

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет інформаційної безпеки та електронних комунікацій

(повне найменування інституту, назва факультету)

Кафедра інформаційної безпеки та наноелектроніки

(найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему Технології безпеки надання доступу до облікових записів у
інформаційних мережах

Виконав(ла): студент(ка) 6 курсу, групи БК-812м

Спеціальності 125 Кібербезпека

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Безпека інформаційних і комунікаційних

мереж

ГОЛОВКО І.О.

(ПРИЗВИЩЕ та ініціали)

Керівник КОЗИНА Г.Л.

(ПРИЗВИЩЕ та ініціали)

Рецензент МОРОЗ Г.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет інформаційної безпеки та електронних комунікацій
Кафедра інформаційної безпеки та наноелектроніки
Ступінь вищої освіти магістр
Спеціальність 125 Кібербезпека
(код і найменування)

Освітня програма (спеціалізація) Безпека інформаційних і комунікаційних мереж
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри ІБтаН

Андрій КОРОТУН

« » 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ГОЛОВКА Івана Олександровича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи): Технології безпеки надання доступу до облікових записів у інформаційних мережах
Security technologies for providing access to accounts in information networks

керівник проєкту (роботи) к.ф.м.н., доцент КОЗІНА Галина Леонідівна
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «28» листопада 2023 року №475

2. Строк подання студентом проєкту (роботи) 11.12.2023

3. Вихідні дані проєкту (роботи): дослідити вплив технології, що дозволяє відтворювати процес мутації даних в мережі блокчейн

4. Зміст розрахунково-пояснювальної записки (перелік питань, що належить розробити): огляд технології блокчейн, принципів її роботи та взаємодії з нею; розгляд шляхів досягнення мутабельності в блокчейн протоколах; порівняння показників протоколів з відтворення мутабельності та без відтворення; приклад смартконтракту, що відтворює мутабельність даних

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) презентація (14 слайдів)

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
Розділи 1-5	КОЗІНА Г.Л., к.ф.м.н., доцент ІБтаН		
Нормоконтроль	КОРОЛЬКОВ Р.Ю., к.т.н. доцент кафедри ІБтаН		

7. Дата видачі завдання: “04” вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1.	Аналіз літературних джерел за тематикою дослідження.	04.09.23 – 18.09.23	
2.	Огляд можливих методів імітації мутацій даних в блокчейні.	19.09.23 – 25.09.23	
3.	Аналіз та систематизація методів мутації.	26.09.23 – 12.10.23	
4.	Аналіз ризиків та загроз, пов’язаних з механізмами мутації.	13.10.23 – 23.10.23	
5.	Збір інформації про стан мереж.	24.10.23 – 27.10.23	
6.	Аналіз впливу наявності механізму мутації в протоколі блокчейн мережі на прикладі блокчейнів Ethereum та Flow.	30.10.23 – 6.11.23	
7.	Розробка смартконтракту, що дозволяє використовувати механізми мутацій даних.	7.11.23 – 10.11.23	
8.	Розробка застосунку, що реалізує механізм мутації даних на основі блокчейн технології.	13.11.23 – 17.11.23	
9.	Оформлення матеріалів магістерської роботи.	20.11.23 – 1.12.23	

Студент(ка)

(підпис) Іван ГОЛОВКО
(Ім’я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Галина КОЗІНА
(Ім’я ПРІЗВИЩЕ)

АНОТАЦІЯ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
91 с., 31 рис., 3 додатки, 22 джерела.

БЛОКЧЕЙН, ГАМАНЕЦЬ, МЕРЕЖА, МУТАЦІЯ ДАНИХ,
ПРОГРАМНИЙ АПАРАТ, РЕСУРС, СЦЕНАРІЙ, СМАРТКОНТРАКТ,
ТОКЕНИ, ТРАНЗАКЦІЇ, ФУНКЦІЯ, CADENCE, NODEJS

Технологія блокчейну широко застосовується в сучасному житті. Валютні операції, збереження даних у банківській сфері, документообіг, медицина, продаж художніх цінностей - це ті галузі, у яких широко використовується технологія блокчейн мереж. Так як в будь-якій сфері присутні допущення, помилки та надзвичайні ситуації, що можуть призвести до спотворення інформації, є необхідність передбачення механізму, що дозволить змінювати або імітувати зміни даних в блокчейн мережах. Відповідно необхідно дослідити вплив даної технології та методи її реалізації.

Об'єктом дослідження є технології мутабельності даних в блокчейні, що надають змінювати чи імітувати зміну даних в блокчейні.

Предметом роботи є аналіз реалізації механізму мутабельності даних на прикладі смартконтракту та застосунку, а також дослідження, як імітація мутабельності (можливості модифікації), може впливати на рівень безпеки блокчейну.

Метою роботи є дослідити технології, що дозволяють імітувати мутабельності даних в блокчейні на прикладі реалізації відповідного програмного забезпечення та можливостей протоколу блокчейну.

Задачі дослідження:

- аналіз впливу алгоритмів мутації даних на захищеність та швидкодії блокчейн мереж;
- аналіз шляхів реалізації алгоритмів, що дозволяють імітувати мутабельність даних на основі блокчейн мережі;
- розробка застосунку та смартконтракту, що дозволяють практичним шляхом дослідити алгоритми імітації мутабельності даних.

Наукова новизна одержаних результатів. Систематизація підходів до створення механізмів, що імітують мутабельність даних в блокчейні. Порівняння характеристик мереж, відносно наявності у них механізму мутабельності.

Практичне значення одержаних результатів. Розроблено застосунок, що здатний зберігати та імітувати зміну даних на основі блокчейну, та не залежить від наявності у блокчейну вбудованих механізмів мутації.

Апробація результатів: Головка І.О., Козіна Г.Л. Аналіз впливу мутабельності даних в блокчейні на захищеність // Тиждень науки: тези доповідей науково-практичної конференції, Запоріжжя, 24–28 квітня 2023 р.: тези доповідей науково-практичної конференції, Запоріжжя, 2023 р. С. 111-113.

Головка І.О. Дослідження можливості реалізації веб-застосунку для цифрового підпису документів з використанням технології blockchain / Головка І.О., Ребриков М.М., Козіна Г.Л. // Інформаційні технології: теорія і практика: V Всеукр. Інтернет-конф. здобувачів вищої освіти і молодих учених, 17-18 березня 2022 р.: тези доповідей. – Запоріжжя, 2022. – С. 126-128.

Головка І.А. Реализация платежной системы с использованием блокчейна / Головка И.А., Ребриков Н.Н., Козина Г.Л. / Математическое моделирование, оптимизация и информационные технологии (ММОТИ-2021): 7-я Международная конференция, Кишинев, 15-19 ноября 2021 г.: Материалы конференции / Evrica – Chisinau, 2021. – С.195-197.

ABSTRACT

Explanatory note to the master's thesis: 91 pages, 31 figures, 3 appendices, 22 sources.

BLOCKCHAIN, CADENCE, DATA MUTATION, FUNCTION, NETWORK, NODEJS, RESOURCE, SCENARIO, SMART CONTRACT, SOFTWARE, TOKENS, TRANSACTIONS, WALLET

Blockchain technology is widely used in modern life, encompassing currency transactions, data storage in the banking sector, document flow, medicine, and the sale of art. These are areas where blockchain technology networks are extensively employed. As errors and exceptional situations can occur in any field, leading to information distortion, there is a need to anticipate a mechanism that allows for changes or simulation of data changes in blockchain networks. Consequently, it is necessary to monitor the impact of this technology and methods of its implementation.

The research object is the technologies of data mutability in blockchain, allowing for the modification or simulation of data changes in the blockchain.

The subject of the research is the analysis of the implementation of the data mutability mechanism using a smart contract and application as an example, as well as the exploration of how the imitation of mutability (modification capabilities) can affect the security level of the blockchain.

The purpose of the work is to explore technologies that allow simulating data mutability in the blockchain through the implementation of relevant software and blockchain protocol capabilities.

Research tasks:

- analysis of the impact of data mutation algorithms on the security and performance of blockchain networks;
- analysis of implementation methods for algorithms that allow simulating data mutability based on blockchain networks;
- development of an application and smart contract that practically explore data mutability simulation algorithms.

Scientific novelty of the obtained results: Systematization of approaches to creating mechanisms that simulate data mutability in the blockchain. Comparison of network characteristics regarding the presence of mutability mechanisms.

Practical significance of the obtained results: A developed application capable of storing and simulating data changes based on the blockchain, independent of the built-in mutation mechanisms in the blockchain.

Results validation: Holovko I.O., Kozina G.L. Analysis of the Impact of Data Mutability in Blockchain on Security // Week of Science: Abstracts of Scientific-Practical Conference, Zaporizhzhia, April 24-28, 2023: Abstracts of Scientific-Practical Conference, Zaporizhzhia, 2023. P. 111-113.

Holovko I.O. Investigation of the Possibility of Implementing a Web Application for Digital Document Signing Using Blockchain Technology / Holovko I.O., Rebrivkov M.M., Kozina G.L. // Information Technologies: Theory and Practice: V All-Ukrainian Internet Conference of Higher Education Students and Young Scientists, March 17-18, 2022: Abstracts of Presentations. – Zaporizhzhia, 2022. – P. 126-128.

Holovko I.O. Implementation of a Payment System Using Blockchain / Holovko I.O., Rebrivkov N.N., Kozina G.L. / Mathematical Modeling, Optimization, and Information Technologies (MMOTI-2021): 7th International Conference, Chisinau, November 15-19, 2021: Conference Materials / Evrica – Chisinau, 2021. – P.195-197.

ЗМІСТ

Перелік скорочень.....	9
Вступ.....	10
1 Огляд блокчейн технології.....	12
1.1 Визначення блокчейну та його основні принципи.....	12
1.2 Типи блокчейнів.....	14
1.3 Переваги та обмеження блокчейну.....	16
1.4 Механізми блокчейну та інструменти взаємодії з ним.....	17
2 Розуміння мутабельності даних.....	20
2.1 Визначення мутабельності даних в контексті блокчейну.....	20
2.2 Типи змін, які можуть бути здійснені в блокчейні.....	21
3 Вплив мутабельності на захищеність блокчейну.....	28
3.1 Аналіз ризиків, пов'язаних з мутабельністю даних.....	28
3.2 Приклади інцидентів, пов'язаних з мутабельністю даних в блокчейні.....	30
4 Конкретні приклади впливу мутабельності даних.....	33
5 Приклади реалізації механізмів мутабельності з застосуванням мережі блокчейн.....	36
5.1 Застосування механізму мутабельності на рівні смартконтракту.....	36
5.2 Огляд поведінки мережі блокчейн на транзакції мутацій.....	41
5.3 Застосування механізму мутабельності на рівні застосунку.....	57
Висновки.....	68
Перелік джерел посилань.....	70

ПЕРЕЛІК СКОРОЧЕНЬ

ООП - об'єктно орієнтоване програмування;

РМП - ресурсно орієнтована мова програмування;

CLI - інтерфейс командного рядку (Command Line Interface);

FT - взаємозамінні токени (Fungible Token);

GDPR - Загальний регламент про захист даних (General Data Protection Regulation);

JSON - запис об'єктів JavaScript (JavaScript Object Notation);

NFT - невзаємозамінні токени (Non Fungible Token);

PoW - Підтвердження роботи (Proof-of-Work);

PoS - Підтвердження транзакції (Proof-of-Stake);

SoT - Джерело істини (Source of true).

ВСТУП

Блокчейн технологія широко використовується як надійний та безпечний спосіб зберігання та обміну даними. Вона відкриває нові можливості у багатьох галузях, включаючи фінанси, логістику, охорону здоров'я та багато інших. Однак, як будь-яка технологія, блокчейн також не є бездоганним і має свої обмеження.

Одним із найважливіших аспектів безпеки в блокчейні є недоторканність даних. Заснований на принципі незмінності, блокчейн забезпечує гарантовану цілісність і безпеку інформації. Однак, існує поняття мутабельності даних, що означає їх змінюваність або модифікованість.

Огляд функціоналу мутабельності важливий для розуміння його потенційного впливу на захищеність блокчейну. Вимоги щодо безпеки і цілісності даних у блокчейні є критичними, тому вивчення наслідків та ризиків мутабельності має велике значення.

Метою цього дослідження є розуміння, як мутабельність даних може вплинути на захищеність блокчейну та які наслідки це може мати для системи загалом. В рамках роботи будуть досліджені можливість зміни даних, обмеження та правила їх модифікації, а також переваги та недоліки такого підходу до управління даними.

Дослідження буде базуватися на прикладі реалізації відповідного програмного забезпечення та можливостей протоколу блокчейну, включаючи технічні аспекти, протоколи і механізми контролю доступу, а також вивчення впливу мутабельності на безпеку, цілісність та конфіденційність даних.

Дана робота досліджує технології, що дозволяють імітувати мутабельності даних в блокчейні, та виявлення способів підвищення безпеки та цілісності систем. Дослідження цієї теми має важливе значення для розвитку блокчейн технологій та забезпечення їх успішної імплементації в різних сферах діяльності.

1 ОГЛЯД БЛОКЧЕЙН ТЕХНОЛОГІЇ

1.1 Визначення блокчейну та його основні принципи.

Даний розділ надає загальний огляд основних принципів та можливостей блокчейну, який є інноваційним способом зберігання та передачі даних. В цьому розділі будуть розглянуті визначення блокчейну, його ключові принципи, типи блокчейнів та механізми консенсусу, що лежать в основі його роботи.

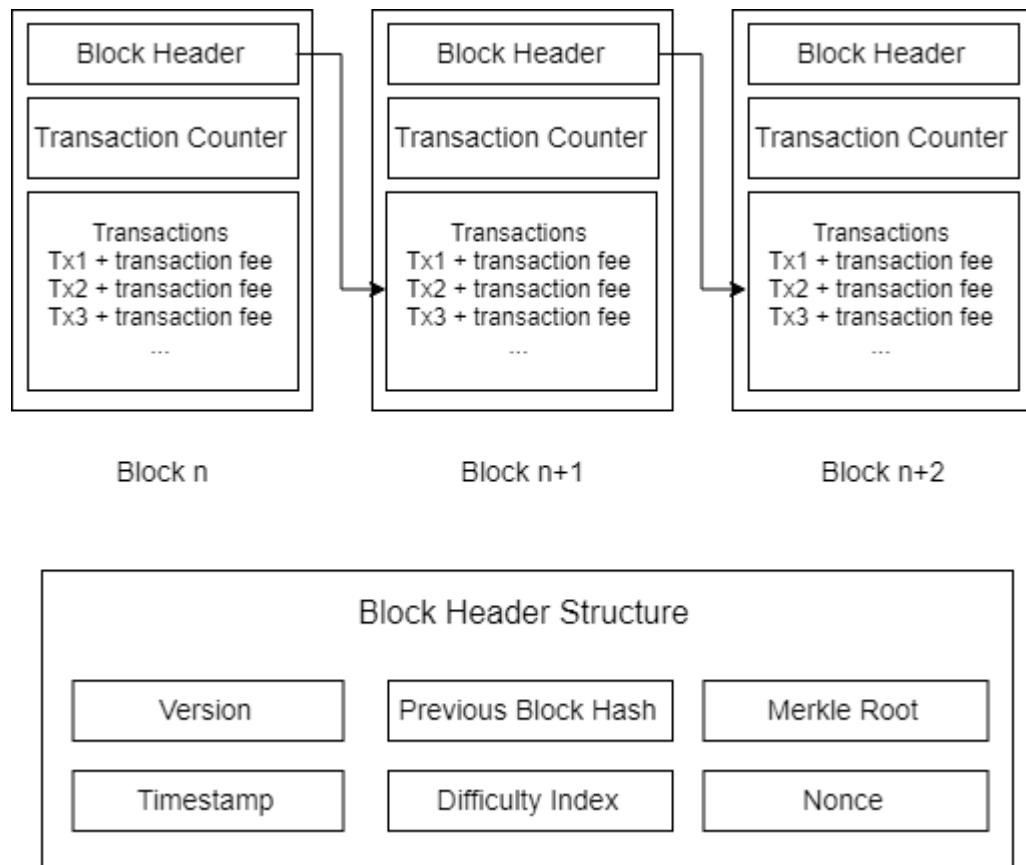


Рисунок 1.1 – Схематичне зображення принципу роботи блокчейну та структури блоків у ньому

Блокчейн (англ. blockchain від block - блок, chain - ланцюг, тобто ланцюжок блоків) - розподілена база даних, що зберігає впорядкований

ланцюжок записів (так званих блоків), що постійно довшає. На рисунку 1.1 наведено схематичне зображення блокчейну[3].

Стюарт Габер і Скотт Сторнетт описали першу роботу над криптографічно захищеним ланцюгом блоків у 1991 році. Їхньою метою було встановити систему, в якій часові позначки документів були неможливими до спотворення або пошкодження. У 1992 році Байєр, Габер і Сторнетт вдосконалили ефективність проекту, використовуючи дерево Меркла, що дозволило включати декілька документів в один блок.

У 2008 році особа (або група осіб), що відома як Сатоші Накамото, розробив задум першого блокчейну. Наступного року Накамото втілює цей задум, створивши основний компонент криптовалюти Bitcoin, який служить відкритою книгою обліку для всіх транзакцій в мережі. Bitcoin став першою криптовалютою, коли проблема подвійних витрат була вирішена без участі довірених вузлів або централізованого сервера. Таким чином, принцип роботи Bitcoin став прототипом для багатьох інших застосувань.

Спочатку, в первинній роботі Сатоші Накамото, слова "блок" і "ланцюг" використовувались окремо, але з 2016 року почали вживатись як одне слово - "блокчейн".

Термін "блокчейн 2.0" походить від нових застосувань розподіленої блокчейн бази даних, що виникли вперше у 2014 році. Одна з реалізацій цього другого покоління блокчейну була описана The Economist як "мова програмування, що дозволяє користувачам створювати більш складні та витончені угоди, які автоматично виконуються після виконання умов". Ці нові можливості допомагають інтегруватись у світову економіку, захищають конфіденційність учасників, дозволяють "монетизувати власну інформацію" і забезпечують компенсацію творцям за їхню інтелектуальну власність. Технологія блокчейну другого покоління також забезпечує стійкий цифровий ідентифікатор особи і пропонує просунутий шлях для вирішення проблеми суспільної нерівності шляхом "потенційної зміни способів розподілу багатства".

До найбільш відомих мереж блокчейну другого покоління можна віднести такі мережі як Ethereum та Flow.

Основні принципи блокчейну включають.

1. Децентралізованість – блокчейн працює в режимі розподіленої мережі, де кожен учасник мережі має копію бази даних. Це виключає залежність від централізованих організацій та забезпечує більшу стійкість та надійність системи.

2. Криптографічна безпека – блокчейн використовує криптографічні методи для забезпечення безпеки та цілісності даних. Кожна транзакція підписується цифровим підписом, а хеш-функції забезпечують унеможливлення зміни даних у блоках без виявлення.

3. Незмінність даних – після того, як блок доданий до ланцюжка, внесенні в нього дані стають імутабельними. Це досягається за рахунок хеш-функцій і посилань на попередні блоки, що ускладнює зміну даних у блоках без зміни всіх попередніх блоків.

4. Консенсус – блокчейн використовує механізми консенсусу для вирішення проблеми подвійних витрат та досягнення згоди між учасниками мережі щодо стану бази даних. Різні алгоритми консенсусу, такі як Proof-of-Work (PoW) та Proof-of-Stake (PoS), використовуються для визначення правильного стану блокчейну.

5. Прозорість – блокчейн забезпечує прозорість участі та транзакцій усіх учасників мережі. Кожен учасник може перевірити та переглянути всі транзакції, що забезпечує довіру та відкритість системи.

1.2 Типи блокчейн мереж

Блокчейн, безсумнівно, визначається як інноваційна технологія, яка використовується в різних галузях, і при цьому її застосування може значно

відрізнятися в залежності від конкретної сфери. Важливо враховувати, що блокчейн – це не просто база даних, а децентралізована система, яка забезпечує цілісність і захист обміну даними.

Умовно можна виділити три види блокчейну:

- публічний блокчейн – цей тип блокчейну відкритий для громадськості. Усі учасники мережі мають можливість брати участь у підтвердженні транзакцій і додаванні нових блоків до ланцюжка. Публічний блокчейн, як, наприклад, у Bitcoin і Ethereum, забезпечує відкритість, прозорість і децентралізацію, дозволяючи кожному користувачеві перевіряти історію транзакцій та стан мережі;

- приватний блокчейн – базується на тих самих принципах, однак «адмініструється» конкретними особами або корпоративно. Учасники приватного блокчейну можуть бути визначені та аутентифіковані, а доступ до даних та процесу підтвердження транзакцій може бути обмежений. Приватні блокчейни застосовуються в бізнес-середовищах, де потрібна більша приватність, контроль та ефективність;

- федерований блокчейн – цей тип блокчейну є комбінацією публічного та приватного блокчейнів. У ньому кілька вузлів або учасників знаходяться під контролем визначених організацій або консорціумів. Федерований блокчейн може бути більш масштабним та швидким, оскільки кількість учасників і вузлів менша, ніж в публічному блокчейні, але він все ще забезпечує децентралізацію.

Приватний блокчейн за своєю суттю є розподіленою приватною базою даних, тому суттєвих особливостей у правовому регулюванні в розумінні GDPR він не викликає.

