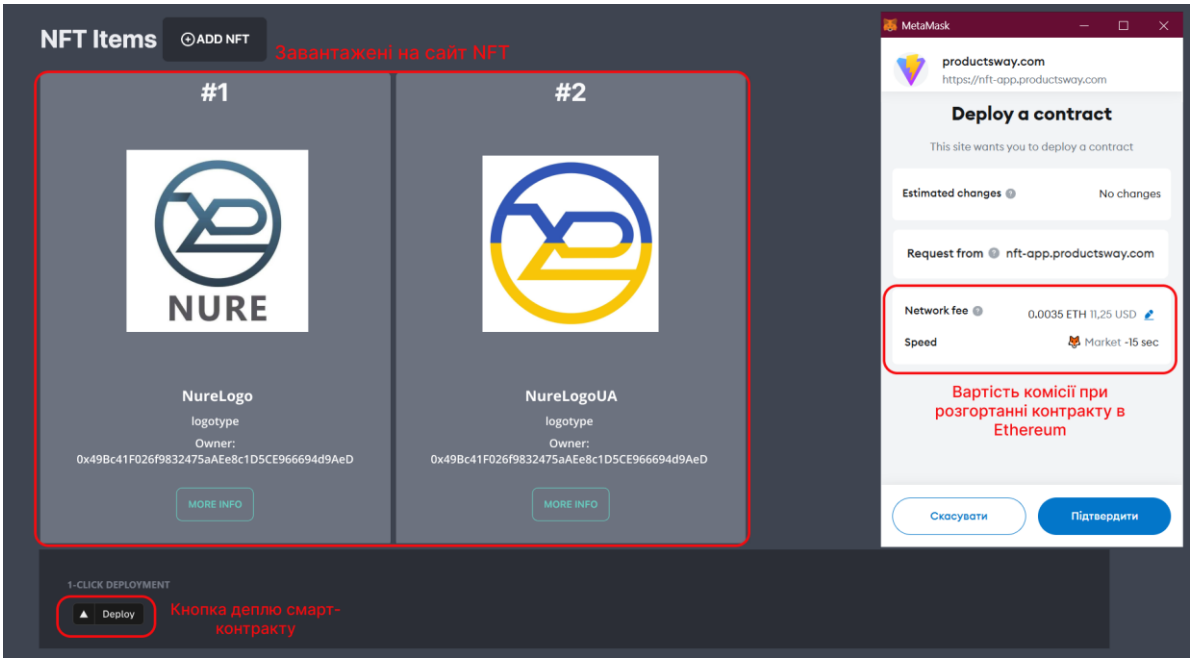

Рисунок 4.8 – Розгортання смарт-контракту

Одразу після успішного розгортання смарт-контракту, автоматично ініціюється транзакція для мінту всіх доступних NFT, що також потребує підтвердження від користувача. Розгорнутий контракт має адресу у блокчейні Ethereum, яку можна переглянути на блокчейн-експлорері Etherscan. Адреса, за якою можна знайти розгорнутий смарт-контракт з назвою DistributedStorage: 0x76570f847aea3b224ca4a1b864e6908ca6c5029b. Цей інструмент дозволяє перевірити статус контракту, його транзакцію розгортання та всі пов'язані дані. На рисунку 4.9 продемонстровано інтерфейс сайту Etherscan та всю доступну інформацію про контракт.