У сферах, де важливий високий рівень приватності, таких як банкінг чи корпоративна сфера, приватний блокчейн виявляється ефективним вибором. В таких системах доступ до блокчейну обмежений, і лише визначені учасники мають право переглядати чи змінювати інформацію. Це дозволяє

забезпечити конфіденційність операцій та захист від несанкціонованого доступу.

Федерований блокчейн, який представляє собою комбінацію публічного та приватного підходів, може бути використаний в галузях, де необхідна співпраця різних учасників, але при цьому важлива надійність та обмеження доступу. Наприклад, у медичній сфері може бути важливим об'єднання даних від різних медичних установ для реалізації спільних досліджень, забезпечуючи при цьому конфіденційність пацієнтів.

Публічний блокчейн, у свою чергу, використовується там, де відкритість та децентралізація є ключовими аспектами, наприклад, у криптовалютних системах. Це дозволяє всім бажаючим приєднатися до мережі, перевіряти операції та сприяє глобальній доступності.

1.3 Переваги та недоліки блокчейну

Блокчейн - це технологія, яка дозволяє ґрунтовно вирішити проблему безпеки та цілісності даних. Однак, як і будь-яка інновація, блокчейн не є універсальним рішенням і має свої переваги та недоліки.

До переваг блокчейну можна віднести:

- незмінність даних: блокчейн забезпечує незмінність та недоступність для редагування даних, що дозволяє створювати безпечні та надійні транзакції;
- децентралізація: блокчейн працює на основі розподіленої мережі вузлів, що дозволяє уникнути централізованого контролю та залежності від одного центру влади;
- прозорість: всі транзакції в блокчейні є публічними і доступними для перегляду, що забезпечує прозорість та довіру серед учасників мережі;

- висока захищеність: блокчейн використовує криптографічні методи для забезпечення безпеки та конфіденційності даних, що робить його важким для взламу або маніпуляцій.

До недоліків блокчейну, можна віднести:

- незмінність даних: перевага стає доволі великим недоліком у випадку, коли важливі дані були внесені неправильно чи некоректно (як наприклад банківська історія). І дана робота ставить за мету дослідити, як вирішити цю проблему не втративши переваг блокчейну;

- низька швидкість: блокчейн може мати обмежену пропускну здатність та швидкість обробки транзакцій, особливо у випадку публічних блокчейнів;

- витрати та енергоємність: деякі блокчейни, особливо ті, що використовують механізм доказу роботи (PoW), можуть бути енергоємними та вимагати значних витрат на обчислення.

1.4 Механізми блокчейну та інструменти взаємодії з ним

Блокчейн - це мережа з декількох вузлів (нод), що містить частину або всі дані блокчейну. Вузли можуть здійснювати пошук по мережі, запис у мережу або підтверджувати транзакції.

Структура блокчейн представляє собою дерево з блоків, кожен блок пов'язаний один з одним через наявність у нього хешу минулого блоку. Під час здійснення запису в блокчейн, вузол обирає будь-який блок з ланцюжка, який він відслідковує, та рахує його хеш, додає до свого локального сховку дані про новий блок з хешем старого, та відправляє результат на узгодження з іншими вузлами. Після процесу узгодження, новий блок буде надісланий всім іншим вузлам, для додавання його у всю мережу.

Взаємодія з блокчейном обмежується операціями 2 типів: читанням та записом. Операції читання повертають оброблені дані з блокчейну. Процес та алгоритм обробки даних надсилається на вузол блокчейну. Після надходження запиту на отримання інформації на вузол, ці дані знаходяться у ланцюжку, і в залежності від можливостей протоколу блокчейн мережі, вони обробляються. Здатність перетворювати дані за сценарієм, який надсилається користувачем притаманна лише для мереж 2 покоління. Дані запити можна порівняти з оператором SELECT, який використовується в SQL. Зазвичай протоколи блокчейну не передбачають механізму комісії для запитів на читання.

Операції, що передбачають додавання нових блоків до ланцюжка, тобто операцій запису, називають – транзакціями. Дані операції є доволі складними та потребують значних розрахункових потужностей. З цієї причини більшість публічних блокчейн протоколів передбачають механізм комісії за операції запису.

Слід зазначити, що можливості та інструменти мереж блокчейну першого та другого поколінь сильно відрізняються. В мережах першого покоління вся логіка та взаємодія з блокчейном обмежена самим протоколом, і не дозволяє розширювати функції, які вже передбачені. В свою чергу мережі другого покоління, такі як Ethereum, мають функціонал смарт-контрактів (Smart Contracts). Смарт-контракт – це самовиконуючий код, який містить правила та умови угоди між двома або більше сторонами. Цей код працює на блокчейні та автоматизує виконання угоди, виконуючи програмовані умови при визначених подіях чи обставинах без необхідності посередників чи централізованих авторитетів. Смарт-контракти використовують технологію блокчейн для забезпечення надійності, сталості та безпеки виконання угод.

Оскільки мережі блокчейн першого покоління не мають технології смарт-контрактів, то їх функціонал чітко обмежений. І взаємодія з мережею блокчейну обмежена чіткими, заздалегідь передбаченими запитамі, типами

даних та транзакціями. Що майже повністю прибирає необхідність будь-яких інструментів мутабельності. Такі мережі зазвичай вирішують виключно одну задачу, як то криптовалюта чи система колекціонування та передачі карток.

```

47 pub contract interface NonFungibleToken {
48
49     /// The total number of tokens of this type in existence
50     pub var totalSupply: UInt64
51
52     /// Event that emitted when the NFT contract is initialized
53     ///
54     pub event ContractInitialized()
55
56     /// Event that is emitted when a token is withdrawn,
57     /// indicating the owner of the collection that it was withdrawn from.
58     ///
59     /// If the collection is not in an account's storage, `from` will be `nil`.
60     ///
61     pub event Withdraw(id: UInt64, from: Address?)
62
63     /// Event that emitted when a token is deposited to a collection.
64     ///
65     /// It indicates the owner of the collection that it was deposited to.
66     ///
67     pub event Deposit(id: UInt64, to: Address?)
68
69     /// Interface that the NFTs have to conform to
70     /// The metadata views methods are included here temporarily
71     /// because enforcing the metadata interfaces in the standard
72     /// would break many contracts in an upgrade. Those breaking changes
73     /// are being saved for the stable cadence milestone
74     ///
75 > pub resource interface INFT {--
96     }
97
98     /// Requirement that all conforming NFT smart contracts have
99     /// to define a resource called NFT that conforms to INFT
100    ///
101 > pub resource NFT: INFT {--
102    }
103
104
105    /// Interface to mediate withdraws from the Collection
106    ///
107 > pub resource interface Provider {--
108    }
109
110
111    /// Interface to mediate deposits to the Collection
112    ///
113 > pub resource interface Receiver {--
114    }
115
116
117
118
119
120
121
122 >
123
124
125
126
127
128
129
130

```

Рисунок 1.2 – Приклад смартконтракту в блокчейні Flow (смартконтракт розташований за адресою 0x1d7e57aa55817448 та має назву NonFungibleToken)

Блокчейн мережі другого покоління, за рахунок технології смарт-контрактів, дають змогу здійснювати будь-які операції, в межах можливостей їх протокола. Як вже зазначалося, заздалегідь, дана можливість досягається за рахунок технології смартконтрактів. Уявити смартконтракт

можна як звичайний програмний клас, який наявний у будь-якій об'єктно орієнтованій мові програмування (JavaScript, C++, C#, Python тощо). На рисунку 1.2 зображено приклад смартконтракту в блокчейні Flow.

Як і класи з об'єктно орієнтованих мов програмування, смартконтракт містить в собі публічні та приватні методи, публічні та приватні змінні, конструктор. Деякі протоколи також дозволяють здійснювати наслідування різних смартконтрактів один від одного.

Також слід зазначити, що у протоколів блокчейну 2 покоління взаємодія з даними є специфічною. Вона полягає в тому, що посилання та реальні дані з ланцюжків чітко розділені. Наприклад: якщо під час транзакції користувач використовує оператор присвоєння до даних з блокчейну, він обов'язково має застосувати оператор запису даних в блокчейн, інакше транзакція просто не запуститься. Також всі дані, що існують під час читання та обробки в операціях читання, це посилання на дані з блокчейну. Дана специфіка викликана фундаментальними правилами блокчейну, які зазначають, що дані не можуть бути видалені з нього.

2 РОЗУМІННЯ МУТАБЕЛЬНОСТІ ДАНИХ

2.1 Визначення мутабельності даних в контексті блокчейну

Мутабельність – можливість змінювати дані, є звичною операцією для більшості систем. Цей аспект має значний вплив на цілісність і безпеку системи, оскільки недоречне втручання в існуючі дані може спотворити їх достовірність та викликати серйозні наслідки.

Концепція блокчейну забороняє змінювати дані в блоках, це аспект, що дозволяє гарантувати цілісність даних в системі, та робить саму технологію блокчейну затребовану. У типових блокчейн мережах, як наприклад Bitcoin і Ethereum, дані, які записуються в блок, зазвичай є незмінними та недоступними для редагування, то якщо надати можливість мутувати дані в блокчейні, змін зазнає не інформація, що зберігається в блоках, а сам принцип обробки та читання ланцюжка, що дозволяє імітувати змінність даних.

Однак, варто враховувати, що навіть програмна імітація мутабельності даних може вплинути на безпеку та незмінність блокчейну. Зміни вже існуючих даних можуть порушити цілісність системи, викликати суперечності або навіть використовуватися для зловживань.

У кожному конкретному випадку, обсяг та межі мутабельності даних в блокчейні будуть залежати від визначених правил, протоколів та дизайну системи. Це вимагає уважного аналізу та збалансованого підходу до розуміння та впровадження мутабельності в блокчейні з метою забезпечення необхідної функціональності та захисту системи.

У цій частині розділу буде розглянуто сутність поняття мутабельності даних в контексті блокчейну. Визначимо, що мається на увазі під мутабельністю даних і як вона відрізняється від імутабельності, що є однією з основних властивостей блокчейн технології. Докладно проаналізуємо, які

типи даних можуть бути мутабельними в блокчейні і які наслідки можуть впливати з їхньої мутабельності.

Імутабельним (англ. immutable) називається об'єкт, стан якого може бути змінено після створення. Результатом будь-якої модифікації такого об'єкта буде новий об'єкт, при цьому старий об'єкт не зміниться.

Мутабільність або мутабельність (від англ. Mutable — «той, що може змінюватися») — поширений термін, котрий описує здатність предмету (суб'єкту) змінювати свої основні характеристики, властивості.

Мутабельність є протилежністю (антитезою, антонімом, антиподом) до поняття (терміну) імутабельності. Але одним з фундаментальних властивостей блокчейн мереж є саме сталість даних (тобто імутабельності) в ланцюжку. Принцип імутабельності даних в ланцюжку є непорушним, відповідно мутабельність в блокчейні присутня, як набір протоколів та алгоритмів, що лише імітують можливість мутації даних. Спираючись на дану інформацію, можна дати визначення мутабельності даних у блокчейні.

Мутабельність даних (у блокчейні) – це імітація змінності або можливість змінювати представлення кінцевих даних, що зберігаються в блокчейні та надаються користувачу, без порушення принципу імутабельності. Слід зазначити, що дані, які зберігаються в блокчейні, не зазнають змін (в іншому випадку порушуватиметься головний принцип блокчейну “сталість даних”), це досягається різними семантичними прийомами.

2.2 Типи змін, які можуть бути здійснені в блокчейні

Так як мутабельність даних у блокчейні існує лише на рівні протоколу, вона досягається купою різних способів. Можна відокремити два основні

підходи, що дозволяють мутувати дані в блокчейні. Головна відмінність цих двох методів - чи є транзакція зміни, окремим видом операції.

Метод, що сприймає запит (транзакцію) на зміну даних, як окремий вид операції зазвичай складається з двох паралельних ланцюжків. В мережі наявний головний ланцюжок, який використовується для звичних потреб мережі, та другорядний ланцюжок, який в свою чергу зберігає дані, які описують історію інструкцій, які необхідно здійснити над батьківським блоком, що зберігається в першому ланцюжку, перед його відображенням користувачу. Даний метод можна застосувати лише на рівні самого протоколу мережі блокчейн. На рисунку 2.1 наведене схематичне зображення реалізації даного методу

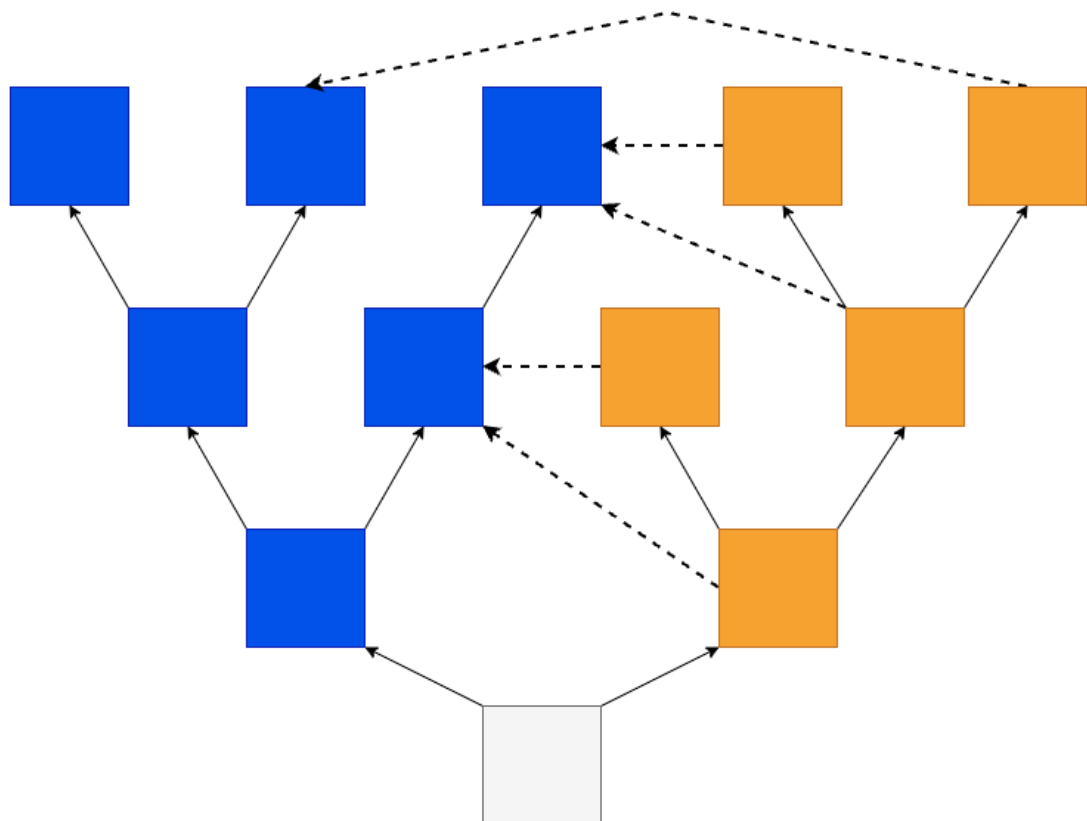


Рисунок 2.1 – Схематичне зображення мережі з реалізацією першого методу (синім показані транзакції основної підмережі; помаранчевим транзакції підмережі змін, пунктирні стрілки – показчики на батьківський блок)

Так як даний підхід можливий до застосування лише на рівні самої мережі блокчейн, він не є популярним. Також недоліком такого підходу є необхідність передбачити окремий тип операцій update. Тобто на рівні зі звичайними транзакціями, на запис будуть існувати окремо операції на оновлення.

На противагу методу з двома ланцюжками, існує інший метод. Він полягає у додаванні в протокол для вузлів блокчейну нової директиви, що при запиті на зміну даних, додає нові дані в основний ланцюг блоків дані, що посилаються на вже існуючий (батьківський) блок. Новостворені дані будуть містити інструкції, що вказують на зміни, які повинні відбутися з батьківським блоком перед видачею його користувачеві, або він може містити вже перетворені дані. Слід зазначити, що даний метод може організовуватися не лише на рівні блокчейну, такий підхід можна організувати на рівні застосунку також.

В свою чергу метод, що не використовує додаткові ланцюжки, вирізняється за методом зв'язування блоків та методом застосування змін. Різниця у алгоритмах реалізації полягає у тому, як саме знаходити, перетворювати та повертати інформацію з ланцюжка. Можна виокремити два підходи до зв'язування блоків.

1. З унікальним ідентифікатором та покажчиком на батьківський блок. Метод полягає у тому, що інструкції знаходяться в нових транзакціях, які, як і будь-яка інша транзакція, містить унікальний ідентифікатор, а також має посилання (в тілі або в заголовку) на батьківський блок. Тобто під час запиту на певний блок ми крім пошуку даного блоку, також здійснюємо пошук всіх блоків, що мають покажчик на шуканий блок. Після чого здійснюють, або зміни, відповідно до інструкцій, або підміну даних, відповідно до вмісту блоків з посиланням на шуканий блок. На рисунку 2.2 наведено схематичне зображення мережі з такою реалізацією.

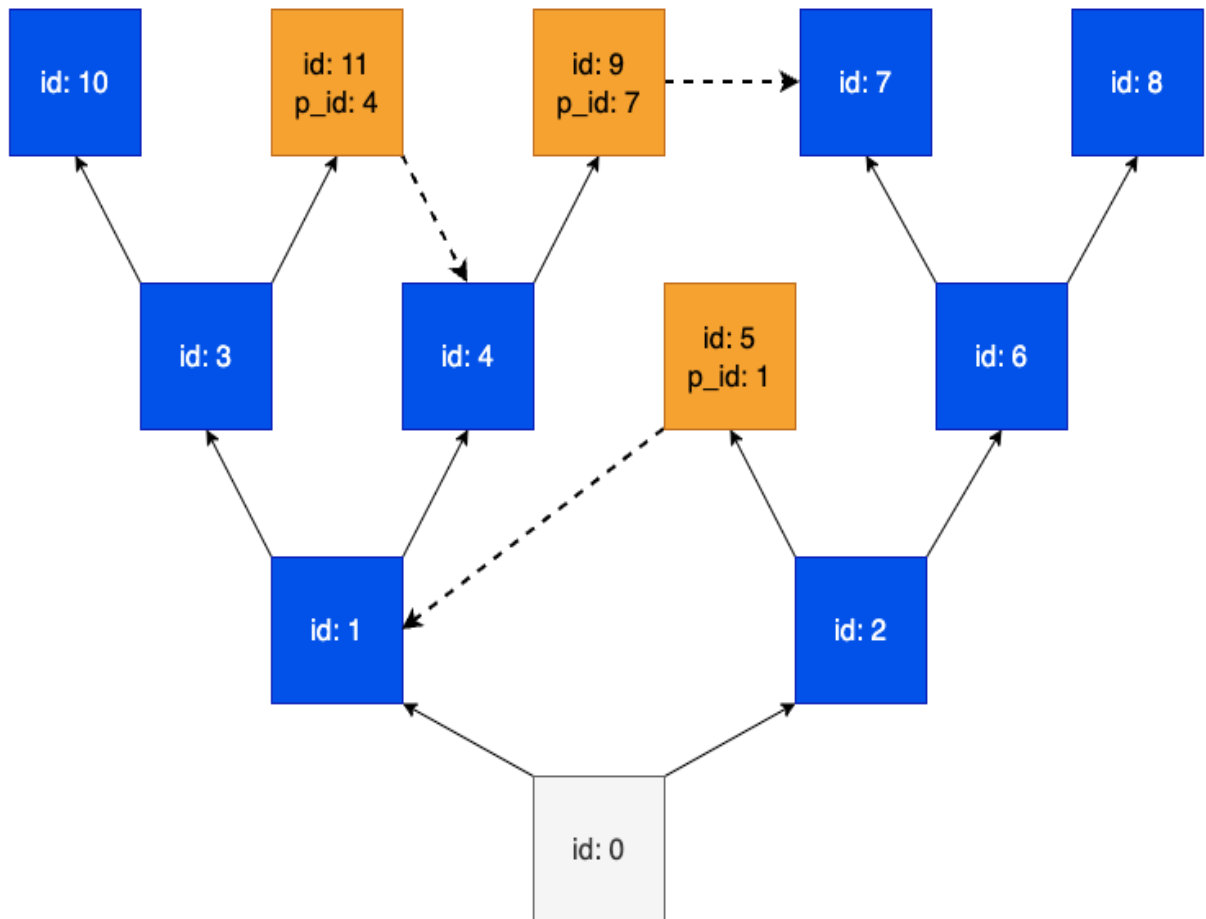


Рисунок 2.2 – Схематичне зображення мережі що реалізує метод, з унікальним ідентифікатором та показником на батьківський блок (пунктирні стрілки – показники на батьківський блок)

2. Без унікального ідентифікатора. Метод полягає у тому, що до мережі додаються нові транзакції, які не мають свого унікального ідентифікатора, їх ідентифікатор дублює батьківський ідентифікатор та одночасно слугує посиланням на батьківський блок. Завдяки цій особливості під час запиту на конкретний блок, ми отримуємо масив транзакцій з однаковим ідентифікатором. Цей масив складається з батьківського блоку та блоків інструкцій, або блоків з оновленою інформацією. І відповідно до отриманих даних ми можемо сформувати результуючу відповідь для користувача. На рисунку 2.3 наведено схематичне зображення мережі яка реалізує даний метод.

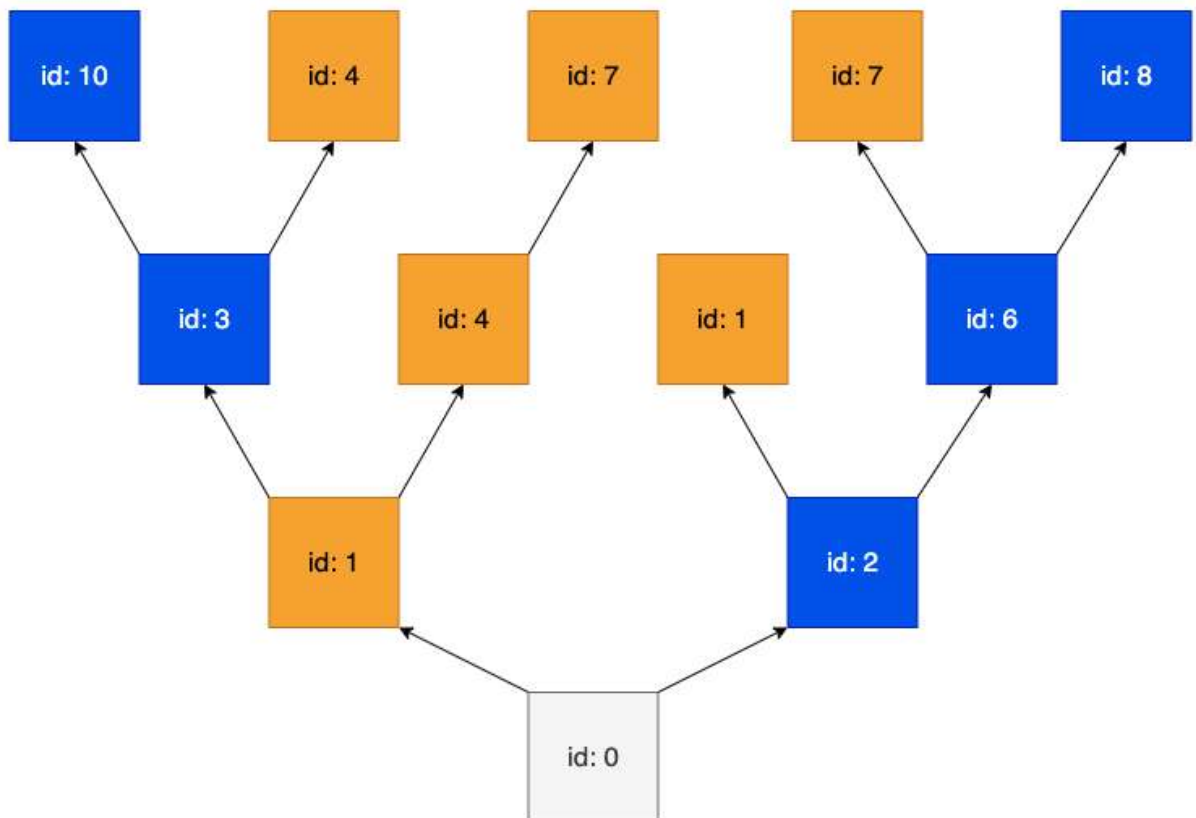


Рисунок 2.3 – Схематичне зображення мережі з реалізацією другого методу, без унікального ідентифікатора

Механізми імітації мутабельності даних в блокчейні також можна поділити за тим, як застосовуються зміни до батьківського блоку. За методом застосування змін можна виокремити такі підходи.

1. заміна існуючої інформації. Метод полягає у тому, що блоки змін, зберігають одразу оновлені дані. У такому випадку, під час пошуку, користувачу одразу повернеться остання “версія” шуканого блоку. Даний підхід має великий недолік – щоб відслідкувати історію змін, необхідно аналізувати весь сховок даних блокчейну, бо під час запиту не надсилаються попередні версії поточного блоку. Але даний метод також має вагому перевагу, так як значно прискорює пошук та відображення актуальних даних. Схематичне зображення ланцюжка, що реалізує даний методу наведено на рисунку 2.4;

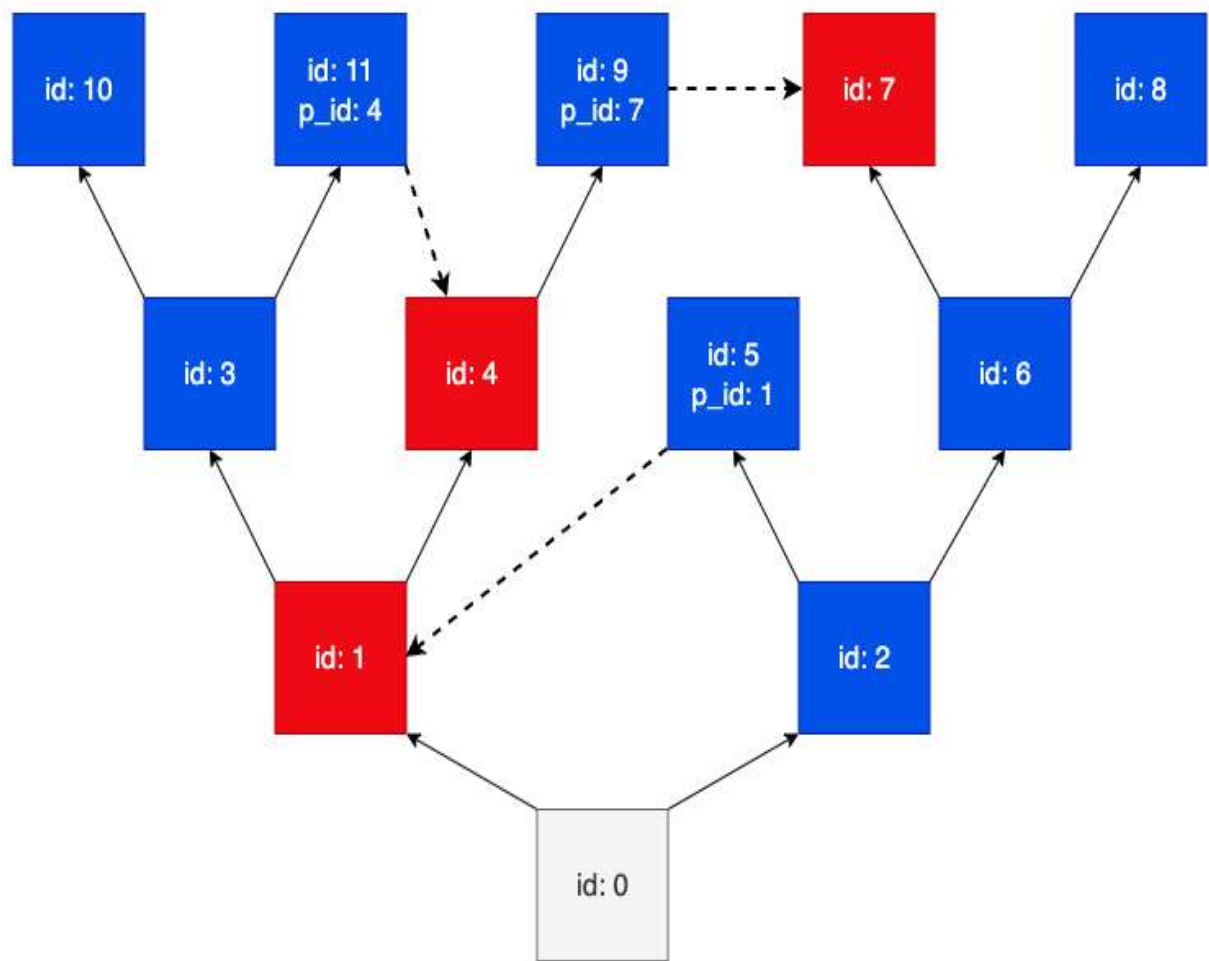


Рисунок 2.4 – Схематичне зображення мережі з реалізацією заміни існуючого блоку (червоні показані транзакції, що не будуть враховуватись при звичайному пошуку)

2. застосування історії. Метод полягає у збереженні історії змін в блоках, історія застосовується до батьківського шуканого блоку, перед його відображенням користувачу. Даний метод подібний до системи Git (розподілена система керування версіями), ми маємо батьківський шуканий блок та блоки, в яких вказано, як треба змінити батьківський блок перед його відображенням. Головним недоліком цього методу є складність реалізації та час обробки даних, дані показники значно поступаються першому методу. На рисунку 2.5 наведене схематичне зображення мережі, що реалізує даний метод застосування змін.

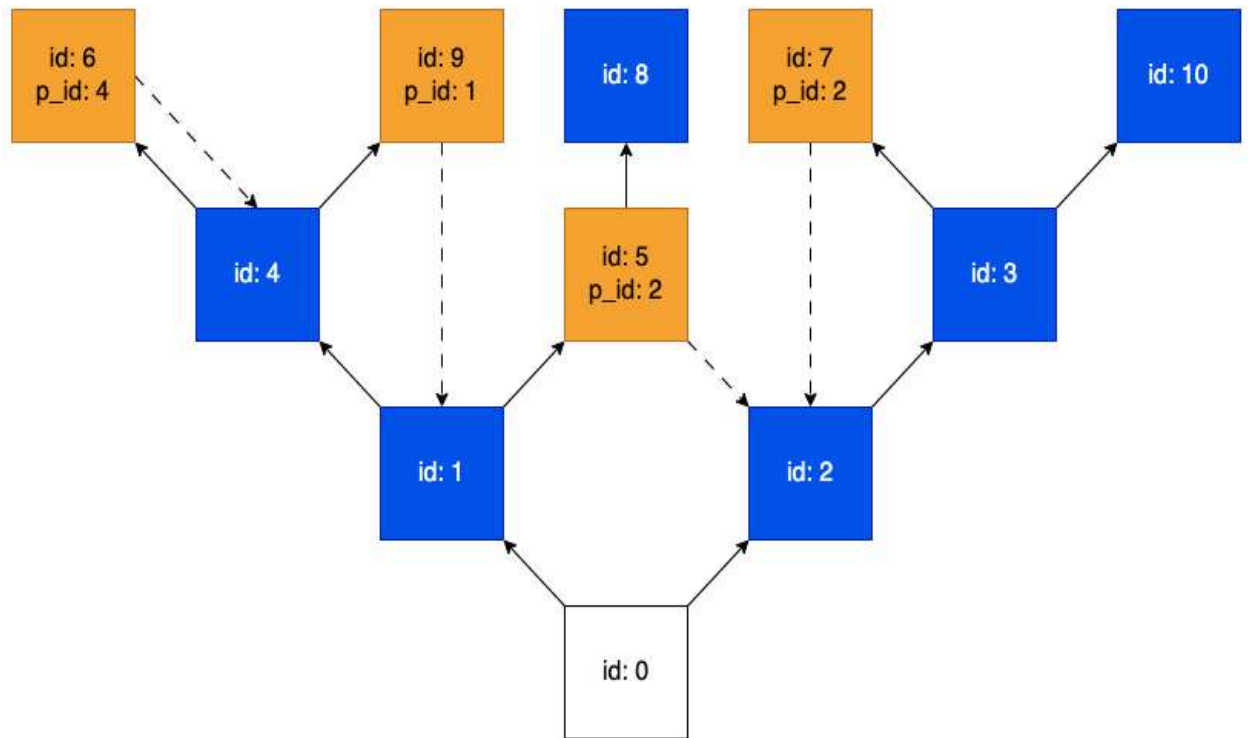


Рисунок 2.5 – Схематичне зображення мережі з реалізацією історії версій блоку (помаранчевим показані транзакції, що містять історію змін в батьківському блоці, пунктиром показані посилання на батьківський блок)

Обидва методи мають свої переваги та недоліки. Хоч вони по суті є лише семантичним доповненням блокчейну, але ці методи надають інструмент видозміни даних в блокчейні. Незважаючи на те, що насправді дані в блоках не мутують, це все одно створює ряд проблем, бо ми вже не можемо гарантувати цілісність даних, адже тепер при викритті секретного ключа, не можна гарантувати, що останні перетворення були внесені не злоумисником.

3 ВПЛИВ МУТАБЕЛЬНОСТІ НА ЗАХИЩЕНІСТЬ БЛОКЧЕЙНУ

3.1 Аналіз ризиків, пов'язаних з мутабельністю даних

В даному розділі буде розглянуто загрози, що виникають в блокчейнах, які реалізують методи, що дозволяють імітувати мутабельності даних. А також будуть наведені приклади мереж, що реалізовували дані методи для відтворення мутабельності, та розглянуті проблеми, з якими зіткнулися дані мережі під час експлуатації різних мутабельностей.

Перша проблема, що виникає при додаванні функціоналу, вразливість несанкціонованих змін. Дана вразливість полягає у тому, що ми не можемо гарантувати, що останні зміни були внесені не зловмисником. Дана вразливість притаманна і звичайному блокчейну, так як можливість внесення даних від вашого імені, можлива лише якщо зловмиснику буде доступний секрет гаманця, тому дану вразливість можна віднести до людського фактору. Але у випадку звичайного блокчейну, момент викриття секретного ключа відслідковується краще.

Атака подвійного витрачання – це заздалегідь передбачена атака, або збій у алгоритмі підтвердження транзакції, при якому один і той же запис потрапляє у ланцюг блокчейну двічі. При застосуванні будь-якого з методів, що імітують мутабельність даних в блокчейні, збільшується навантаження на мережу блокчейну, адже окрім запитів на створення та видачу даних з'являються також запити на оновлення даних. При збільшенні кількості запитів виникає збільшення небезпека атаки на подвійне витрачання. Адже через великий потік запитів вірогідність неузгодження між декількома вузлами збільшується, відповідно можливе дублювання даних. Але блокчейн мережі, у яких PoW є централізованим процесом, уникають атак подвійного витрачання, так як PoW виконують лише певні синхронізовані вузли. Недоліком таких мереж є те, що перевага блокчейну, як незалежного сховища даних, нівелюється залежністю від конкретної малої кількості вузлів.

Вразливість більшості (або атака 51%) – атака, яка виникає при концентрації $> 50\%$ технічних ресурсів блокчейну під контролем зловмисника, або групи зловмисників, що дозволяє їм вносити будь-які транзакції без необхідності їх підтвердження у інших учасників. В блокчейні, який використовує метод мутабельності, що передбачає додаткову підмережу, також збільшується вірогідність атаки більшості (або атака 51%). Тому що зловмиснику або групі зловмисників тепер немає необхідності складати більшість всієї системи, їм достатньо стати більшою частиною підмережі ланцюжка змін. Слід зазначити що другий метод хоч і вирізняється більшим навантаженням на мережу ніж перший, але з точки зору атаки більшості є більш захищений навіть за звичайну мережу блокчейну. Через дану проблематику відкриті блокчейн мережі взагалі не використовують перший метод на практиці. Але ця проблема також нівелюється, якщо процес PoW є централізованим процесом, але, як було зазначено вище, даний підхід має свої недоліки.

Також слід зазначити, що наявність механізмів мутування даних в блокчейні, також призводить до збільшення навантаження на мережу. Цей фактор може призвести до таких негативних факторів, як: збільшення часу на обробку транзакції, збільшення плати за транзакцію у “майнерів” мережі, можливих відмов на обробку транзакцій, тощо. Але навантаження на систему має певні переваги для блокчейн систем, які будуть розглянуті у розділі 4.

Найбільшим недоліком механізму мутації даних в блокчейні є людський фактор. Він може відігравати роль ключового фактора, що надає зловмисникам можливість отримувати доступ до приватних даних та підміняти дані в мережі, що тягне за собою ще ряд загроз для користувачів. Незважаючи на технологічний прогрес та впровадження заходів безпеки, недбалість та недостатня увага користувачів можуть створювати серйозні ризики для їхньої кібербезпеки: підміни даних документа, що зберігається в мережі; зміна суми транзакції, тощо. Хоч принцип імутабельності в блокчейн мережі дозволяє відслідкувати та переглянути всі зміни, слід зазначити, що

більшість користувачів блокчейн мереж не розуміють специфіку та цілі такого інструменту – це в свою чергу призводить до необачності, що має свою ціну.

3.2 Приклади інцидентів, пов'язаних з мутабельністю даних в блокчейні

Мутабельність даних в блокчейні дуже корисний інструмент, хоча й небезпечний. В наш час він вже використовується деякими блокчейнами, наприклад Flow (це найбільший представник блокчейну в якому мутабельність даних представлена у повному вигляді), або найбільший представник сучасних блокчейнів Ethereum. Мутабельність є просто життєво необхідним атрибутом сучасних blockchain 2.0 мереж, адже вони дозволяють формувати доповнення до основного протоколу, що встановлює свої правила створення транзакцій – смартконтрактами. Смартконтракти зберігаються в основному ланцюгу блокчейну, основним призначенням мутабельності в блокчейні є надання можливості змінювати вміст смартконтракту, що дозволить скоротити розмір мережі.

Так як можливість мутації смартконтрактів є вбудованим функціоналом у протоколі більшості представників blockchain 2.0, то колізій з правами доступу до редагування певних смартконтрактів, майже не виникає. Адже доступ до можливості створення транзакції, що включають зміни для смартконтракту, має лише той гаманець (або у випадку flow акаунт), котрий створив транзакцію з батьківським смартконтрактом.

Також користувач, що створив смартконтракт може надавати можливість мутувати дані, створенні через смартконтракт. Але смартконтракти, як і будь-яка технологія, не є повністю безпечними і можуть мати певні небезпеки, пов'язані з їх використанням користувачами. Наведені

вразливості, що притаманні смартконтрактам, які мають можливість мутувати дані, такі як.

1. Вразливості програмного забезпечення – смартконтракти побудовані на програмному забезпеченні, яке може містити помилки, баги або вразливості. Це може призвести до некоректної роботи контракту або використання зловмисниками для виконання шкідливих дій.

2. Недостатня безпека аудиту – смартконтракти можуть мати складну логіку і велику кількість умов, що важко аудитувати. Це може призвести до пропуску потенційних вразливостей або недоліків в безпеці контракту.

3. Атаки на вразливості – зловмисники можуть експлуатувати вразливості смартконтрактів, щоб виконати шкідливі дії, такі як крадіжка коштів або вплив на результати операцій. Недбале програмування або відсутність достатнього тестування можуть створювати простір для таких атак. На рисунку 3.1 наведений приклад функції оновлення, через яку може здійснюватись атака.

```
pub fun update(report: HistoryPoint){
    self.history.append(
        HistoryPoint(
            metadata: self.metadata,
            file: self.file,
            data: self.data
        )
    )
    self.metadata = report.metadata
    self.file = report.file
    self.data = report.data
}
```

Рисунок 3.1 – Приклад функції оновлення у смартконтракті DocHub у ресурсі NFT[13] (смартконтракт знаходиться за адресою 0x88b76b3b47dec36d)

Також можна навести декілька прикладів вразливостей, що виникали через наявність механізму мутацій даних. Наприклад на перших версіях блокчейну Flow існувала проблема, яка полягала в тому що при співпадінні хешів двох смарт контрактів (контракту 1 та контракту 2), наступна версія наприклад контракту 2, могла бути прийнята протоколом, також, як наступна версія контракту 1. На даний час ця проблема вирішена.

Також зустрічаються випадки, коли розробник смартконтракту не передбачав контролю доступу до створених нових ресурсів, що в свою чергу дозволяв зловмиснику здійснювати атаку (створювати несанкціоновано NFT чи FT, створювати несанкціоновані транзакції, змінювати вже створені дані, тощо). Дана вразливість є прикладом впливу людського фактора під час розробки алгоритмів смартконтракту. Уникнення таких помилок вимагає системної роботи тестувальників, та може зустрічатися не лише у контексті блокчейну, але у класичній реалізації блокчейн протоколів, подібні вразливості не можливі з технічної точки зору.

4 КОНКРЕТНІ ПРИКЛАДИ ВПЛИВУ МУТАБЕЛЬНОСТІ ДАНИХ

У контексті блокчейну, поняття мутабельності даних має не лише негативний, але й позитивний вплив на систему. Однією з головних переваг додавання функціоналу, що реалізує мутацію даних, є зменшення ймовірності, що зловмисники можуть обійти блокчейн, рахуючи хеши ланцюжків швидше, ніж сама система.

Слід також зазначити, що головною перевагою функціоналу, що надає змогу імітувати мутабельність, є збільшення навантаження на мережу блокчейн. Завдяки можливості змінювати дані, зловмисникам необхідно мати значну потужність, щоб перевершити потужність самої мережі блокчейн. Таким чином, несанкціонована зміна даних в блокчейні стає малоімовірною, оскільки це можливо лише за умови, що зловмисник має розрахункову потужність, достатню для здійснення операції швидше за основну мережу.