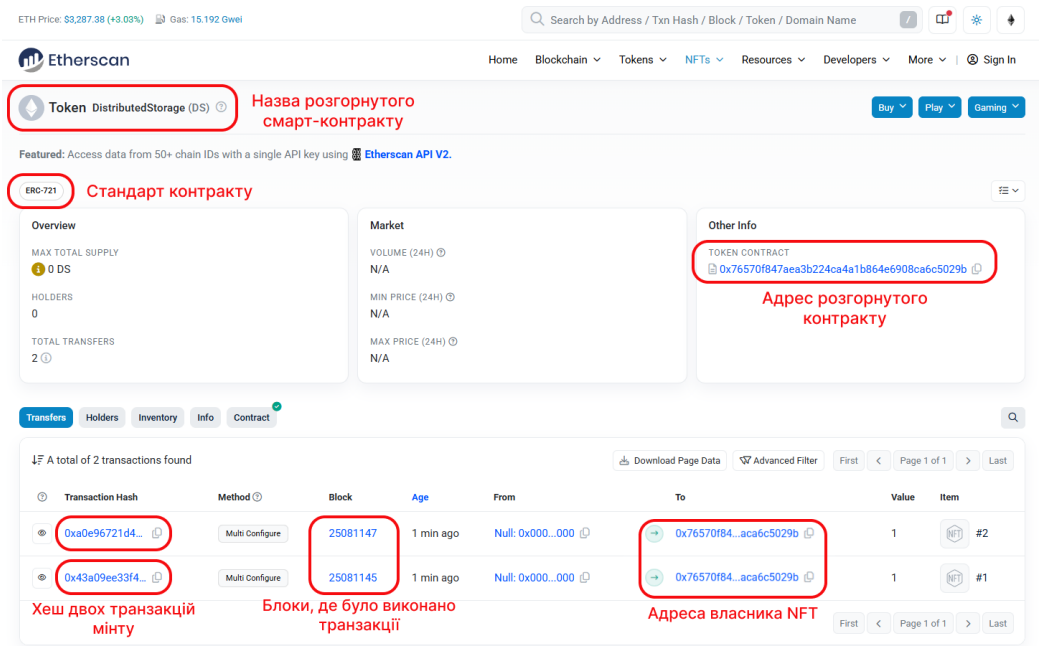

Рисунок 4.9 – Сайт Etherscan та інформація про розгорнутий смарт-контракт

За допомогою сайту Etherscan є можливість інтерактивної взаємодії зі смарт-контрактом. Користувач може перевіряти дані про завантажені NFT і метадані, а також взаємодіяти з ними через функції контракту. У розділі «Code» користувач може переглянути розгорнутий код смарт-контракту. У розділах «Read Contract» та «Write Contract» (рисунок 4.10) можна побачити реалізовані функції смарт-контракту та виконати будь-які доступні дії з ним.

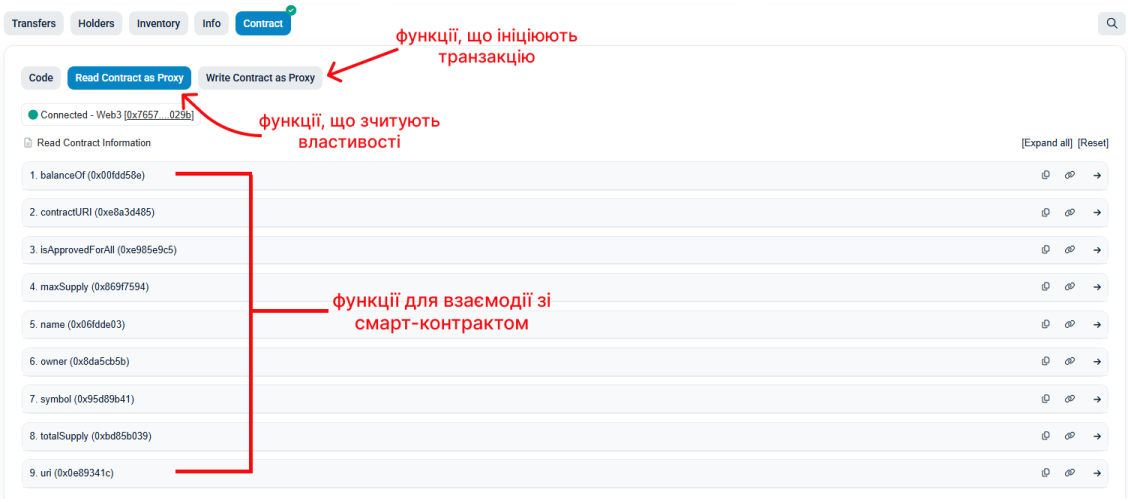

Рисунок 4.10 – Перегляд функцій смарт-контракту через Etherscan

Завантажені зображення та дані про них, додані до блокчейну Ethereum, можна наочно переглянути у розділі «Inventory» (рисунок 4.11). Кожна NFT тепер має унікальний ідентифікатор та зберігає інформацію, надану через застосунок. Таким чином, NFT та її метадані зберігаються у децентралізованій мережі та можуть бути доступними не лише через застосунок або сайт Etherscan, а також у будь-якому децентралізованому додатку, який підтримує інтеграцію зі стандартом ERC-721 та мережею Ethereum.