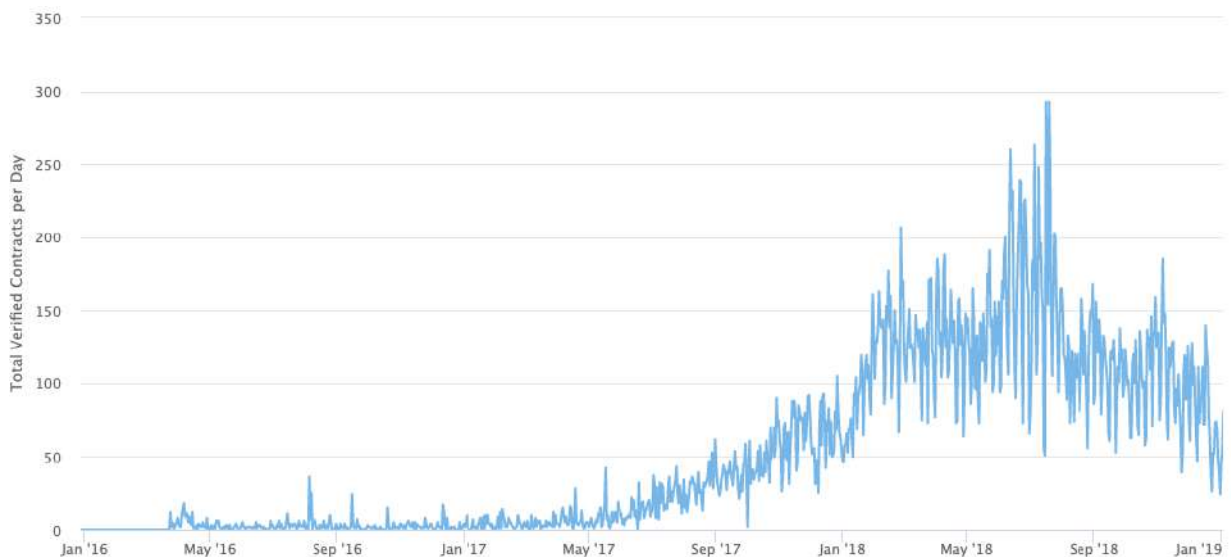


Рисунок 4.1 – Кількість смартконтрактів створених в мережі Ethereum за добу, в проміжку з 14 червня 2016 по 19 червня 2019 року

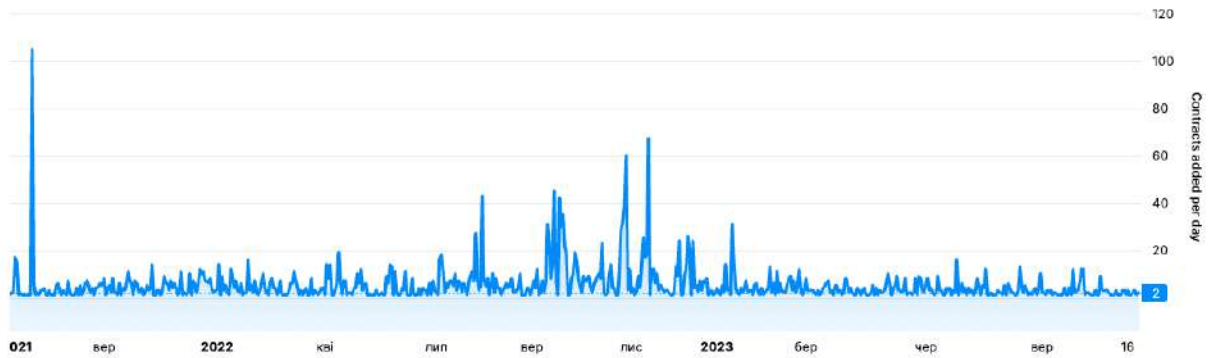


Рисунок 4.2 – Кількість смартконтрактів створених в мережі Flow за добу, в проміжку з 14 грудня 2020 по 25 листопада 2023 року

Особливо ефективною стає ця перевага у випадку, коли система блокчейну працює з великим навантаженням та має пропорційно велику кількість вузлів. Фізично неможливо мати достатню потужність, щоб випередити велику кількість вузлів. Таким чином, мутабельність даних у блокчейні допомагає забезпечити стійкість та надійність системи.

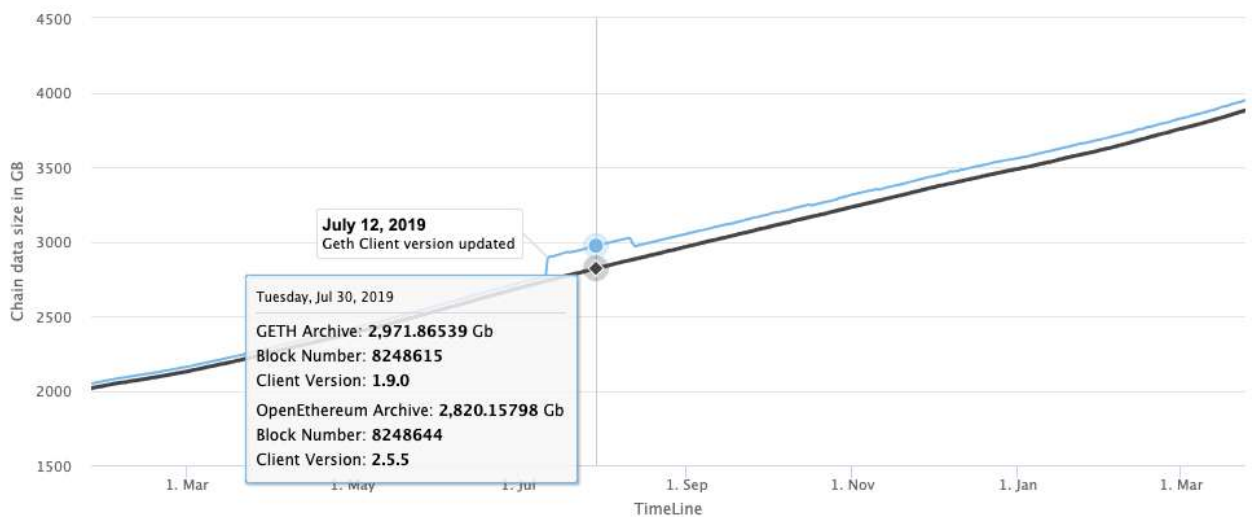


Рисунок 4.3 – Розмір мережі Ethereum та кількість транзакцій ланцюгу на 30 липня 2019 року

Слід зазначити, що наявність певних методів мутацій дозволяє також зменшити загальний розмір блокчейну. Досягти зменшення загального

розміру блокчейну дозволяють всі методи мутабельності, крім методу, який безпосередньо вносить у блокчейн нову версію блоку, яка тепер буде вважатися останньою. Це досягається за рахунок того, що у випадку створення некоректних даних, ми не маємо необхідності створювати новий коректний екземпляр даних, а просто зберігаємо зміни новою транзакцією. Але даний метод має певний ряд інших переваг, що пришвидшують роботу з мутованими даними.

Для прикладу можна взяти блокчейн Ethereum, який до 2022 року не мав механізмів мутабельності, та блокчейн Flow, що з самого початку підтримував дані механізми. За період з 14 березня 2016 року, коли Ethereum вийшов з ранньої альфа-версії, по 30 липня 2019 року кількість транзакцій в блокчейн налічувала 8,248,644 транзакцій, а загальний розмір даних в мережі дорівнював 2971.86 Гб (рисунок 4.3). В свою чергу розмір блокчейну Flow з моменту запуску, у жовтні 2020 року, і по сьогоднішній день налічує 401,129,954 транзакцій, а загальний розмір мережі не перевищує 500 Гб.

Також слід зазначити, що за аналогічні періоди протяжністю 3 роки, з запуску блокчейнів Ethereum та Flow, вони налічували понад 5000 та 3119 смартконтрактів відповідно (рисунок 4.1 та 4.2). Дана тенденція спричинена відсутністю механізму мутабельності в блокчейн мережі Ethereum, що призводило до необхідності створювати кожен раз новий смартконтракт з відповідними змінами. Менша кількість контрактів дозволяє легше орієнтуватись пошуковим системам блокчейну, а також має, хоч і незначний, вплив на розміри блокчейну. І зменшення кількості смартконтрактів досягається шляхом застосування механізму мутабельності до них.

Розуміння впливу мутабельності даних в блокчейні на захищеність є важливим аспектом дослідження. Розглядання переваг та обмежень цього функціоналу допоможе зрозуміти, як він може вплинути на безпеку системи та які заходи слід прийняти для забезпечення її надійності.

5 ПРИКЛАДИ РЕАЛІЗАЦІЇ МЕХАНІЗМІВ МУТАБЕЛЬНОСТІ З ЗАСТОСУВАННЯМ МЕРЕЖІ БЛОКЧЕЙН

5.1 Застосування механізму мутабельності на рівні смартконтракту

У даному розділі будуть наведені приклади реалізації механізмів мутації даних з застосуванням мережі блокчейн. Приклади будуть стосуватися як програмної реалізації так і реалізації за допомогою смартконтрактів. Також буде наведено розбір механізму мутації в блокчейні Flow на рівні протоколу.

Для розгляду та відтворення механізмів мутації будуть застосована блокчейн мережа Flow. Це третя за популярністю блокчейн мережа, цільовим застосуванням якої є робота з Non Fungible Token (NFT), також передбачені стандарти реалізації та відображення інформації для NFT та для FungibleToken, що полегшує роботу та розробку різних смартконтрактів. Мова програмування смартконтрактів у блокчейні Flow є Cadence. Мова натхненна такими мовами програмування як GoLang, C# та TypeScript.

Для реалізації механізмів, що імітують мутабельність даних в блокчейні, на рівні застосунку, буде використовуватись програмний пакет NodeJS.

Розглянемо приклад застосування механізму мутації на прикладі смартконтракту DocHub, в тестовій мережі блокчейну Flow за адресою 0xf8d6e0586b0a20c7 [13]. Метою цього смартконтракту є створення, збереження та оновлення даних про документи. Документ в цьому смартконтракті буде розглядатись як NFT.

Як було зазначено вище, смартконтракти блокчейн мережі Flow, пишуться мовою програмування Cadence. Анонсована у 2020 році мова програмування Cadence є ресурсорієнтованою мовою програмування (РМП). РМП – це мови програмування, що представляють набір чітко визначених методів для взаємодії різних компонентів. Cadence є жорстко типізованою,

об'єктно орієнтованою мовою програмування, та дозволяє легко писати максимально безпечні смарт-контракти для блокчейн мережі Flow [16].

Кожен смартконтракт представляє собою конструкцію, схожу на namespace (область видимості) у таких мовах програмування як TypeScript, C# або C++. Смартконтракт містить в собі набір інтерфейсів, ресурсів, змінних, функцій та обов'язково має містити конструктор (функція що ініціалізує даний смартконтракт). Конструкція оголошення смартконтракту виглядає наступним чином:

```
pub contract SmartContractName {}
```

у цьому конкретному прикладі смартконтракту DocHub, оголошення буде виглядати наступним чином:

```
import NonFungibleToken from 0x631e88ae7f1d7c20
pub contract DocHub: NonFungibleToken {}
```

Як можна помітити, тут також відбувається імпорт зовнішнього смартконтракту, та використання синтаксису наслідування від даного зовнішнього смартконтракту під назвою NonFungibleToken, що знаходиться за адресою 0x631e88ae7f1d7c20. Смартконтракт NonFungibleToken - це абстрактний смартконтракт, передбачений розробниками флоу, з ціллю узгодження підходів до розробки смартконтрактів, які за мету мають реалізацію механізмів NFT. Також у Flow передбачений аналогічний абстрактний клас для розробки криптовалют, під назвою FungibleToken, що знаходиться за адресою 0x631e88ae7f1d7c20.

Наступним кроком у створенні смартконтракту є створення функції ініціалізації (конструктора) даного смартконтракту. Так як виклик конструктора смартконтракту здійснюється перед його створенням, у ньому будуть передбаченні значення змінних, які будуть оголошені у цьому смартконтракті. Далі наведений приклад конструктора смартконтракту DocHub:

```
init(){
    self.totalSupply = 0
    self.storagePath = /storage/twentyFourMiles
```

```

        self.publicPath = /public/twentyFourMiles
        self.privatePath = /private/twentyFourMiles
    }

```

Як ми бачимо, в конструкторі смартконтракту присвоєно значення таким змінним, як `totalSupply`, `storagePath`, `publicPath` та `privatePath`. Як можна зрозуміти з назви `totalSupply`, це змінна, відповідальна за підрахунок всіх випущених документів (NFT). Змінні `privatePath`, `publicPath` та `storagePath`, містять в собі значення типу `path`, це особливий тип, що вказує на розташування ресурсу на акаунті. Більш детально про ці змінні та їх призначення буде описано у огляді створення, збереження та доступу колекцій в акаунті.

В блокчейні Flow колекція – це ресурс, який дозволяє його власнику керувати кількома об'єктами стекінгу в одному обліковому записі через єдиний шлях зберігання та виконувати дії стекінгу. В свою чергу ресурс (в мові програмування Cadence) – це складений тип, як структура або клас, але з певною відмінністю, яка грає ключову роль у контексті блокчейн мереж. На відміну від класів або структур, ресурс має чітко регламентовані операції копіювання, клонування або видалення. Тобто за дані операції відповідають окремі оператори або функції передбаченні розробником. Будь-які дії над колекцією або виклик будь-яких функцій колекції, здійснюється через транзакції, і є дійсним лише для акаунту, на якому зберігається колекція. Ресурс повинен мати свою адресу всередині акаунту, з метою індексації та доступу до нього, дана адреса ресурсу в акаунті називається шляхом.

Ресурс колекції оголошується в тілі смартконтракту та нагадує клас. Розглянемо оголошення ресурсу колекції у смартконтракті DocHub:

```

pub resource Collection:
    NonFungibleToken.Provider,
    NonFungibleToken.Receiver,
    NonFungibleToken.CollectionPublic,
    MetadataViews.ResolverCollection,

```

```

DocCollectionPublic
{
    pub var ownedNFTs: @{UInt64: NonFungibleToken.NFT}
}

```

З наведеного вище коду зрозуміло, що колекція представляє собою ресурс з назвою “Collection”, що в свою чергу наслідується від ряду інших абстрактних ресурсів, які в більшості належать зовнішнім смартконтрактам, які слугують стандартами блокчейну Flow, це зроблено з метою надання користувачам загально відомого інтерфейсу для взаємодії з даним ресурсом. Серед списку всіх ресурсів, що наслідуються слід виокремити DocCollectionPublic, який оголошений всередині смартконтракту, що розглядається, ціль якого є розширення списку публічних функцій clone даного ресурсу.

Окремої уваги потребують методи реалізовані в ресурсі Collection. Функція withdraw вилучає NFT із ресурсу, який його реалізує, і переміщує його до абонента, у нашому випадку NFT зберігається у змінній ownedNFTs, функція withdraw є частиною ресурсу NonFungibleToken.Provider. Функція deposit додає NFT до ресурсу, який його реалізує, дана функція є частиною ресурсу NonFungibleToken.Receiver. Функції getIDs, borrowNFT та borrowNFTSafe, є частиною ресурсу CollectionPublic, їх основна мета - повернення інформації про ресурс та його NFT. Функція getIDs відповідає за повернення цифрових ідентифікаторів NFT ресурсу, borrowNFT – повернення посилання на будь-яку NFT ресурсу, але у випадку відсутності останньої, завершується помилкою, borrowNFTSafe – повернення посилання на будь-яку NFT ресурсу, у разі відсутності останньої, поверне порожнє посилання. Окрім цього ресурс містить функцію ініціалізації та ряд функцій для відображення метаданих NFT даного ресурсу, наслідуванні від ресурсу MetadataViews.ResolverCollection.

Розглянемо також ресурс NFT. Даний ресурс також містить змінні та методи, що є необхідними для реалізації токенів, що містять дані про документ. Оголошення даного ресурсу виглядає наступним чином:

```

pub resource NFT:
    NonFungibleToken.INFT,
    MetadataViews.Resolver
{
    pub var history: [HistoryPoint]
    pub let id: UInt64
    pub var metadata: MetadataViews.NFTView
    pub var file: {MetadataViews.File}?
    pub var data: String
}

```

Як бачимо, ресурс NFT передбачає наступні змінні, такі як: `id`, `metadata`, `file`, `data` та `history`. Розглянемо їх більш детально. Константна змінна `id` містить загальний ідентифікатор NFT в смартконтракті та всередині ресурсу колекції. Змінна `metadata` призначена для збереження метаданих відображення, дана змінна має тип `MetadataViews.NFTView`, який передбачено смартконтратом `MetadataViews`, передбаченим блокчейном `Flow`, як стандарт. Змінна `file` та `data`, передбаченні для збереження даних про NFT документа. `file` є обов'язковим та містить дані про розташування зовнішнього файлу даної NFT. `data`, містить дані передбачені користувачем. Змінна `history` представляє собою масив з об'єктів типу `HistoryPoint` (даний тип буде розглянуто далі) і зберігає дані про попередні версії ресурсу даної NFT.

Наступним кроком є збереження історії версій поточного ресурсу. Це не обов'язковий крок, так як історію версій ми можемо отримати з блокчейн мережі у вигляді блоків, але для зручності доступу до інформації історії, версії слід передбачити цю змінну. В ресурсі NFT дана змінна має назву `history` і відповідає типу масив структур типу `HistoryPoint`, в смартконтракті `DocHub` вона має наступне оголошення:

```

pub struct HistoryPoint {
    pub let metadata: MetadataViews.NFTView
    pub let file: {MetadataViews.File}?
    pub let data: String
    init(

```

```

        metadata: MetadataViews.NFTView,
        file: {MetadataViews.File}?,
        data: String
    ){
        self.metadata = metadata
        self.file = file
        self.data = data
    }
}

```

Як можна помітити дана структура реалізує всі неконстантні змінні ресурсу NFT. Ці змінні містять значення відповідного ресурсу NFT до виклику функції оновлення. Також, як і будь-яка структура, вона містить функцію ініціалізації (конструктор), що повертає сутність даної структури з відповідно заповненими полями.

Також смартконтракт містить функцію створення порожньої колекції, `createEmptyCollection`. Дана функція передбачена для ініціалізації колекції у користувача, що немає відповідної колекції. Та функцію `mintDoc`, що відповідає за створення ресурсу NFT даного смартконтракту.

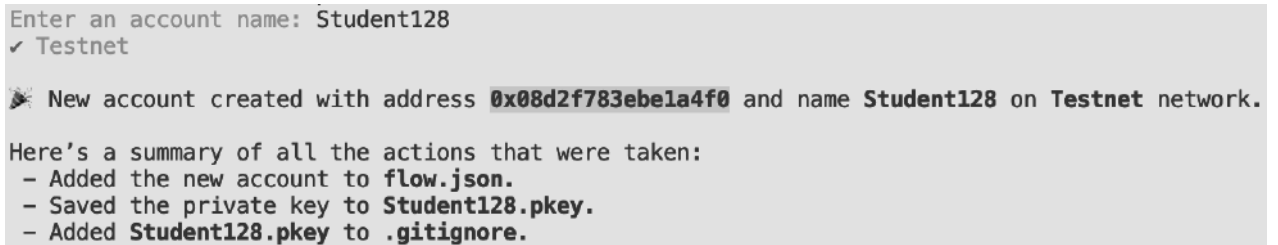
5.2 Огляд поведінки мережі блокчейн на транзакції мутацій

Для розгляду механізму мутації ресурсу в блокчейн мережі Flow, буде використаний смартконтракт `DocHub` в тестовій мережі за адресою “0xf8d6e0586b0a20c7” [13].

Першочергово необхідно мати акаунт в мережі блокчейн, у даному випадку буде створений новий акаунт за допомогою CLI інструменту блокчейну Flow. Далі наведена відповідна команда створення акаунту:

```
flow accounts create
```

після запуску команди, буде необхідно ввести локальне ім'я акаунту та обрати мережу, в якій ви бажаєте створити акаунт. В даному випадку було обрано ім'я “Student128”, мережа – Testnet, в результаті був отриманий гаманець за адресою “0x08d2f783eb1a4f0” (рисунок 5.1).



```

Enter an account name: Student128
✓ Testnet

🐙 New account created with address 0x08d2f783eb1a4f0 and name Student128 on Testnet network.

Here's a summary of all the actions that were taken:
- Added the new account to flow.json.
- Saved the private key to Student128.pkey.
- Added Student128.pkey to .gitignore.
  
```

Рисунок 5.1 – Результат виконання команди створення гаманця

Після створення гаманця та отримання його адреси, необхідно отримати певну кількість токенів, щоб здійснювати транзакції всередині мережі. Адже за виконання транзакцій кожен вузол має право стягувати платню у вигляді FT. Для отримання токенів, в тестовій мережі блокчейну Flow, можна скористатися сервісом FlowFaucet.

Наступним кроком у взаємодії зі смартконтрактом є створення його колекції в акаунті. Дана операція, є операцією запису в блокчейн, тому здійснюється через виклик операції типу транзакція. Далі буде наведений приклад транзакції:

```

transaction {
  let accoutn: AuthAccount
  prepare(acct: AuthAccount) {
    self.accoutn = acct
  }
  execute {
    let collection <- DocHub.createEmptyCollection()
    self.accoutn.save(<-collection, to: DocHub.storagePath)
    let collectionCapab =
      self.accoutn.link<{NonFungibleToken.Receiver}>(<
        DocHub.publicPath,
  
```

```

        target: DocHub.storagePath
    )
    ?? panic("Collection in storage not exist")
self.accountn.link<
    &{NonFungibleToken.Provider,
    NonFungibleToken.CollectionPublic,
    DocHub.DocCollectionPublic}
>(DocHub.privatePath, target: DocHub.storagePath)
}
}

```

На прикладі вищезазначеної транзакції можна прослідкувати, що транзакції мають вигляд подібний до структур. Головною частиною транзакції є конструкція `execute`, яка представляє собою головну функцію транзакції, саме в ній має відбуватися робота з ресурсами. Функція `prepare` є функцією, виклик якої здійснюється перед викликом функції `execute`. Головним призначенням даної функції є налаштування, присвоєння та обробка переданих змінних, єдиний параметр, який приймає дана функція, це інстан авторизованого акаунту, який здійснив підпис транзакції.

Transaction ID: 96d6a56091eb37cc208b538fd5b39c5ae5e90f8761368dcaa1e3f562665ae4c0	
Block ID	a447a10a873a191b2a9ddd8b0788e7c368287b0bed44a2a5f5ab2262cf484eba
Block Height	135574776
Status	☑ SEALED
ID	96d6a56091eb37cc208b538fd5b39c5ae5e90f8761368dcaa1e3f562665ae4c0
Payer	08d2f783ebe1a4f0
Authorizers	[08d2f783ebe1a4f0]
Proposal Key:	
Address	08d2f783ebe1a4f0
Index	0
Sequence	0

Рисунок 5.2 – Приклад результату виконання команди по виклику транзакції в блокчейні

Для здійснення транзакції в мережі блокчейну Flow, буде використовуватися програмний пакет Flow CLI. Даний програмний пакет

розроблений для можливості взаємодії користувача з мережею блокчейну Flow через командний рядок терміналу. Для виклику транзакції використовується наступна команда:

```
flow transactions send ./add-collection.tr.cdc --signer
Student128 --network testnet
```

де `./add-collection.tr.cdc` – це шлях до файлу, що містить код транзакції; `Student128` – це ім'я акаунту у файлі `flow.json`, який відповідальний за збереження всіх параметрів взаємодії з блокчейном; `testnet` – назва мережі, до якої буде надіслана транзакція. Результат виконання команди зображений на рисунку 5.2.

За результатом виконання команди “`flow transactions send`” ми бачимо, що кожна транзакція записується у відповідний блок, який містить всі відомості про транзакцію. Для перегляду всіх блоків та даних можна скористатися відкритими веб-ресурсами, такими як `flowdiver` [18], або використовувати Flow CLI. Flow CLI передбачає наступну команду для отримання блоку:

```
flow blocks get <block_id>
```

блок містить дані про транзакції, його вагу, час здійснення транзакції та статус його завершення. Щоб отримати вміст блоку “`a447a10a873a191b2a9ddd8b0788e7c368287b0bed44a2a5f5ab2262cf484eba`” включно з транзакціями скористаємось, наступною командою:

```
flow blocks get
a447a10a873a191b2a9ddd8b0788e7c368287b0bed44a2a5f5ab2262cf484eba
--network testnet --include transactions
```

Як бачимо, цей блок містить дві транзакції, викликані двома різними користувачами для двох різних колекцій, це можна спостерігати на рисунку 5.3 який відображає виконання команди в командному рядку, та на рисунку 5.4, що відображає вміст блоку у візуальній частині веб-застосунку “`flowdiver`”.

```

Block ID          a447a10a873a191b2a9ddd8b0788e7c368287b0bed44a2a5f5ab2262cf484eba
Parent ID         a55135e6505bd46355dd2538d95c9d41c75a4f39b5e1d3d0ecd809cf0556cd89
Proposal Timestamp 2023-11-27 14:31:53.261515408 +0000 UTC
Proposal Timestamp Unix 1701095513
Height            135574776
Status            Sealed
Total Seals        0
Total Collections  2
  Collection 0:     897d9ae4de64473a062dcabd2fa72dde1e232a0f0586134dae73f94ceeffb9a9
    Transaction 0:  b4b3f38d597280255983e093b60ee4b684488862f84aa9624e040cb43a22cf67
  Collection 1:     38c6765dcd3538008eef855bddc80dfe0db577e6b22fab965e0f829d5c059ec9
    Transaction 0:  96d6a56091eb37cc208b538fd5b39c5ae5e90f8761368dcaa1e3f562665ae4c0

```

Рисунок 5.3 – Приклад результату виконання команди “flow blocks get”, за допомогою Flow CLI

У випадку, коли необхідно отримати більш детальну інформацію про конкретну транзакцію у блокчейні Flow, отримати її можна, як скориставшись веб-застосунком flowdiver, так і використовуючи програмний пакет Flow CLI. У випадку використання пакету Flow CLI, команда на отримання транзакції буде виглядати наступним чином:

```
flow transactions get <tx_id> --include payload
```

прапорець include даної команди може приймати два значення “payload” та “code”. “payload” – це корисне навантаження транзакції, воно є внутрішньою частиною транзакції та містить дані, які однозначно ідентифікують операції, застосовані транзакцією.

Block Height: 135574776				
Transaction	Authorizers	Gas Used	Events	Status
b4b3f38d597280255983e093b60ee4b684488862f84aa9624e040cb43a22cf67 понеділок, 27 листопада 2023 р. о 16:31:53 GMT+2	0x28e3764cd5c5ead	37	1 event	SEALED
96d6a56091eb37cc208b538fd5b39c5ae5e90f8761368dcaa1e3f562665ae4c0 понеділок, 27 листопада 2023 р. о 16:31:53 GMT+2	0x08d2f783ebe1a4f0	21	0 events	SEALED

Рисунок 5.4 – Приклад відображення сутності блоку у веб-застосунку flowdiver

У Flow дві транзакції з однаковим корисним навантаженням ніколи не виконуватимуться більше одного разу. Корисне навантаження містить наступні дані: код транзакції, аргументи транзакції, викликані івенти, ліміт на

плату за комісію транзакції, дані про користувача, що викликав транзакцію та про платника комісії. Саме корисне навантаження транзакції, у разі успішного виконання, буде додано до ланцюжка і спираючись на ці дані, блокчейн буде робити висновки про вміст гаманця чи будь-якого іншого ресурсу.

```

Block ID      a447a10a873a191b2a9ddd8b0788e7c368287b0bed44a2a5f5ab2262cf484eba
Block Height  135574776
Status        ☒ SEALED
ID            96d6a56091eb37cc208b538fd5b39c5ae5e90f8761368dcaa1e3f562665ae4c0
Payer         08d2f783ebe1a4f0
Authorizers   [08d2f783ebe1a4f0]

Proposal Key:
  Address     08d2f783ebe1a4f0
  Index       0
  Sequence    0

No Payload Signatures

Envelope Signature 0: 08d2f783ebe1a4f0
Signatures (minimized, use --include signatures)

Events: None

Arguments     No arguments

Code

import NonFungibleToken from 0x631e88ae7f1d7c20
import MetadataViews from 0x631e88ae7f1d7c20
import DocHub from 0x88b76b3b47dec36d

transaction {
  let accoutn: AuthAccount

  prepare(acct: AuthAccount) {
    self.accoutn = acct
  }

  execute {
    let collection <-DocHub.createEmptyCollection()
    self.accoutn.save(<-collection, to: DocHub.storagePath)
    let collectionCapab =
      self.accoutn.link<&{NonFungibleToken.Receiver}>({
        DocHub.publicPath,
        target: DocHub.storagePath
      })
    ?? panic("Collection in storage not exist")
    self.accoutn.link<
      &{NonFungibleToken.Provider,
        NonFungibleToken.CollectionPublic,
        DocHub.DocCollectionPublic}
    >{DocHub.privatePath, target: DocHub.storagePath}
  }
}

```

Рисунок 5.5 – Приклад не перетвореного корисного навантаження транзакції
“a447a10a873a191b2a9ddd8b0788e7c368287b0bed44a2a5f5ab2262cf484eba”


```

transaction (docUrl: String, mediaUrl: String, data: String) {
  let collection: &{NonFungibleToken.Receiver}
  prepare(acct: AuthAccount) {
    self.collection = acct.getCapability<
      &{NonFungibleToken.Receiver}
    >(DocHub.publicPath)
      .borrow()
      ?? panic("Could not borrow receiver reference")
  }
  execute {
    let file = MetadataViews.HTTPFile(url: mediaUrl)
    let display = MetadataViews.Display(
      name: "second document",
      description: "upload for test",
      thumbnail: file
    )
    let fileOfDocument = MetadataViews.HTTPFile(url: docUrl)
    let metadata = MetadataViews.NFTView(
      id: DocHub.totalSupply,
      uuid: DocHub.totalSupply,
      display: display,
      externalURL: MetadataViews.ExternalURL(url: docUrl)
    )
    let newDoc <- DocHub.mintDoc(
      metadata: metadata,
      file: fileOfDocument,
      data: data
    )
    self.collection.deposit(token: <- newDoc)
  }
}

```

Виклик даної транзакції здійснюється аналогічно до попередньої, з використанням програмного пакету Flow CLI. Але має певні синтаксичні

відмінності, так як на відміну від попередньої, дана транзакція приймає параметри. Такі як “docUrl”, “mediaUrl” та “data”, ці змінні у випадку даного смартконтракту необхідні для заповнення даних ресурсу NFT. Синтаксис команди виклику транзакції з аргументами буде виглядати наступним чином:

```
flow transactions send ./tr/doc/mint-doc.tr.cdc 'docUrl'
'mediaUrl' 'data' --signer Student128 -network testnet.
```

Як можна прослідкувати з коду самої транзакції, вона відповідальна за створення ресурсу NFT та збереження даного ресурсу в колекцію користувача. Слід зазначити, що більшість смартконтрактів побудовані таким чином, що права на створення NFT прив’язаних до смартконтракту є лише у певної особи або групи осіб, це зумовлено специфікою ринку NFT, який в першу чергу націлений на розповсюдження та продаж NFT певного смартконтракту. Але у даному конкретному випадку мета даного смартконтракту є чітко регламентована та прозора - робота з документами, які в даному випадку виступають NFT, тому тут користувач має змогу створювати свої власні NFT. У цьому випадку команда виклику транзакції, матиме наступний вигляд:

```
flow transactions send ./tr/doc/mint-doc.tr.cdc
'https://drive.google.com/file/d/1YHyvIDcGUr90eBZT7soHPeynX8EZut-5/view?usp=sharing'
'https://ipfs.io/ipfs/bafybeidcjjk6yr645njktgbk3f6dfrlm7gynu2mohvfz3d
5tmyqwanfru/%D0%97%D0%BD%D1%96%D0%BC%D0%BE%D0%BA%20%D0%B5%D0%BA%D1%80%
D0%B0%D0%BD%D0%B0%202023-11-28%20%D0%BE%2013.08.55.png' '{
    "title": "Document #1",
    "description": "Test document for report",
    "data": "Data about report ..."
}' --signer Student128 --network testnet
```

Перший параметр даної транзакції – рядок, який є посиланням на зовнішнє джерело документу, що буде збережений. У конкретному випадку – це текстовий документ, що розміщується на хмарному середовищі Google Drive. Другий параметр – це посилання на файл в розподіленій системі

збереження файлів IPFS, дана система представляє собою розподілене між багатьма користувачами сховище файлів, в якому частини файлів можуть зберігатися у різних користувачів, а їх пошук здійснюється по унікальному хеш ідентифікатору. Третім параметром, є вміст файлу, в даному випадку вміст – це JSON структура, яка містить такі поля як "title", "description" та "data" (даний вміст є лише прикладом. Поле, дата може зберігати будь-яке текстове представлення файлу).

```
Transaction ID: 0fe16af8574a8b559d052ffeffcfff64f9d339037332698a585ffdc23a9208143

Block ID      7be1867329c1513c16e42152f76c7412267b3198286f291b21b12319868e85cc
Block Height  135880218
Status       ☒ SEALED
ID           0fe16af8574a8b559d052ffeffcfff64f9d339037332698a585ffdc23a9208143
Payer        08d2f783ebe1a4f0
Authorizers   [08d2f783ebe1a4f0]

Proposal Key:
  Address     08d2f783ebe1a4f0
  Index       0
  Sequence    26

No Payload Signatures

Envelope Signature 0: 08d2f783ebe1a4f0
Signatures (minimized, use --include signatures)

Events:
  Index       0
  Type        A.88b76b3b47dec36d.DocHub.NFTMinted
  Tx ID       0fe16af8574a8b559d052ffeffcfff64f9d339037332698a585ffdc23a9208143
  Values
    - id (UInt64): 13
    - setId (Int): 1
    - templateId (Int): 1
    - serial (UInt64): 13
```

Рисунок 5.7 – Приклад виконання транзакції на створення NFT

Результатом виконання вище згаданої транзакції буде створення відповідного NFT (рисунок 5.7), що буде переміщена до відповідної колекції користувача, що викликав дану транзакцію. Також, як і попередня транзакція, дана має у собі ідентифікатор блока до котрого вона належить, власний ідентифікатор, корисне навантаження, котре в свою чергу містить: підпис, код транзакції, аргументи виклику, тощо. Створену NFT можна переглянути через сервіс flowview або через відповідний запит у блокчейн (рисунок 5.8).

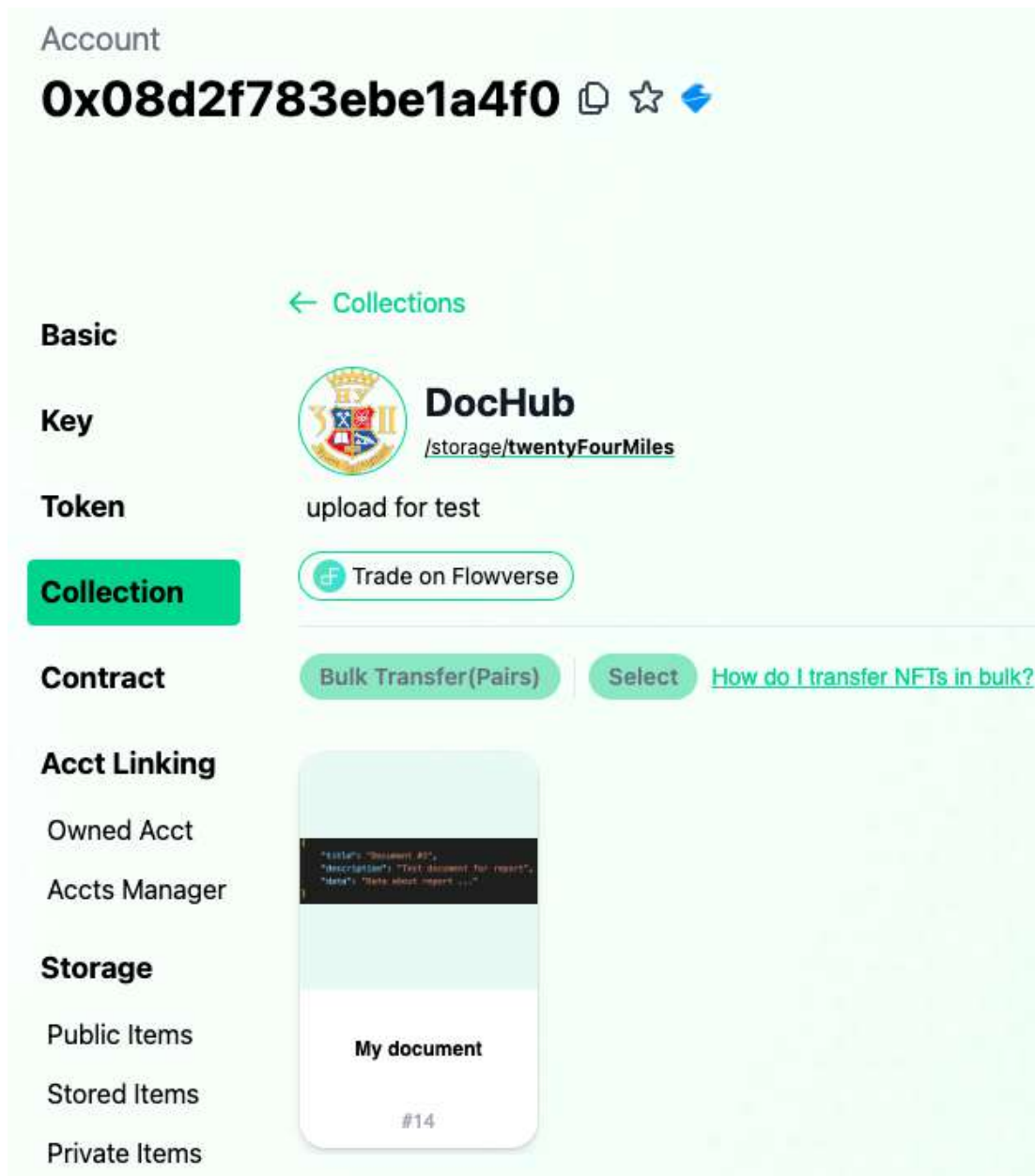


Рисунок 5.8 – Приклад відображення NFT у веб-застосунку flowview

Формування запитів у блокчейн подібне до формування транзакцій. Дані запити у блокчейн мережі Flow називають сценаріями (scripts). Далі наведено приклад сценарію, що має отримати посилання на NFT у блокчейн мережі та повернути дані, що містяться у NFT:

```
pub fun main(address: Address, id: UInt64):
  &NonFungibleToken.NFT? {
    let acc = getAccount(address)
    let collection = acc.getCapability<
      &{NonFungibleToken.CollectionPublic}
```

```

>(DocHub.publickPath)
.borrow()
?? panic("Could not borrow receiver reference")
return collection.borrowNFTSafe(id: id)
}

```

Подібно до таких мов програмування як C++ та C#, сценарії мають головну функцію виклику `main`, яка приймає параметри та здійснює повернення результуючих даних сценарію. В даному випадку присутні два параметри “address” та “id”. “address” – це адреса гаманця/акаунту у мережі блокчейн, до якого ми будемо посилатися для перегляду NFT. “id” – це загальний ідентифікатор NFT у смартконтракті. Для акаунту `0x08d2f783ebe1a4f0` та NFT з ідентифікатором рівному 14, виклик сценарію буде виглядати наступним чином:

```

flow scripts execute ./scripts/doc.cdc '0x08d2f783ebe1a4f0' 14
--network testnet

```

```

Result: A.88b76b3b47dec36d.DocHub.NFT(uuid: 180111673, history: [], id: 14, metadata: A.631e88ae7f1d7c20.MetadataViews.NFTView(id: 14, uuid: 14, display: A.631e88ae7f1d7c20.MetadataViews.Display(name: "My document", description: "{\n  \"title\": \"Document #1\", \n  \"description\": \"Test document for report\", \n  \"data\": \"Data about report ...\", \n}"), thumbnail: A.631e88ae7f1d7c20.MetadataViews.HTTPFile(url: "https://ipfs.io/ipfs/bafybeidcjjk6yr645njktgbk3f6dfrlm7gynu2mohvfz3d5tmyqwanfru/%D0%97%D0%BD%D1%96%D0%BC%D0%BE%D0%BA%20%D0%B5%D0%BA%D1%80%D0%B0%D0%BD%D0%B0%202023-11-28%20%D0%BE%2013.08.55.png")), externalURL: A.631e88ae7f1d7c20.MetadataViews.ExternalURL(url: "https://drive.google.com/file/d/1YHyvIDcGUr90eBZT7soHPeynX8EZut-5/view?usp=sharing"), collectionData: nil, collectionDisplay: nil, royalties: nil, traits: nil), file: A.631e88ae7f1d7c20.MetadataViews.HTTPFile(url: "https://drive.google.com/file/d/1YHyvIDcGUr90eBZT7soHPeynX8EZut-5/view?usp=sharing"), data: "{\n  \"title\": \"Document #1\", \n  \"description\": \"Test document for report\", \n  \"data\": \"Data about report ...\", \n}"))

```

Рисунок 5.9 – Приклад результату виконання сценарію блокчейн мережею

Як можна спостерігати на рисунку 5.9, відповідь сценарію містить усі дані про шукану NFT. Можна спостерігати як `uuid`, який є загальним ідентифікатором ресурсу у блокчейн мережі Flow, зміну `history` яка розглядалася в попередньому розділі, `metadat` поточної NFT, та інша інформація, що відповідає даним NFT включно з усіма посиланнями.

Тепер розглянемо випадок, коли існує необхідність змінити дані всередині ресурсу, тобто провести операцію з мутації даних.

Дана операція буде здійснюватися також через транзакцію та заздалегідь створений метод update ресурсу NFT. Далі буде наведено приклад транзакції, що використовує метод update ресурсу NFT смартконтракту DocHub:

```
transaction (id: UInt64, fileUrl: String) {
  let collection: &{DocHub.DocCollectionPublic}
  prepare(acct: AuthAccount) {
    self.collection = acct.getCapability<
      &{DocHub.DocCollectionPublic}
    >(DocHub.privatePath)
      .borrow()
  }
  execute {
    let nft = self.collection.myNFT(id: id)
    ?? panic("Can't cast data")
    let file = MetadataViews.ExternalURL(url: fileUrl)
    let metadata = MetadataViews.NFTView(
      id: nft.metadata.id,
      uuid: nft.metadata.uuid,
      display: nft.metadata.display,
      externalURL: file,
      collectionData: nft.metadata.collectionData,
      collectionDisplay: nft.metadata.collectionDisplay,
      royalties: nft.metadata.royalties,
      traits: nft.metadata.traits
    )
    nft.update(report: DocHub.HistoryPoint(
      metadata: metadata,
      file: nft.file,
      data: nft.data
    ))
  }
}
```

Як можна спостерігати з вищенаведеного коду транзакції, у основному тілі транзакції (тобто в `execute`), спочатку здійснюється виклик методу `myNFT` ресурсу колекції, даний метод повертає посилання на ресурс NFT даного смартконтракту, що належить поточній колекції. Після ініціалізації та присвоєння значень змінним `“file”` та `“metadata”`, здійснюється виклик методу `update` поточного ресурсу NFT. У даному методі відбувається безпосередня мутація вмісту даного ресурсу та доповнення ресурсу.

```
Transaction ID: 2a33d9eecd5a207d9e54269b7b5990554d0eaa9e47c44add7fc8b63c414a0266

Block ID      95e321724816bc76b9a089d4551b619e2ea2ae33680d84505c8622db7d7a1c01
Block Height  136039943
Status       ☒ SEALED
ID           2a33d9eecd5a207d9e54269b7b5990554d0eaa9e47c44add7fc8b63c414a0266
Payer        08d2f783ebe1a4f0
Authorizers   [08d2f783ebe1a4f0]

Proposal Key:
  Address     08d2f783ebe1a4f0
  Index       0
  Sequence    40

No Payload Signatures

Envelope Signature 0: 08d2f783ebe1a4f0
Signatures (minimized, use --include signatures)

Events: None
```

Рисунок 5.10 – Приклад результату виконання транзакції на мутацію даних

Щоб переглянути корисне навантаження даної транзакції, буде використана, раніше згадана, команда `“flow transactions get”`. Рисунок 5.10 відображає вміст корисного навантаження для транзакції з ідентифікатором `“2a33d9eecd5a207d9e54269b7b5990554d0eaa9e47c44add7fc8b63c414a0266”`.