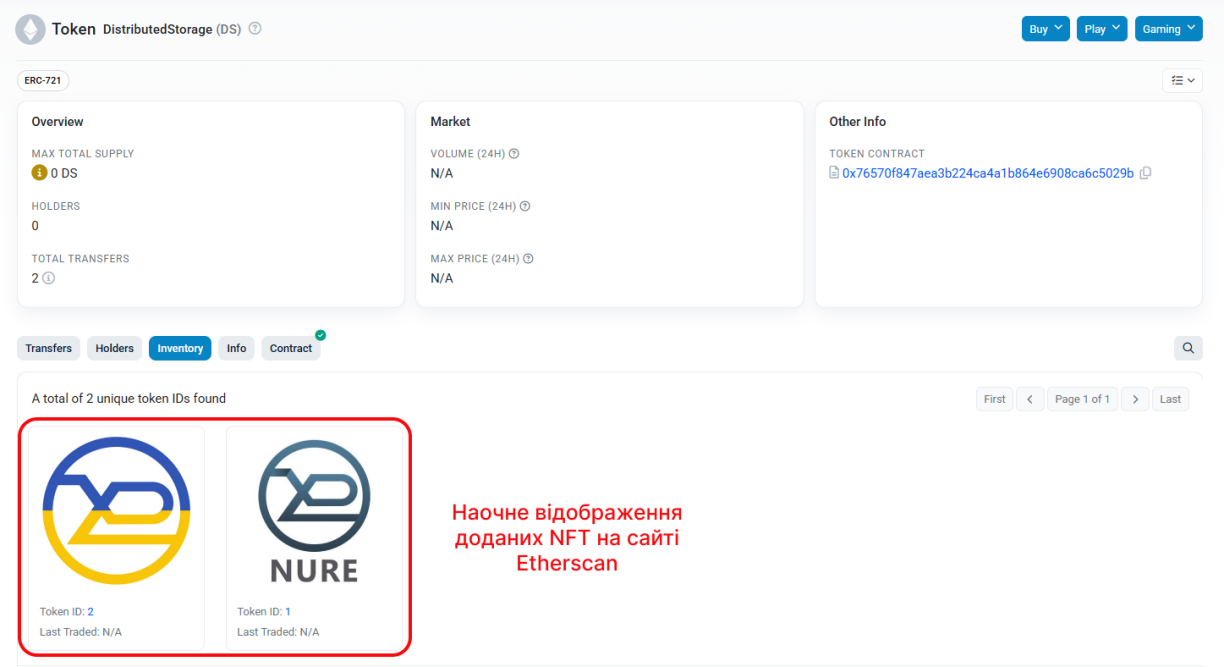

Рисунок 4.11 – Відображення створених NFT на сторінці контракту токена

Прикладом подібного децентралізованого додатку, що також підтримує мережу Ethereum і роботу зі стандартом ERC-721, є OpenSea (рисунок 4.12). Ця платформа може відображати NFT, які зберігаються у децентралізованій мережі, та надає можливість безпосередньо взаємодіяти з функціями смарт-контракту NFT [40]. Той факт, що розгорнутий контракт і створені NFT вдало відображаються у додатку OpenSea, є підтвердженням правильного розгортання смарт-контракту та коректної роботи реалізованого застосунку.