Суть виконання, вище згаданої, транзакції з мутації є зміна зовнішнього посилання на файл. Як можна помітити, зміни зовнішнього посилання відбулися як у змінній `metadata`, так і в змінній `file` ресурсу NFT. Результати зміни поля `metadata` можна спостерігати через веб-застосунок `flowview`, у якому відповідне значення відображається у правому нижньому куті зображення (рисунок 5.12). Як бачимо, дане посилання змінилося з

“https://drive.google.com/file/d/1YHyvIDcGUr90eBZT7soHPeynX8EZut-5/view?usp=sharing”, на –
 “https://drive.google.com/file/d/1nXVTbft60ixl5c33j-EsqRKyt2Y5zPxL/view?usp=sharing”. Результат мутації та зміни поля можна спостерігати на рисунку 5.13.

```
Block ID      95e321724816bc76b9a089d4551b619e2ea2ae33680d84505c8622db7d7a1c01
Block Height 136039943
Status       ☒ SEALED
ID           2a33d9eecd5a207d9e54269b7b5990554d0eaa9e47c44add7fc8b63c414a0266
Payer        08d2f783ebe1a4f0
Authorizers  [08d2f783ebe1a4f0]

Proposal Key:
Address      08d2f783ebe1a4f0
Index        0
Sequence     40

No Payload Signatures

Envelope Signature 0: 08d2f783ebe1a4f0
Signatures (minimized, use --include signatures)

Events: None

Arguments (2):
- Argument 0: {"value":"14","type":"UInt64"}
- Argument 1: {"value":"https://drive.google.com/file/d/1nXVTbft60ixl5c33j-EsqRKyt2Y5zPxL/view?usp=sharing","type":"String"}

Code

import NonFungibleToken from 0x631e88ae7fd7c20
import MetadataViews from 0x631e88ae7fd7c20
import DocHub from 0x88b76b3b47dec36d

transaction (id: UInt64, fileUrl: String) {
  let collection: &{DocHub.DocCollectionPublic}

  prepare(acct: AuthAccount) {
    self.collection = acct.getCapability<
      &{DocHub.DocCollectionPublic}
    >{DocHub.privatePath}
    .borrow()
    ?? panic("Could not borrow receiver reference")
  }
  execute {
    let nft = self.collection.myNFT(id: id)
    ?? panic("Can't cast data")
    let file = MetadataViews.ExternalURL(url: fileUrl)
    let metadata = MetadataViews.NFTView(
      id: nft.metadata.id,
      uuid: nft.metadata.uuid,
      display: nft.metadata.display,
      externalURL: file,
      collectionData: nft.metadata.collectionData,
      collectionDisplay: nft.metadata.collectionDisplay,
      royalties: nft.metadata.royalties,
      traits: nft.metadata.traits
    )
    nft.update(report: DocHub.HistoryPoint(
      metadata: metadata,
      file: nft.file,

```

Рисунок 5.11 – Корисне навантаження транзакції

“2a33d9eecd5a207d9e54269b7b5990554d0eaa9e47c44add7fc8b63c414a0266”

Щоб переглянути поточний вміст ресурсу NFT, що є прихованим для веб-застосунку flowview, є можливість скористатися наведеним вище сценарієм для перегляду вмісту шуканого ресурсу NFT, відповідного гаманця.

В даному випадку спостерігається реалізація механізму мутації даних в блокчейні на рівні смартконтракту та протоколу. Спостерігаючи за реакцією блокчейну на операцію мутації, можна констатувати, що цей механізм є реалізацією метода без унікального покажчика та з заміною блоку на поточний. Даний метод є найшвидшим представником механізмів мутації даних, але в той же час він програє іншим рішенням в порівнянні розмірів самої мережі та в ергономічності для користувачів з підтвердженням дійсності історії зміни ресурсів в блокчейні.

5.3 Застосування механізму мутабельності на рівні застосунку

Даний розділ буде зосереджений на відтворенні механізму мутабельності блокчейну на рівні застосунку. Для даної реалізації буде розроблений застосунок з використання програмного пакету NodeJS, з застосування відкритої бібліотеки для взаємодії з блокчейном “@onflow/fcl” [15]. Також буде розглянутий смартконтракт DocHubV2 за адресою “0x88b76b3b47dec36d” [14], розроблений спеціально для демонстрації даної реалізації механізму мутації даних на рівні застосунку.

Слід зазначити головні відмінності смартконтракту DocHubV2 від раніше розглянутого смартконтракту DocHub. В більшій мірі вони однакові, але у смартконтракті DocHubV2 для ресурсу NFT була ліквідована функція update, та додана нова функція addVersion. Також для даного ресурсу було прибрано поле “history” та додано нове – “newInfo”. Дані зміни зумовлені специфікацією даної реалізації, їх роль буде описана далі.

Після відповідних змін ресурс NFT смартконтракту DocHubV2, має наступний вигляд:

```
pub resource NFT
```

```

{
    pub let id: UInt64
    pub var newInfo: UInt64?
    pub let metadata: MetadataViews.NFTView
    pub let file: {MetadataViews.File}?
    pub let data: String
    pub let date: UFix64?
    pub fun getViews(): [Type] {
        return [Type<MetadataViews.NFTView>()]
    }
    pub fun resolveView(_ view: Type): AnyStruct? {
        let data = self.history[self.history.length - 1]
        switch view {
            ...
        }
        return nil
    }
    pub fun addVersion(report: HistoryPoint){
        self.history.append(report)
    }
    init(
        id: UInt64,
        metadata: MetadataViews.NFTView,
        file: {MetadataViews.File}?,
        data: String
    ){
        self.id = id
        self.history = [HistoryPoint (metadata, file, data)]
    }
}

```

Як видно з вищенаведеного коду ресурсу NFT, його методи “resolveView”, “addVersion” та “init” зазнали змін відповідно до потреб даного смартконтракту та механізму мутації, що буде розглянутий. У іншому структурі смартконтракту однакові.

Як було вказано вище, для даної реалізації буде використовуватись програмний апарат NodeJS. З використанням мови програмування TypeScript та пакету (бібліотеки) “@onflow/fcl” на стороні сервера. Для реалізації візуальної частини, буде використана мова розмітки HTML, препроцесор CSS та мова програмування JavaScript.

Для розробки серверної частини застосунку був використаний фреймворк Express. Взаємодія користувача з системою здійснюється через HTTP протокол.

Для здійснення операцій з гаманцем та смартконтрактом необхідно здійснити операцію аутентифікації гаманця. Маршрут “http://localhost:3000/auth” відповідає за отримання візуальної частини інтерфейсу аутентифікації (візуальне представлення даного інтерфейсу зображено на рисунку 5.14). По натисканню кнопки “Log In”, буде здійснено запит на створення сховку даних про гаманець, даний запит знаходиться за адресою “http://localhost:3000/auth/api/login”.



Auth in the app

Address

Secret Key

Log In

Рисунок 5.14 – Візуальне представлення сторінки авторизації акаунту в блокчейні

Після аутентифікації гаманця буде здійснена перевірка наявності у даного користувача відповідної колекції. Код сценарію для перевірки наявності колекції у гаманця наведено далі:

```
import NonFungibleToken from 0x631e88ae7f1d7c20
```

```

import MetadataViews from 0x631e88ae7f1d7c20
import DocHubV2 from 0x88b76b3b47dec36d
pub fun main(address: Address):
    &{NonFungibleToken.CollectionPublic}?
{
    let acc = getAccount(address)
    let collection = acc.getCapability<
        &{NonFungibleToken.CollectionPublic}
    >(DocHubV2.publicPath).borrow()
    return collection
}

```

Слід звернути увагу, що в даному випадку імпортування смартконтрактів здійснюється не з локально розміщених файлів, а на пряму з блокчейну. Це зумовлено тим, що виклик даного смартконтракту буде здійснено не з локального середовища Flow CLI, а на пряму в блокчейні.

Як було зазначено вище, для здійснення надсилання сценаріїв та транзакцій до блокчейну буде використаний пакет “@onflow/fcl”. Для використання даного пакету треба спочатку здійснити його ініціалізацію, далі наведений приклад коду ініціалізації даного пакету:

```

fcl
    .config({"app.detail.title": "DocHubV2"})
    .put("accessNode.api", "https://rest-testnet.onflow.org")
    .put(
        "discovery.wallet",
        "https://staging.accounts.meetdapper.com/fcl/authn-restricted"
    );

```

з вищенаведеного кода можна спостерігати, що для ініціалізації з’єднання до блокчейну через даний пакет, необхідно вказати адрес вузла блокчейну, до якого буде здійснено підключення (“accessNode.api”), ім’я застосунку (“app.detail.title”, є не обов’язковим полем), та сервіс перевірки гаманців (“discovery.wallet”). Для надсилання сценаріїв до блокчейну, використовується наступний код:

```
await fcl.query({
  cadence: scriptText,
  args: (arg, t) => [arg(address, t.Address)],
});
```

Для надсилання сценарію на певний вузол мережі необхідно вказати такі параметри як: “cadence”, що є тестовим представленням коду сценарію, та “args” – масив аргументів для виконання сценарію. Кожен елемент, що входить до параметру “args”, має оброблятися спеціальною функцією “arg” для перетворення передаваних даних до вигляду, що буде сприйнятий блокчейн мережею.

У результаті виклику сценарію ми отримаємо відповідь у вигляді значення null, якщо колекція у аутентифікованого гаманця відсутня, або об’єкт JSON представлення колекції. У випадку відсутності колекції буде здійснена транзакція на створення останньої.

Синтаксис виклику транзакцій схожий на аналогічний синтаксис виклику сценаріїв, але здійснюється з використанням методу “mutate” замість “query”. Приклад коду, що надсилає транзакцію наведений нижче:

```
await fcl.mutate({
  cadence: transactionText,
  args: (arg, t) => [
    arg(data.name, t.String),
    arg(data.file, t.String),
    arg(data.image, t.String),
    arg(data.data, t.String),
  ],
});
```

Гаманець, що здійснює надсилання транзакції має бути заздалегідь встановленим через використання методу “authenticate”.

Після закінчення процесу аутентифікації, та створення колекції, у разі її відсутності на гаманці. Застосунок здійснює перенаправлення користувача за адресою “http://localhost:3000/collection” . На даній сторінці можна спостерігати два відокремлені блокіс (рисунок 5.15). В блоці, що знаходиться

зліва, представлена форма для створення нового документу (NFT) у блокчейні. Праворуч – форма для пошуку NFT у даній колекції поточного гаманця.

Create document

Get document

Рисунок 5.15 – Візуальне відображення сторінки створення та пошуку NFT

Як можна спостерігати, форма створення нового документу включає в себе такі поля, як “name”, “fileUrl”, “imageUrl” та “data”, значення та роль цих полів еквівалентна аналогічним полям з ресурсу NFT смартконтракту DocHubV2.

Create document

Create Document Successfully

Рисунок 5.16 – Вигляд форми створення документу в блокчейні після успішного запиту

Заповнивши поля відповідними значеннями, ми маємо змогу здійснити виклик кінцевої точки за адресою “<http://localhost:3000/collection/api/create>”. У результаті успішного виконання запиту форма прийме відповідний вигляд до рисунку 5.16. Перевірка успішності виконання даного запиту, може бути здійснена за допомогою веб-застосунку flowview, на рисунку 5.17 зображено вміст даної колекції на даному веб-застосунку.

Як ми бачимо з рисунку 5.17, що створений NFT має “id” рівний 0. Якщо відкрити даний ресурс, можна побачити, що всі дані відповідають даним, введеним при створенні даного ресурсу. Знаючи ідентифікатор даного ресурсу, можна скористатися формою для пошуку ресурсу NFT за його ідентифікатором.

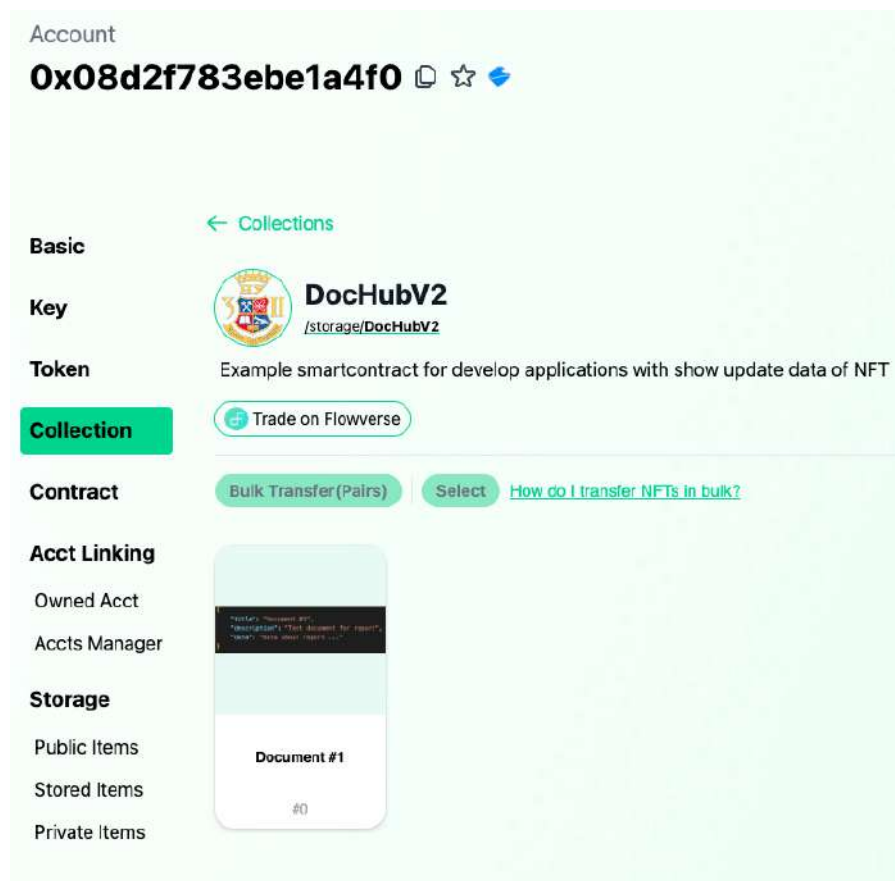


Рисунок 5.17 – Вміст колекції смартконтракту DocHubV2 гаманця
“0x08d2f783ebe1a4f0” у веб-застосунку flowview

Перейшовши до форми, що знаходиться справа, передавши у поле вводу ідентифікатор шуканого ресурсу NFT, з'явиться змога здійснити запит на кінцеву точку “http://localhost:3000/collection/api/find”. На стороні сервера використовується сценарій пошуку ресурсу NFT, який у разі успіху поверне JSON представлення ресурсу, а у випадку невдачі null, також у випадку невдачі, візуальна частина сервісу відобразить повідомлення про помилку. У разі успіху, застосунок здійснить перенаправлення на сторінку документу, за адресою “http://localhost:3000/document/api/:id”.

На рисунку 5.18 зображене візуальне представлення сторінки документу за адресою “http://localhost:3000/document/0”. Як можна побачити, сторінка відображає значення змінних ресурсу NFT, такі як ім'я, дані, посилання на зовнішній файл та посилання на картинку документу, а також візуалізує дану картинку.



Рисунок 5.18 – Візуальне представлення сторінки документу з ідентифікатором 0

У випадку зміни даних у будь-якому полі, буде наявна можливість здійснити запит на оновлення інформації. При взаємодії з кнопкою “Update”, відбудеться запит на кінцеву точку “http://localhost:3000/document/api/:docId”, даний запит буде містити корисне навантаження, необхідне для оновлення даних про поточний ресурс.

Так як даний застосунок реалізує метод мутації з заміною та з покажчиком на батьківський блок, то оновленні дані будуть збережені як окремий ресурс, який буде містити маркування на батьківський блок. За зберігання інформації про батьківський блок в даному випадку відповідає змінна “newInfo” ресурсу NFT. Спостерігати результат виконання даної транзакції можна знову завдяки веб-застосунку flowview.

Як видно з рисунку 5.19, наразі колекція налічує два ресурса NFT. Ресурс, що має ідентифікатор 3, містить оновлені дані для ресурсу з ідентифікатором 1, а також містить покажчик на відповідний ресурс. Відповідно до покажчика, ми можемо сформулювати алгоритм повертання даних. Далі наведений приклад функції, що обробляє запит на блок, та здійснює пошук ресурсів з покажчиком на шуканий блок:

```
export const getDocument = async (
  userId: string,
  docId: string
): Promise<IDocNFT> => {
  const scriptText = await readFile(scriptPath);
  const defaultJson: IDefaultAccountJSON = JSON.parse(
    await readFile(`../../flow/configs/${userId}.json`)
  );

  const address =
    `0x${defaultJson?.accounts?.[userId]?.address}`;

  let nft: IDocNFT = await fcl.query({
    cadence: scriptText,
    args: (arg, t) => [arg(address, t.Address), arg(docId, t.UInt64)]
  });

  while (nft?.newInfo)
    nft = await fcl.query({
      cadence: scriptText,
      args: (arg, t) => [
        arg(address, t.Address),
        arg(`${nft?.newInfo}`, t.UInt64),
      ],
    });
}
```

```
});
return nft;
};
```

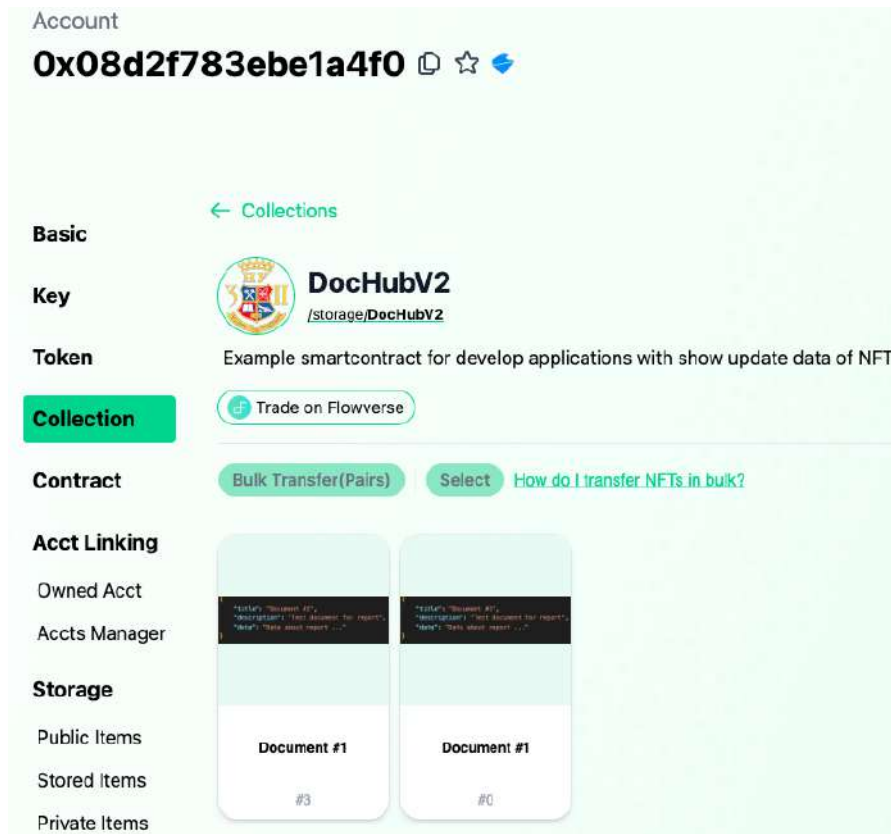


Рисунок 5.19 – Вміст колекції смартконтракту DocHubV2 гаманця “0x08d2f783ebe1a4f0” у веб-застосунку flowview

Якщо повторно здійснити запит за адресою “http://localhost:3000/document/0”, можна простежити зміну даних, що відображаються для вмісту ресурсу NFT. На рисунку 5.20 наведений приклад відображення даних з для ресурсу з ідентифікатором 0, після транзакції на оновлення даних для даного ресурсу.