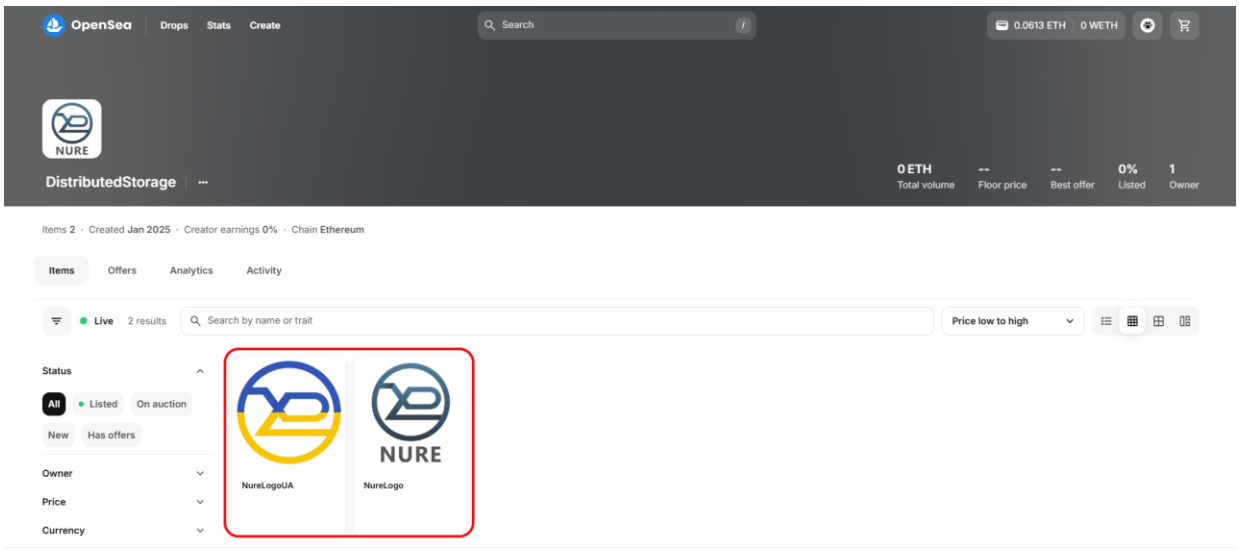

Рисунок 4.12 – Відображення створених NFT у децентралізованому додатку OpenSea

Проведене тестування та демонстрація результатів підтвердили коректність роботи децентралізованого застосунку та смарт-контракту. Було створено новий гаманець для взаємодії з додатком, перевірено процес підключення та авторизації через Metamask, а також виконано завантаження метаданих NFT у децентралізовану мережу IPFS із використанням бібліотеки NFT.Storage.

Розгортання смарт-контракту у мережі Ethereum пройшло без помилок, про що свідчать дані, доступні на Etherscan, зокрема статус транзакції, розгорнутий код контракту та його функції. Взаємодія з контрактом через інтерфейс Etherscan підтвердила доступність і працездатність функцій. Завантажені NFT були успішно додані до блокчейну та відображені на платформі OpenSea, що свідчить про відповідність контракту стандарту ERC-721.

Результати тестування демонструють, що децентралізований застосунок забезпечує коректну взаємодію між користувачем, смарт-контрактом та блокчейном. Це підтверджує готовність додатку до реального використання та його відповідність початковим вимогам.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проаналізовано сучасні загрози інформаційній безпеці в цифровому середовищі, включаючи ризики крадіжки даних, залежність від централізованих систем і можливість втрати інформації через зловмисні дії чи технічні несправності. На основі аналізу обґрунтовано необхідність використання децентралізованих технологій, як альтернативного варіанту забезпечення прозорості та безпеки даних.

Досліджено концепцію децентралізації у блокчейн-технологіях та її значення для сучасних інформаційних систем. Визначено, що децентралізовані системи мають ключові переваги, такі як стійкість до зовнішніх атак, виключення залежності від єдиного вузла та прозорість процесів.

Виявлено ключові переваги блокчейну Ethereum, зокрема незмінність, масштабованість і стабільність роботи, що робить його оптимальним середовищем для розробки децентралізованих застосунків.

Додатково було досліджено стандарт ERC-721, який забезпечує створення унікальних і довговічних цифрових активів. Вивчено його функціональні можливості та переваги для зберігання даних.

Розроблено модель децентралізованого зберігання даних із використанням смарт-контракту в мережі Ethereum, що відповідає вимогам прозорості, безпеки та ефективності роботи з цифровими активами. На базі моделі розроблено децентралізований застосунок із використанням сучасних технологій, включаючи інтеграцію з NFT.Storage для зберігання метаданих і Ethereum для взаємодії зі смарт-контрактами.

Було протестовано створену систему, підтверджено її працездатність. Проведене тестування показало ефективність взаємодії між користувачем, смарт-контрактом та блокчейн-мережею. Зокрема, успішно реалізовано створення, зберігання та передачу NFT-токенів, а також забезпечено

коректність інтеграції з інструментами на кшталт Etherscan.