Механізм імітації мутабельності даних в блокчейні, який з заміною та з показником на батьківський блок спрощує пошук пов’язаних змін. Реалізація даного метода на рівні застосунку також прибирає ряд ризиків, таких як: можливість вносити зміни в ресурс, який зберігається в блокчейні; можливі помилки в реалізації смартконтракту, тощо.

```
{
  "title": "Document #1",
  "description": "Test document for report",
  "data": "Data about report ..."
}
```

Data

Document #1

https://pfs.io/pfs/balybeicbbjyhw4bdlr4ocr68fgqv2xguf38naeyxrwod83x44b3bylNLJ2P.png

https://pfs.io/pfs/balybeicbjk6yr645njkgxk3f5-dfrim7gymu2mohvz3d56myqwanfu%KDO%97%DO'

{ "title": "Document #1", "description": "Test document for report", "data": "Data about report..." }

Рисунок 5.20 – Візуальне відображення даних для ресурсу NFT, після транзакції на оновлення даних для даного ресурсу

Але слід зазначити, що такій реалізації притаманні вразливості звичайних застосунків такі, як атака у відмові в доступі, атака людина посередині, тощо.

ВИСНОВКИ

Мутабельність – це важлива властивість інформації. Більшість систем потребують реалізації даної властивості для роботи з даними, внесення відповідних коректив, виправлення помилок, тощо. Реалізація даної властивості інформації в блокчейні є необхідним інструментом, що дозволило б цій технології зайняти своє місце у ще більшій кількості сфер людського життя. Хоч мутабельність, як властивість інформації, протирічить концепції блокчейну, це не є перепорою. Існує ряд семантичних методів, які дозволяють здійснювати імітацію мутабельності даних.

Хоч дані методи є лише доповненнями, які імітують дану властивість інформації, це все одно збільшує спектр загроз.

Аналізуючи загрози, описані у роботі, можна зробити висновок, що хоч мутабельність і є корисним та необхідним інструментом, вона також приносить з собою ряд потенційних загроз для захищеності блокчейну. Проте важливо враховувати, що загрози можуть відрізнятися в залежності від методу, який був використаний для реалізації функціоналу мутації даних.

Один з можливих підходів до забезпечення мутабельності даних може мати переваги над класичною реалізацією блокчейну. Наприклад, використання мутування смартконтрактів дозволяє гнучко налаштовувати правила зміни даних, що може приносити певні переваги у визначених випадках. Однак, цей підхід також може створювати нові вразливості та ризики, які потребують уважного аналізу та заходів запобігання ризиків.

Розглянувши можливі методи реалізації мутабельності даних у блокчейні на практиці, можна зазначити, що важливо розробляти та використовувати механізми, які дозволяють контролювати та обмежувати можливості смартконтрактів і функціоналу мутації даних. Це включає в себе ретельну перевірку коду, врахування можливих сценаріїв зловживання та

розробку ефективних механізмів виявлення та реагування на потенційні атаки.

Узагальнюючи, мутабельність даних в блокчейні має як переваги, так і недоліки. Вона може забезпечити гнучкість, розширені можливості, пришвидчити роботу блокчейну та позитивно вплинути на динаміку розширення мережі. Проте водночас вносить певні загрози для безпеки. Важливо ретельно аналізувати ці загрози та розробляти відповідні заходи протидії ризикам для забезпечення надійності та захищеності системи блокчейну. Такі як наприклад контроль вмісту смартконтрактів. З огляду на вищезазначене, важливо визначити, що розвиток ефективних стратегій управління мутабельністю даних у блокчейні є критичним завданням для подальшого просування технології. Шляхом вдосконалення механізмів контролю та обмеження смартконтрактів можна створити більш безпечні умови для роботи з даними в блокчейні.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Melanie Swan. Blockchain: Blueprint for a New Economy. — 2015. — 152 p. (дата звернення 14.09.2023)
2. GDPR та блокчейн: поєднати неможливо – Електронний ресурс – Режим доступу: <https://legalitgroup.com/gdpr-ta-blokchejn-poyednati-neroyednane> (дата звернення 18.10.2023)
3. Blockchain – Електронний ресурс – Режим доступу: <https://www.it.ua/knowledge-base/technology-innovation/blockchain> (дата звернення 19.10.2023)
4. The block structure of Bitcoin blockchain – Електронний ресурс – Режим доступу: https://www.researchgate.net/figure/The-block-structure-of-Bitcoin-blockchain_fig2_341189041 (дата звернення 20.10.2023)
5. Імутабельність – Електронний ресурс – Режим доступу: <https://habr.com/ru/companies/developersoft/articles/302118/> (дата звернення 20.10.2023)
6. Ethereum Full Node Sync (Archive) – Електронний ресурс – Режим доступу: <https://etherscan.io/chartsync/chainarchive> (дата звернення 21.10.2023)
7. Flow statistic – Електронний ресурс – Режим доступу: <https://flowverse.co> (дата звернення 22.10.2023)
8. Bitcoin: A Peer-to-Peer Electronic Cash System – Електронний ресурс – Режим доступу: <https://web.archive.org/web/20140320135003/https://bitcoin.org/bitcoin.pdf> (дата звернення 23.10.2023)
9. Загрози в мережі Blockchain – Електронний ресурс – Режим доступу: <https://openarchive.nure.ua/server/api/core/bitstreams/a58dba8c-0bad-4f2c-8648-70d8a1cd6a3d/content> (дата звернення 24.10.2023)
10. Flow (blockchain)(review) – Електронний ресурс – Режим доступу: [https://golden.com/wiki/Flow_\(blockchain\)-AMZMANP](https://golden.com/wiki/Flow_(blockchain)-AMZMANP) (дата звернення 30.10.2023)

11. Ethereum – Електронний ресурс – Режим доступу: <https://ethereum.org/en/> (дата звернення 1.11.2023)
12. Flow – Електронний ресурс – Режим доступу: <https://flow.com/>
13. “DocHub Smartcontract” DNZ – Електронний ресурс – Режим доступу: <https://f.dnz.dev/0x88b76b3b47dec36d/DocHub> (дата звернення 2.11.2023)
14. “DocHubV2 Smartcontract” DNZ – Електронний ресурс – Режим доступу: <https://f.dnz.dev/0x88b76b3b47dec36d/DocHubV2> (дата звернення 3.11.2023)
15. “Flow CLI” Flow Developer – Електронний ресурс – Режим доступу: <https://developers.flow.com/build/flow> (дата звернення 5.11.2023)
16. “Resource Contract Tutorial” Cadence – Електронний ресурс – Режим доступу: <https://cadence-lang.org/docs/tutorial/resources> (дата звернення 8.11.2023)
17. “Authorization Function” GitHub – Електронний ресурс – Режим доступу: <https://github.com/onflow/fcl-js/blob/master/packages/fcl/src/wallet-provider-spec/authorization-function.md> (дата звернення 9.11.2023)
18. “Flow Blockchain Analytics” Flowdiver – Електронний ресурс – Режим доступу: <https://www.flowdiver.io/analytics> (дата звернення 10.11.2023)
19. “Transaction” Flowdiver – Електронний ресурс – Режим доступу: <https://testnet.flowdiver.io/tx/2a33d9eecd5a207d9e54269b7b5990554d0eaa9e47c44add7fc8b63c414a0266?tab=overview> (дата звернення 10.11.2023)
20. “Welcome to the IPFS docs” IPFS – Електронний ресурс – Режим доступу: <https://docs.ipfs.tech/> (дата звернення 10.11.2023)
21. “Flow Client Library ” Flow Developer – Електронний ресурс – Режим доступу: <https://docs.ipfs.tech/> (дата звернення 10.11.2023)
22. Головка І.О., Козіна Г.Л. Аналіз впливу мутабельності даних в блокчейні на захищеність. Тиждень науки-2023. Факультет радіоелектроніки та телекомунікацій. Тези доповідей науково-практичної конференції, Запоріжжя, 24–28 квітня 2023 р. С. 111-113 (дата звернення 8.11.2023)

ДОДАТОК А

Код реалізації смарт-контракту з можливістю мутації ресурсу

doc.contract.cdc

```
import NonFungibleToken from 0x631e88ae7f1d7c20
import MetadataViews from 0x631e88ae7f1d7c20
pub contract DocHub: NonFungibleToken {
  pub var totalSupply: UInt64
  pub let storagePath: StoragePath
  pub let publicPath: PublicPath
  pub let privatePath: PrivatePath
  pub struct HistoryPoint {
    pub let metadata: MetadataViews.NFTView
    pub let file: {MetadataViews.File}?
    pub let data: String
    init(
      metadata: MetadataViews.NFTView,
      file: {MetadataViews.File}?,
      data: String
    ){
      self.metadata = metadata
      self.file = file
      self.data = data
    }
  }
  pub resource interface INFT {
    pub var history: [HistoryPoint]
    pub fun update(report: HistoryPoint)
    pub var metadata: MetadataViews.NFTView
    pub var file: {MetadataViews.File}?
    pub var data: String
  }
  pub resource NFT:
    NonFungibleToken.INFT,
    INFT,
    MetadataViews.Resolver
  {
    pub var history: [HistoryPoint]
    pub let id: UInt64
```

```

pub var metadata: MetadataViews.NFTView
pub var file: {MetadataViews.File}?
pub var data: String
pub fun getViews(): [Type] { return [Type<MetadataViews.NFTView>()] }
pub fun resolveView(_ view: Type): AnyStruct? {
    switch view {
        case Type<MetadataViews.NFTView>():
            return self.metadata
        case Type<MetadataViews.Display>():
            return self.metadata.display
        case Type<MetadataViews.NFTCollectionDisplay>():
            return DocHub.getCollectionDisplay()
        case Type<MetadataViews.NFTCollectionData>():
            return DocHub.getNFTCollectionData()
        case Type<{MetadataViews.File}?>():
            return self.file
        case Type<MetadataViews.ExternalURL>():
            return self.metadata.externalURL
        case Type<String>():
            return self.data
    }
    return nil
}
pub fun update(report: HistoryPoint){
    self.history.append(
        HistoryPoint(
            metadata: self.metadata,
            file: self.file,
            data: self.data
        )
    )
    self.metadata = report.metadata
    self.file = report.file
    self.data = report.data
}
init(
    id: UInt64,
    metadata: MetadataViews.NFTView,
    file: {MetadataViews.File}?,
    data: String
){

```

```

        self.id = id
        self.metadata = metadata
        self.file = file
        self.data = data
        self.history = []
    }
}

pub resource interface DocCollectionPublic {
    pub fun clone(cloneID: UInt64): @NFT
    pub fun myNFT(id: UInt64): &NFT?
}

pub resource Collection:
    NonFungibleToken.Provider,
    NonFungibleToken.Receiver,
    NonFungibleToken.CollectionPublic,
    MetadataViews.ResolverCollection,
    DocCollectionPublic
{
    pub var ownedNFTs: @{UInt64: NonFungibleToken.NFT }
    pub fun withdraw(withdrawID: UInt64): @NonFungibleToken.NFT {
        let doc <- self.ownedNFTs.remove(key: withdrawID)!
        return <- doc
    }

    pub fun clone(cloneID: UInt64): @NFT {
        pre {
            self.ownedNFTs[cloneID] != nil: "NFT does not exist in the
collection!"
        }
        let doc = (&self.ownedNFTs[cloneID] as &NonFungibleToken.NFT?)!
        let metadataAny = doc.resolveView(Type<MetadataViews.NFTView>())
        ?? panic( "withdrawn NFT is not of specified type")
        let metadata = metadataAny! as! MetadataViews.NFTView
        let fileAny = doc.resolveView(Type<{MetadataViews.File}>())
        let file = fileAny! as! {MetadataViews.File}?
        let dataAny = doc.resolveView(Type<String>())
        ?? panic( "withdrawn NFT is not of specified type")
        let data = dataAny! as! String
        let copyDoc <- create NFT(
            id: DocHub.totalSupply,
            metadata: metadata,
            file: file,

```

```

        data: data
    )
    DocHub.totalSupply = DocHub.totalSupply + 1
    return <- copyDoc
}
pub fun deposit(token: @NonFungibleToken.NFT){
    self.ownedNFTs[token.id] <-! token
}
pub fun getIDs(): [UInt64] {
    return self.ownedNFTs.keys
}
pub fun borrowNFT(id: UInt64): &NonFungibleToken.NFT {
    pre {
        self.ownedNFTs[id] != nil: "NFT does not exist in the
collection!"
    }
    return (&self.ownedNFTs[id] as &NonFungibleToken.NFT?)!
}
pub fun borrowNFTSafe(id: UInt64): &NonFungibleToken.NFT? {
    post {
        result == nil || result!.id == id: "The returned reference's
ID does not match the requested ID"
    }
    return (&self.ownedNFTs[id] as &NonFungibleToken.NFT?)
}
pub fun myNFT(id: UInt64): &NFT?{
    let nft = (&self.ownedNFTs[id] as auth &NonFungibleToken.NFT?)!
    return nft as! &NFT
}

pub fun borrowViewResolver(id: UInt64):
&AnyResource{MetadataViews.Resolver} {
    pre {
        self.ownedNFTs[id] != nil : "NFT does not exist in
collection."
    }
    let nftRef = (&self.ownedNFTs[id] as auth &NonFungibleToken.NFT?)!
    let fullNft = nftRef as! &NFT
    return fullNft
}
init() {
    self.ownedNFTs <- {}
}

```

```

    }

    destroy() {
        destroy self.ownedNFTs
    }
}

init(){
    self.totalSupply = 0
    self.storagePath = /storage/twentyFourMiles
    self.publicPath = /public/twentyFourMiles
    self.privatePath = /private/twentyFourMiles
}

pub fun createEmptyCollection(): @NonFungibleToken.Collection {
    return <-create Collection()
}

        pub fun mintDoc(metadata: MetadataViews.NFTView, file:
{MetadataViews.File}?, data: String): @NFT {
    var newDoc <- create NFT(
        id: self.totalSupply,
        metadata: metadata,
        file: file,
        data: data
    )
    emit NFTMinted(id: self.totalSupply, setId: 1, templateId: 1, serial:
self.totalSupply)
    self.totalSupply = self.totalSupply + 1
    return <-newDoc
}

pub fun getNFTCollectionData(): MetadataViews.NFTCollectionData {
    let pubType = Type<&{
        NonFungibleToken.CollectionPublic,
        NonFungibleToken.Receiver,
        MetadataViews.ResolverCollection
    }>()
    let privateType = Type<&{
        NonFungibleToken.Provider,
        NonFungibleToken.CollectionPublic,
        MetadataViews.ResolverCollection
    }>()
    return MetadataViews.NFTCollectionData(
        storagePath: DocHub.storagePath,

```

```

        publicPath: DocHub.publicPath,
        providerPath: DocHub.privatePath,
        publicCollection: pubType,
        publicLinkedType: pubType,
        providerLinkedType: privateType,
        createEmptyCollectionFunction: DocHub.createEmptyCollection
    )
}

pub fun getCollectionDisplay(): MetadataViews.NFTCollectionDisplay{
    let file = MetadataViews.HTTPFile(url:
"https://ipfs.io/ipfs/bafybeiccbbjyhv4bdlrlc4ocr66rgfqv2xgef36naeyqxrwoc63x44b
3by/NUZP.png")
    let media = MetadataViews.Media(file: file, mediaType: "image/jpeg")
    return MetadataViews.NFTCollectionDisplay(
        name: "DocHub",
        description: "upload for test",
        externalURL: MetadataViews.ExternalURL(url:
"https://fc.vseosvita.ua/002m8e-c811/002.jpg"),
        squareImage: media,
        bannerImage: media,
        socials: {}
    )
}
}

```

ДОДАТОК Б

Код реалізації смарт-контракту для застосунку що імітує мутабельність

doc-v2.contract.cdc

```
import NonFungibleToken from 0x631e88ae7f1d7c20
import MetadataViews from 0x631e88ae7f1d7c20
pub contract DocHubV2: NonFungibleToken {
  pub var totalSupply: UInt64
  pub let storagePath: StoragePath
  pub let publicPath: PublicPath
  pub let privatePath: PrivatePath
  pub event ContractInitialized()
  pub event Withdraw(id: UInt64, from: Address?)
  pub event Deposit(id: UInt64, to: Address?)
  pub event AddReport(nftId: UInt64)
  pub struct HistoryPoint {
    pub let metadata: MetadataViews.NFTView
    pub let file: {MetadataViews.File}?
    pub let data: String
    pub let date: UFix64
    init(
      metadata: MetadataViews.NFTView,
      file: {MetadataViews.File}?,
      data: String,
    ){
      self.metadata = metadata
      self.file = file
      self.data = data
      self.date = getCurrentBlock().timestamp
    }
  }
  pub resource interface INFT {
    pub fun addVersion(report: HistoryPoint): @NFT
  }
  pub resource NFT:
    NonFungibleToken.INFT,
    INFT,
    MetadataViews.Resolver
  {
```

```

pub let id: UInt64
pub var newInfo: UInt64?
pub let metadata: MetadataViews.NFTView
pub let file: {MetadataViews.File}?
pub let data: String
pub let date: UFix64?
pub fun getViews(): [Type] { return [Type<MetadataViews.NFTView>()] }
pub fun resolveView(_ view: Type): AnyStruct? {
  switch view {
    case Type<MetadataViews.NFTView>():
      return self.metadata
    case Type<MetadataViews.Display>():
      return self.metadata.display
    case Type<MetadataViews.NFTCollectionDisplay>():
      return DocHubV2.getCollectionDisplay()
    case Type<MetadataViews.NFTCollectionData>():
      return DocHubV2.getNFTCollectionData()
    case Type<{MetadataViews.File}?>():
      return self.file
    case Type<MetadataViews.ExternalURL>():
      return self.metadata.externalURL
    case Type<String>():
      return self.data
  }
  return nil
}
pub fun addVersion(report: HistoryPoint): @NFT {
  let nft: @NFT <- DocHubV2.mintDoc(
    metadata: report.metadata,
    file: report.file,
    data: report.data
  )
  self.newInfo = nft.id
  return <-nft
}
init(
  id: UInt64,
  metadata: MetadataViews.NFTView,
  file: {MetadataViews.File}?,
  data: String
){

```

```

        self.id = id
        self.metadata = metadata
        self.file = file
        self.data = data
        self.date = getCurrentBlock().timestamp
        self.newInfo = nil
    }
}

pub resource interface DocCollectionPublic {
    pub fun myNFT(id: UInt64): &NFT?
}

pub resource Collection:
    NonFungibleToken.Provider,
    NonFungibleToken.Receiver,
    NonFungibleToken.CollectionPublic,
    MetadataViews.ResolverCollection,
    DocCollectionPublic
{
    pub var ownedNFTs: @{UInt64: NonFungibleToken.NFT }
    pub fun withdraw(withdrawID: UInt64): @NonFungibleToken.NFT {
        let doc <- self.ownedNFTs.remove(key: withdrawID)!
        return <- doc
    }
    pub fun deposit(token: @NonFungibleToken.NFT){
        self.ownedNFTs[token.id] <-! token
    }
    pub fun getIDs(): [UInt64] {
        return self.ownedNFTs.keys
    }
    pub fun borrowNFT(id: UInt64): &NonFungibleToken.NFT {
        pre {
            self.ownedNFTs[id] != nil: "NFT does not exist in the
collection!"
        }
        return (&self.ownedNFTs[id] as &NonFungibleToken.NFT?)!
    }
    pub fun borrowNFTSafe(id: UInt64): &NonFungibleToken.NFT? {
        post {
            result == nil || result!.id == id: "The returned reference's
ID does not match the requested ID"
        }
    }
}

```

```

        return (&self.ownedNFTs[id] as &NonFungibleToken.NFT?)
    }
    pub fun myNFT(id: UInt64): &NFT?{
        let nft = (&self.ownedNFTs[id] as auth &NonFungibleToken.NFT?)!
        return nft as! &NFT
    }

    pub fun borrowViewResolver(id: UInt64):
    &AnyResource{MetadataViews.Resolver} {
        pre {
            self.ownedNFTs[id] != nil : "NFT does not exist in
collection."
        }
        let nftRef = (&self.ownedNFTs[id] as auth &NonFungibleToken.NFT?)!
        let fullNft = nftRef as! &NFT
        return fullNft
    }
    init() { self.ownedNFTs <- {} }
    destroy() { destroy self.ownedNFTs }
}
init(){
    self.totalSupply = 0
    self.storagePath = /storage/DocHubV2
    self.publicPath = /public/DocHubV2
    self.privatePath = /private/DocHubV2
}
pub fun createEmptyCollection(): @NonFungibleToken.Collection {
    return <-create Collection()
}

    pub fun mintDoc(metadata: MetadataViews.NFTView, file:
{MetadataViews.File}?, data: String): @NFT {
        var newDoc <- create NFT(
            id: self.totalSupply,
            metadata: metadata,
            file: file,
            data: data
        )
        emit NFTMinted(id: self.totalSupply, setId: 1, templateId: 1, serial:
self.totalSupply)
        self.totalSupply = self.totalSupply + 1
        return <-newDoc
    }
}

```

```

pub fun getNFTCollectionData(): MetadataViews.NFTCollectionData {
    let pubType = Type<&{
        NonFungibleToken.CollectionPublic,
        NonFungibleToken.Receiver,
        MetadataViews.ResolverCollection
    }>()
    let privateType = Type<&{
        NonFungibleToken.Provider,
        NonFungibleToken.CollectionPublic,
        MetadataViews.ResolverCollection
    }>()
    return MetadataViews.NFTCollectionData(
        storagePath: DocHubV2.storagePath,
        publicPath: DocHubV2.publicPath,
        providerPath: DocHubV2.privatePath,
        publicCollection: pubType,
        publicLinkedType: pubType,
        providerLinkedType: privateType,
        createEmptyCollectionFunction: DocHubV2.createEmptyCollection
    )
}

pub fun getCollectionDisplay(): MetadataViews.NFTCollectionDisplay{
    let file = MetadataViews.HTTPFile(url:
"https://ipfs.io/ipfs/bafybeiccbbjyhv4bdlrlc4ocr66rgfqv2xgef36naeyqxrwo63x44b
3by/NUZP.png")
    let media = MetadataViews.Media(file: file, mediaType: "image/jpeg")
    return MetadataViews.NFTCollectionDisplay(
        name: "DocHubV2",
        description: "Example smartcontract for develop applications with
show update data of NFT",
        externalURL: MetadataViews.ExternalURL(url: ""),
        squareImage: media,
        bannerImage: media,
        socials: {}
    )
}
}

```

ДОДАТОК В

ПРЕЗЕНТАЦІЯ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Кафедра інформаційної безпеки та наноелектроніки

Технології безпеки надання доступу до облікових записів у інформаційних мережах

Підготував:
Студент групи БК-812м
І.О. ГОЛОВКО

Керівник:
к.ф.м.н., доцент кафедри ІБтаН
Г.Л. КОЗІНА

Рисунок В.1 – Титульний слайд

АНОТАЦІЯ

- **Об'єкт дослідження** – технології мутабельності даних в блокчейні, що надають можливість змінювати чи імітувати зміну даних в блокчейні.
- **Предмет роботи** – можливі реалізації механізму мутабельності даних на прикладі смартконтракту та застосунок, а також дослідження: як імітація мутабельності (можливості модифікації) може впливати на рівень безпеки блокчейну.
- **Мета роботи:**
 - дослідити технології, що дозволяють імітувати мутабельності даних в блокчейні на прикладі реалізації відповідного програмного забезпечення та можливостей протоколу блокчейну;
 - реалізувати застосунок, що дозволяє імітувати мутабельність даних в блокчейні;
 - реалізувати та вивантажити у мережу смартконтракт, що дозволяє мутувати ресурс всередині мережі блокчейн.