Проєкт відкриває перспективи для масштабування до інших блокчейн-мереж і впровадження нових функцій, таких як динамічне оновлення метаданих і автоматизація управління NFT через інтерфейс.

Важливим подальшим напрямом розвитку є реалізація безпосереднього управління NFT через інтерфейс додатку, що дозволить користувачам створювати, передавати та знищувати токени без необхідності прямої взаємодії зі смарт-контрактом або сторонніми інструментами. Це значно спростить роботу для користувачів і зробить застосунок більш зручним та інтуїтивним.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Anderson, R. Security Engineering: A Guide to Building Dependable Distributed Systems. 3rd ed. – Indianapolis: Wiley, 2020. – 1232 p. – ISBN 9781119642787.

2. Tapscott, D. & Tapscott, A. Blockchain Revolution: How the Technology Behind Bitcoin and Other Cryptocurrencies is Changing the World. – Portfolio, 2018. – 432 p. – ISBN 9781101980149.

3. Filecoin Documentation. A Guide to Filecoin and IPFS. Filecoin Docs. [Електронний ресурс] – Filecoin – Режим доступу: www / URL: https://docs.filecoin.io/basics/what-is-filecoin (дата звернення: 01.10.2024)

4. Stallings, W. Network Security Essentials: Applications and Standards. 6th ed. – Upper Saddle River: Pearson, 2017. – 464 p. – ISBN 9780134527338.

5. Anderson, R. Security Engineering: A Guide to Building Dependable Distributed Systems /– Wiley, 2021. – 1200 p. – ISBN 9781119642787.

6. Stallings, W. Cryptography and Network Security: Principles and Practice. 7th ed. – Upper Saddle River: Pearson, 2017. – 768 p. – ISBN 9780134444284.

7. Northcutt, S., Novak, J. Network Intrusion Detection. 3rd ed. – Indianapolis: Sams Publishing, 2002. – 512 p. – ISBN 9780735712652.

8. Гуз А.М., Довгань О.Д., Марущак А.І. та ін. Організація захисту інформації з обмеженим доступом : підручник / за ред. Є.Д. Скулиша. – К.: Наук.-вид. відділ НА СБ України, 2011. – 376 с.

9. Narayanan A., Bonneau J., Felten E. Bitcoin and Cryptocurrency Technologies / A. Narayanan, J. Bonneau, E. Felten. – Princeton University Press, 2016. – 336 p. – ISBN 9780691171692.

10. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System / Satoshi Nakamoto. – bitcoin.org, 2008. – 9 p.

11. Mougayar, W., Buterin, V. The Business Blockchain: Promise, Practice,

and Application of the Next Internet Technology, 1st ed. – Hoboken, NJ: Wiley, 2016. – 208 p. – ISBN 9781119300311.

12. Ovezik, C., Karakostas, D., Kiayias, A. SoK: A Stratified Approach to Blockchain Decentralization. – University of Edinburgh, IOG, 2022 – 43 p.

13. Що таке блокчейн і як він працює? [Електронний ресурс] – Binance Academy, 2023. – Режим доступу: www / URL: https://academy.binance.com/uk-UA/articles/what-is-blockchain-and-how-does-it-work.

14. Блокчейн: основні принципи та сфери застосування [Електронний ресурс] – Вгору, 2024. – Режим доступу: www / URL: https://vgoru.org/pererva-na-kavu/blokchejn-osnovni-principi-ta-sferi-zastosuvannya.

15. Bitcoin vs. Ethereum: What's the Difference? [Електронний ресурс] – Investopedia, 2023. – Режим доступу: www / URL: https://www.investopedia.com/articles/investing/031416/bitcoin-vs-ethereum-driven-different-purposes.asp.

16. What is Ethereum? Ethereum Explained [Електронний ресурс].– Режим доступу: www / URL: https://www.coindesk.com/learn/what-is-ethereum.

17. Buterin, V. Merkling in Ethereum [Електронний ресурс] / Vitalik Buterin. – blog.ethereum.org, 2015. – Режим доступу: www / URL: https://blog.ethereum.org/2015/11/15/merkling-in-ethereum.