Рисунок В.2 – Слайд 2

ЗАДАЧІ РОБОТИ

1. Аналіз впливу алгоритмів мутації даних на захищеність та швидкодії блокчейн мереж.
2. Аналіз шляхів реалізації алгоритмів, що дозволяють імітувати мутабельність даних на основі блокчейн мережі.
3. Розробка реалізацій, що дозволяють практичним шляхом дослідити алгоритми імітації мутабельності даних
 - a. Розробка серверного застосунку з застосуванням програмного пакету NodeJS.
 - b. Розробка смартконтракту, що реалізує можливість мутування даних.

3/18

Рисунок В.3 – Слайд 3

ВИЗНАЧЕННЯ БЛОКЧЕЙНУ ТА ЙОГО ОСНОВНІ ПРИНЦИПИ

Блокчейн - розподілена база даних, яка зберігає впорядкований ланцюжок блоків, що постійно довшас.

Головні принципи блокчейн мереж:

1. Децентралізованість.
2. Криптографічна безпека.
3. Незмінність даних.
4. Консенсус.
5. Прозорість.

Публічний блокчейн – тип блокчейну, що відкритий для громадськості.



Приватний блокчейн – тип блокчейну, що базується на тих самих принципах, що і публічний, але «адмініструється» конкретними особами або корпоративно.

Федерований блокчейн – тип блокчейну, який є комбінацією публічного та приватного блокчейнів.



4/18

Рисунок В.4 – Слайд 4

ПЕРЕВАГИ ТА НЕДОЛІКИ БЛОКЧЕЙНУ

Переваги:

- незмінність даних – блокчейн забезпечує незмінність та недоступність для редагування даних;
- децентралізація – блокчейн працює на основі розподіленої мережі вузлів;
- прозорість – всі транзакції в блокчейні є публічними і доступними для перегляду;
- висока захищеність – блокчейн використовує криптографічні методи для забезпечення безпеки та конфіденційності даних.

Недоліки:

- незмінність даних – неможливо виправити похибку у збережених даних;
- низька швидкість – блокчейн може мати обмежену пропускну здатність та швидкість обробки транзакцій;
- витрати та енергосміність – мережі блокчейн, через складність криптоперетворень, є енергоємними та вимагають значних витрат на обчислення.

5/18

Рисунок В.5 – Слайд 5

МЕХАНІЗМИ БЛОКЧЕЙНУ ТА ІНСТРУМЕНТИ ВЗАЄМОДІЇ З НИМ

Транзакції – операції запису.

Сценарії – операції читання.

Смартконтракт – шаблонізоване розширення протоколу блокчейн мережі.

```

47 pub contract interface NonfungibleToken {
48
49     /// The total number of tokens of this type in existence
50     pub var totalSupply: UInt64
51
52     /// Event that emitted when the NFT contract is initialized
53     ///
54     pub event ContractInitialized()
55
56     /// Event that is emitted when a token is withdrawn,
57     /// indicating the owner of the collection that it was withdrawn from.
58     ///
59     /// If the collection is not in an account's storage, 'from' will be 'nil'.
60     ///
61     pub event Withdraw(id: UInt64, from: Address?)
62
63     /// Event that emitted when a token is deposited to a collection.
64     ///
65     /// It indicates the owner of the collection that it was deposited to.
66     ///
67     pub event Deposit(id: UInt64, to: Address?)
68
69     /// Interface that the NFTs have to conform to
70     /// The metadata views methods are included here temporarily
71     /// because enforcing the metadata interfaces in the standard
72     /// would break many contracts in an upgrade. Those breaking changes
73     /// are being saved for the stable cadence milestone
74     ///
75     pub resource interface DNFT {
76     }
77
78     /// Requirement that all conforming NFT smart contracts have
79     /// to define a resource called NFT that conforms to DNFT
80     ///
81     pub resource NFT: DNFT {
82     }
83
84     /// Interface to mediate withdraws from the Collection
85     ///
86     pub resource interface Provider {
87     }
88
89     /// Interface to mediate deposits to the Collection
90     ///
91     pub resource interface Receiver {
92     }
93
94 }

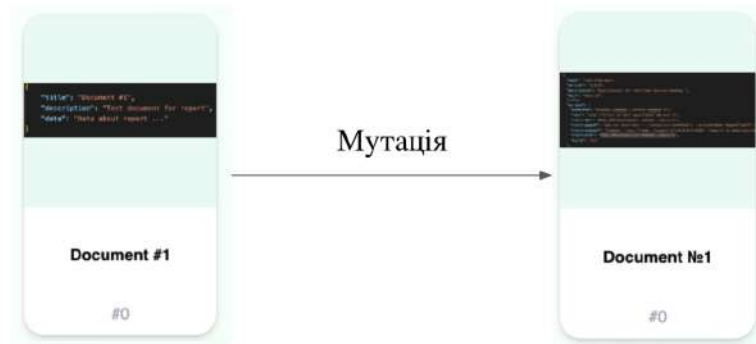
```

6/18

Рисунок В.6 – Слайд 6

МУТАБЕЛЬНІСТЬ

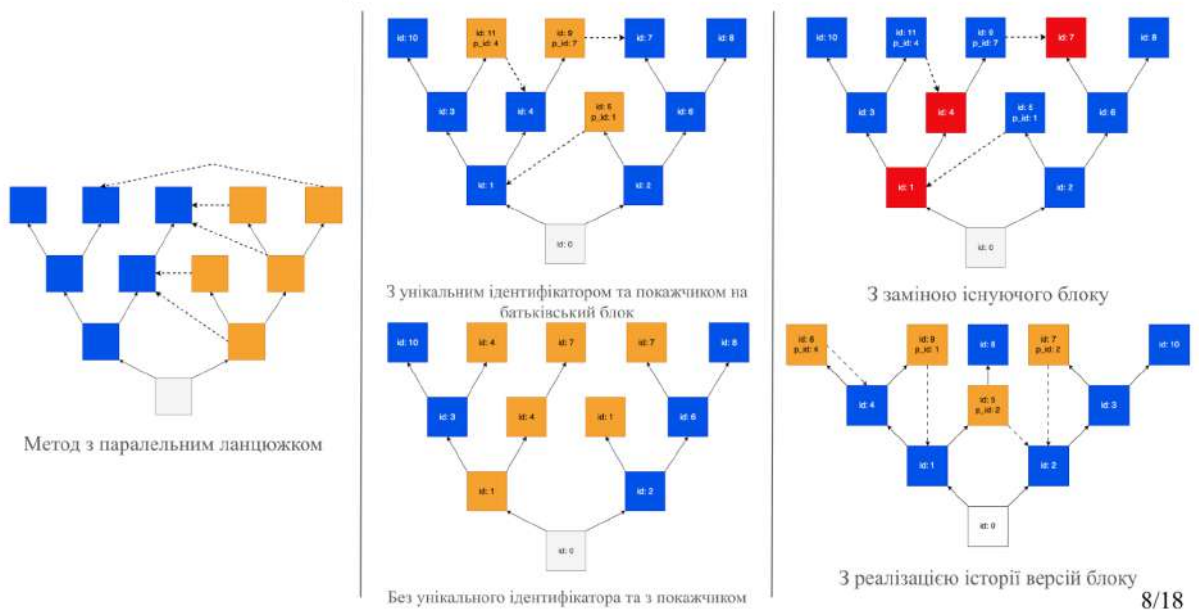
Мутабельність даних (у блокчейні) – це імітація змінності або можливість змінювати представлення кінцевих даних, що зберігаються в блокчейні та надаються користувачу, без порушення принципу імутабельності.



7/18

Рисунок В.7 – Слайд 7

МЕХАНІЗМИ МУТАЦІЇ В БЛОКЧЕЙНІ



8/18

Рисунок В.8 – Слайд 8

ВПЛИВ МУТАБЕЛЬНОСТІ НА ЗАХИЩЕНІСТЬ БЛОКЧЕЙНУ

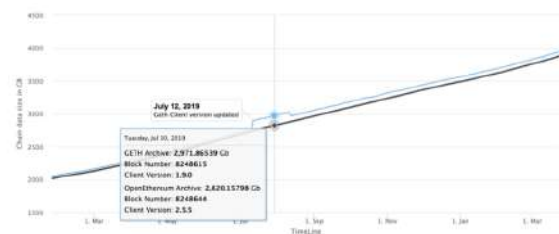
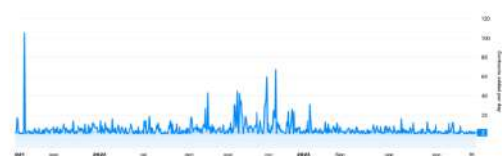
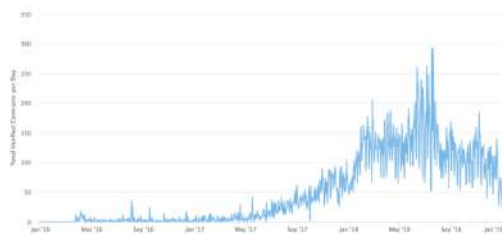
Небезпеки, що супроводжують механізм мутації в мережі блокчейну:

1. Вразливість несанкціонованих змін. Дана вразливість полягає у тому, що ми не можемо гарантувати, що останні зміни були внесені не зловмисником.
2. Атака подвійного витрачання – це заздалегідь передбачена атака, або збій у алгоритмі підтвердження транзакції, при якому один і той же запис потрапляє у ланцюг блокчейну двічі.
3. Вразливість більшості (або атака 51%) – атака, яка виникає при концентрації > 50% технічних ресурсів блокчейну під контролем зловмисника, або групи зловмисників (притаманна лише для реалізації з паралельним ланцюжком).
4. Навантаження на мережу. Зі збільшенням кількості запитів навантаження пропорційно збільшується.
5. Збільшення простору для можливого впливу людського фактора на достовірність даної інформації.

9/18

Рисунок В.9 – Слайд 9

КОНКРЕТНІ ПРИКЛАДИ ВПЛИВУ МУТАБЕЛЬНОСТІ ДАНИХ



Ethereum

14 березня 2016 року – 30 липня 2019 року

Кількість транзакцій – 8,248,644

Загальний розмір даних – 2971.86 Гб

Flow

11 жовтня 2020 року – 24 квітня 2023 року

Кількість транзакцій – 401,129,954

Загальний розмір даних – 498.86 Гб

10/18

Рисунок В.10 – Слайд 10

РОЗРОБКА СМАРТКОНТРАКТУ, ЩО РЕАЛІЗУЄ ФУНКЦІЇ МУТАЦІЇ



Назва смартконтракту – DocHub

Ресурси смартконтракту – Collection, NFT

Ресурс що реалізує метод мутабельності – NFT



Посилання на смартконтракт DocHub

11/18

Рисунок В.11 – Слайд 11

ПРИКЛАД ТРАНЗАКЦІЙ ДЛЯ СТВОРЕННЯ РЕСУРСІВ

Транзакція на створення ресурсу колекції

```
import NonFungibleToken from 0x631e88ae7f1d7c20
import MetadataViews from 0x631e88ae7f1d7c20
import DocHub from 0x88b76b3b47dec36d

transaction {
  let accountn: AuthAccount
  prepare(acct: AuthAccount) {
    self.accountn = acct
  }
  execute {
    let collection <- DocHub.createEmptyCollection()
    self.accountn.save(<- collection, to: DocHub.storagePath)
    let collectionCapab =
      self.accountn.link<&(NonFungibleToken.Receiver)>{
        DocHub.publicPath,
        target: DocHub.storagePath
      }
    ?? panic("Collection in storage not exist")
    self.accountn.link<
      &(NonFungibleToken.Provider,
        NonFungibleToken.CollectionPublic,
        DocHub.DocCollectionPublic)
    >{DocHub.privatePath, target: DocHub.storagePath}
  }
}
```

Transaction ID: 96d6a56091eb37cc208b538fd5b39c5ae5e90f8761368dcaae3f562665ae4c8

Block ID: a447a19a873a191b2a9dd8b08789e7c368287b0bed44a2a5f5ab2262cf484eba

Block Height: 13557476

Status: SEALED

ID: 96d6a56091eb37cc208b538fd5b39c5ae5e90f8761368dcaae3f562665ae4c8

Payer: 08d2f783ebe1a4f0

Authorizers: [08d2f783ebe1a4f0]

Proposal Key:

Address: 08d2f783ebe1a4f0

Index: 0

Sequence: 0

```
import NonFungibleToken from 0x631e88ae7f1d7c20
import MetadataViews from 0x631e88ae7f1d7c20
import DocHub from 0x88b76b3b47dec36d

transaction (docUrl: String, mediaUrl: String, data: String) {
  let collection: &(NonFungibleToken.Receiver)
  prepare(acct: AuthAccount) {
    self.collection = acct.getCapability<
      &(NonFungibleToken.Receiver)
    >{DocHub.publicPath}
    .borrow()
    ?? panic("Could not borrow receiver reference")
  }
  execute {
    let file = MetadataViews.HTTPFile(url: mediaUrl)
    let display = MetadataViews.Display(
      name: "second document",
      description: "upload for test",
      thumbnail: file
    )
    let fileOfDocument = MetadataViews.HTTPFile(url: docUrl)
    let metadata = MetadataViews.NFTView(
      id: DocHub.totalSupply,
      uuid: DocHub.totalSupply,
      display: display,
      externalURL: MetadataViews.ExternalURL(url: docUrl)
    )
    let newDoc <- DocHub.mintDoc(
      metadata: metadata,
      file: fileOfDocument,
      data: data
    )
    self.collection.deposit(token: <- newDoc)
  }
}
```

Транзакція на створення ресурсу NFT

12/18

Рисунок В.12 – Слайд 12

ПОКРОКОВА СХЕМА ВИКОРИСТАННЯ ЗАСТОСУНКУ

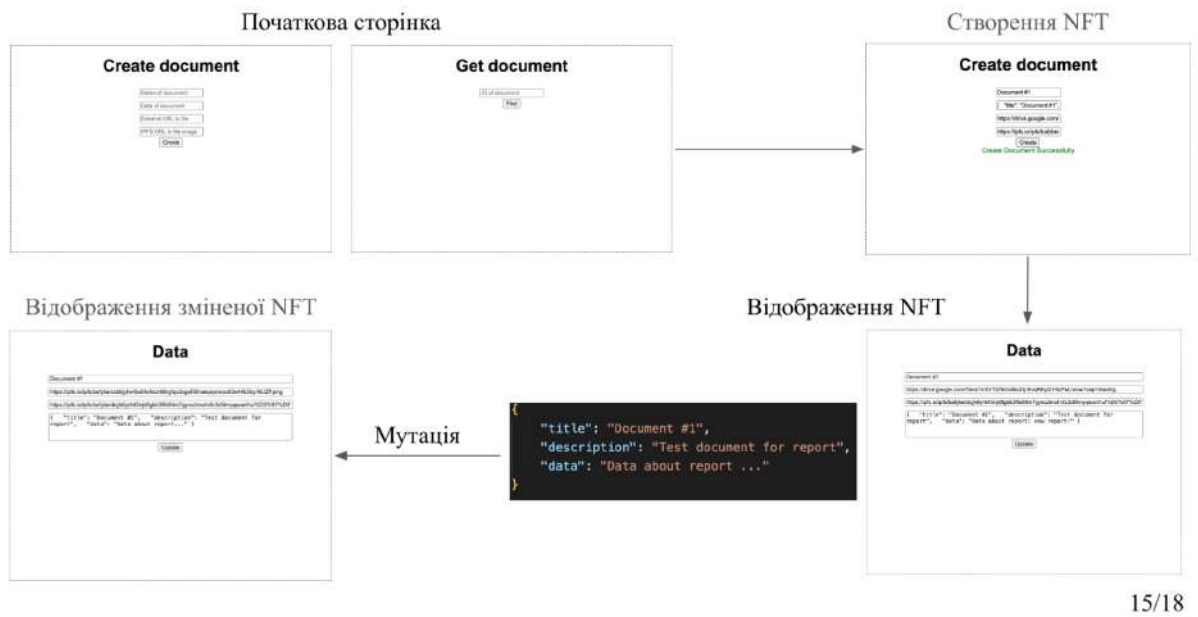


Рисунок В.15 – Слайд 15

ВІЗУАЛІЗАЦІЯ ЗМІН У МЕРЕЖІ БЛОКЧЕЙНУ

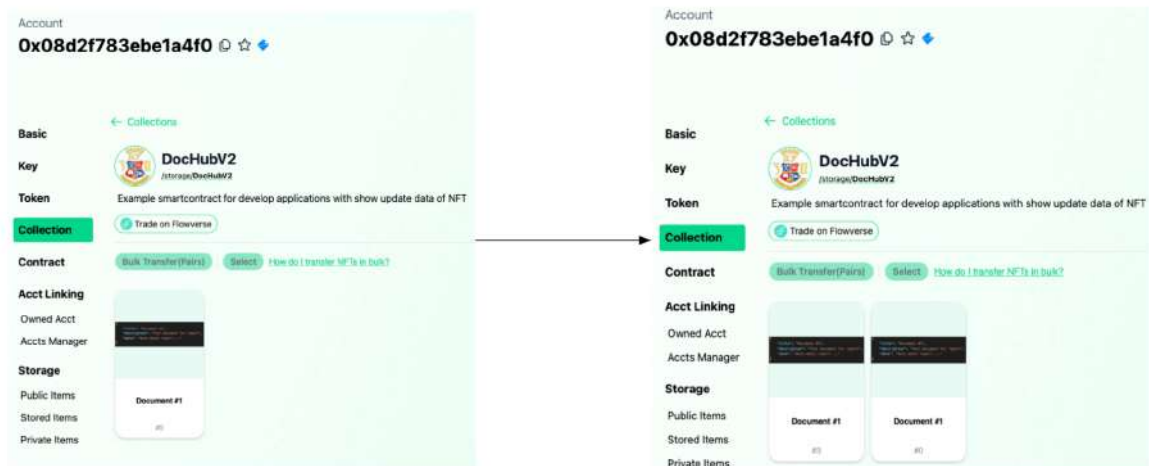


Рисунок В.16 – Слайд 16

ВИСНОВКИ

- Реалізація механізмів імітації мутабельності даних в блокчейні, має ряд позитивних факторів, таких як:
 - збільшене навантаження на мережу – ускладнення розрахункового завдання для зловмисника;
 - зменшення впливу потенційних помилок;
 - зменшення розміру блокчейну, відповідно зростання швидкості пошуку інформації в мережі.
- До недоліків реалізації механізмів мутації відносяться:
 - збільшене навантаження на мережу – збільшена вірогідність відмови у доступі;
 - збільшений простір до вразливостей викликаних людським фактором (помилки в алгоритмах смартконтракту, атаки що використовують соціальну інженерію).
- Аналіз розроблених реалізацій механізмів імітації мутації з використанням блокчейну, :
 - реалізація механізмів мутації з використанням вбудованих механізмів блокчейну (смартконтрактів), надає швидкий та простий інструмент для взаємодії з ресурсами в блокчейні, але має загрози, що базуються на: потенційних вразливостях протоколу; атаках що використовують соціальну інженерію тощо
 - реалізація механізмів мутації з використанням вбудованих механізмів блокчейну на рівні застосунку має складніший та повільний механізм надання доступу до даних, та підвернений таким загрозам: атака відмова у доступі, атака людина посередині тощо.

17/18

Рисунок В.17 – Слайд 17

Дякую за увагу !

Рисунок В.14 – Фінальний слайд