18. Wood, G. Ethereum Virtual Machine Architecture [Електронний ресурс] / Gavin Wood. – Ethereum.org, 2014. – Режим доступу: www / URL: https://ethereum.github.io/yellowpaper/paper.pdf.

19. Rettig, L. Inside the Ethereum Virtual Machine [Електронний ресурс] / Lane Rettig. – Medium, 2018. – Режим доступу: www / URL: https://medium.com/@lrettig/inside-the-ethereum-virtual-machine-dc6efbde8b69

20. Buterin, V. A Next-Generation Smart Contract and Decentralized Application Platform [Електронний ресурс] / Vitalik Buterin. – Ethereum.org, 2015. – Режим доступу: www / URL: https://ethereum.org/en/whitepaper/

21. Mukhopadhyay M. Ethereum Smart Contract Development: Build blockchain-based decentralized applications using Solidity. – Packt Publishing,

2018. – 288 p. – ISBN 9781788473040.

22. Raval S. Decentralized Applications: Harnessing Bitcoin's Blockchain Technology – O'Reilly Media, 2017. – 116 p. – ISBN 9781491924549.

23. ERC-721 Standard Documentation. [Електронний ресурс]. – Режим доступу: www / URL: https://ethereum.org/en/developers/docs/standards/tokens/erc-721/.

24. ERC-20 Token Standard Documentation. [Електронний ресурс]. – Режим доступу: www / URL: https://ethereum.org/en/developers/docs/standards/tokens/erc-20/.

25. ERC-721: Non-Fungible Token Standard [Електронний ресурс]. – Режим доступу: www / URL: https://eips.ethereum.org/EIPS/eip-721.

26. Селіванов І.О. Особливості предметної області розробки децентралізованого зберігання даних і смарт-контракту в мережі Ethereum. – The 1st International scientific and practical conference «European congress of scientific discovery» (December 29-31, 2024). – Barca Academy Publishing, Madrid, Spain. 2024 – 266-270 с. – ISBN 9788415927303.

27. Chittoda J. Mastering Blockchain Programming with Solidity – Packt Publishing, 1st edition, 2019. – 488 p. – ISBN 9781839218262.

28. Solidity Compiler Documentation. [Електронний ресурс]. – Режим доступу: www / URL: https://docs.soliditylang.org.

29. Visual Studio Code Documentation [Електронний ресурс]. – Режим доступу: www / URL: https://code.visualstudio.com/docs.

30. Yakov F., Moiseev A. TypeScript Quickly – Manning Publications, 2020.–488 p. – ISBN 9781617295942.

31. Banks A. Learning React: Modern Patterns for Developing React Apps / A. Banks. – O'Reilly Media, 2nd edition, 2020. – 307 p. – ISBN 9781492051725.

32. OpenZeppelin Smart Contract Libraries. [Електронний ресурс]. – Режим доступу: www / URL: https://docs.openzeppelin.com/contracts.

33. NFT.Storage Documentation. [Електронний ресурс]. – Режим доступу: www / URL: https://app.nft.storage/v1/docs/intro.

34. MetaMask Documentation [Електронний ресурс]. – Режим доступу: https://docs.metamask.io/.

35. OpenZeppelin ERC-721 Documentation. [Електронний ресурс]. – Режим доступу: www / URL: https://docs.openzeppelin.com/contracts/5.x/erc721

36. Smart Contract Functions Explained. [Електронний ресурс]. – Режим доступу: www / URL: https://ethereum.org/en/developers/docs/smart-contracts/anatomy/#functions.

37. Event Usage in Smart Contracts. [Електронний ресурс]. – Режим доступу: www / URL: https://docs.soliditylang.org/en/latest/contracts.html#events

38. Constructors in Solidity. [Електронний ресурс]. – Режим доступу : www / URL: https://docs.soliditylang.org/en/latest/contracts.html#constructors

39. NFT Minter Tutorial. [Електронний ресурс]. – Режим доступу : www / URL: https://ethereum.org/en/developers/tutorials/nft-minter/

40. Arsath Natheem S. NFT BOOK FOR BEGINNERS: Welcome to NFTverse: Deepdive into NFT Ecosystem, Metaverse, Decentralization, Web3, DeFi and Blockchain. – Kindle Edition, 2022.