Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ

завідувач кафедри
мережевих та інтернет технологій
_____ Ю.В. Кравченко
«_____»_____2022 року

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

галузі знань 17 «Електроніка та телекомунікації»
за спеціальністю 172 «Телекомунікації та радіотехніка»
освітньо-професійна програма «Мережеві та інтернет технологій»

на тему:

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПРОВЕДЕННЯ РОЗРАХУНКІВ ЗА ТОВАРИ ТА ПОСЛУГИ З ВИКОРИСТАННЯМ НЕВЗАЄМОЗАМІНЮВАНОГО ТОКЕНА

Виконав: студент групи МІТм -21

Сергій ІГНАТЮК _____
(прізвище ім'я по-батькові) (підпис)

Керівник: доцент кафедри мережевих та інтернет технологій

к.т.н. Оксана ГЕРАСИМЕНКО _____
(посада, прізвище ім'я по-батькові) (підпис)

Київ - 2022

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій
_____ Юрій КРАВЧЕНКО

«_____»_____2022 року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ

Здобувачу вищої освіти Ігнатюку Сергію Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи:
Інформаційна технологія проведення розрахунків за товари та послуги з використанням невзаємозамінюваного токена

затверджена на засіданні кафедри МІТ «31» серпня 2022 р. протокол №1
2. Термін здачі закінченої роботи «05» грудня 2022 р.
3. Вихідні дані до проекту (роботи)

Блокчейн Ethereum, стандарти ERC-721, ERC-20, ERC-1155.

4. Зміст пояснювальної записки (перелік питань, що їх потрібно розробити, обсяг – 70-80 стор.)

Вступ

1. Дослідження методів та засобів організації платежів з використанням токенів. Постановка задачі

1.1 Огляд і аналіз існуючих технологій організації платежів з використанням токенів

1.2 Аналіз проблем реалізації смарт-контрактів

1.3 Постановка задачі

2. Розробка інформаційної технології проведення розрахунків за товари та послуги з використанням невзаємозамінюваного токена

2.1 Концепція проведення розрахунків за товари та послуги з використанням невзаємозамінюваного токена

2.2 Реалізація смарт-контракту невзаємозамінюваного токена для використання у платежах

2.3 Проектування та реалізація програмного додатку для проведення платежів із використання розробленого контракту. Тестування інформаційної технології

3. Аналіз смарт-контрактів на предмет виявлення вразливостей

3.1 Фреймворки для виявлення вразливостей у смарт-контрактах Ethereum

3.2 Розробка рекомендацій щодо усунення вразливостей смарт-контрактів

Висновки

5. Перелік графічного матеріалу 8-10 слайдів

Дата видачі завдання 01.09.2022р.

Керівник роботи Оксана ГЕРАСИМЕНКО

(підпис) (посада, прізвище, ім'я, по батькові)

Завдання прийняв до виконання Сергій ІГНАТЮК

(підпис)

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ РОБОТИ

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Підготовчий	24.10.2022	
2	Розділ 1	01.11.2022	
3	Розділ 2	15.11.2022	
4	Розділ 3	01.12.2022	
5	Доповідь та слайди	05.12.2021	
6	Пояснювальна записка	05.12.2022	

Здобувач вищої освіти _____ Сергій ІГНАТЮК
(підпис)

Керівник _____ Оксана ГЕРАСИМЕНКО
(підпис)

РЕФЕРАТ

Пояснювальна записка: 77 с., 34 рис., 3 додатки, 27 джерел.

Об’єкт дослідження: механізм розрахунків за товари та послуги на базі блокчейна Ethereum.

Предмет дослідження: смарт-контракти для реалізації розрахунків з використанням невзаємозамінюваного токена.

Мета роботи (проекту): спроектувати та реалізувати технологію оплати товарів з використанням невзаємозамінюваного токена блокчейна Ethereum.

Методи дослідження: системний підхід, порівняльний аналіз, структурний аналіз, методи проектування програмного забезпечення.

У спеціальній частині розглянуто сучасний стан і особливості систем оплат з використанням блокчейн технологій.

У роботі проведено аналіз існуючих систем оплати товарів з використанням токенів та криптовалюти.

Запропоновано використання невзаємозамінюваного токена як альтернативного засобу доступу користувачів до додаткових послуг у процесі оплати за товари та послуги на платформі блокчейн.

Розроблено смарт-контракти на мові Solidity, розгортання яких на платформі Ethereum дозволяє реалізувати технологію оплати товарів з використанням невзаємозамінюваного токена блокчейна Ethereum.

Практичне значення роботи полягає у розробці технології оплати товарів з використанням невзаємозамінюваного токена, що може бути використано як альтернативний спосіб заохочення клієнтів та надання їм додаткових послуг. Також у роботі розроблено рекомендації щодо створення смарт-контрактів з метою уникнення вразливостей у них.

Результати здійснених у кваліфікаційному проекті досліджень можуть бути використані як продукт для бізнес систем.

Наукова новизна полягає у тому, що створена система оплати товарів та послуг, яка, на відміну від існуючих, базується на використанні невзаємозамінюваного токена, що дає можливість індивідуального підходу до клієнтів та нівелює можливості неправомірного використання адресних заохочень третіми особами. Розроблена технологія проходила тестування в реальних умовах і оптимізована під справжню існуючу систему оплати.

Напрямки подальшого розвитку роботи полягають у вдосконаленні створеної системи з метою підвищення ефективності; додавання можливості для створення індивідуальних невзаємозамінюваних токенів; додавання функціоналу бонусів з використання криптовалюти.

Ключові слова: БЛОКЧЕЙН, СИСТЕМА ОПЛАТИ, НЕВЗАЄМОЗАМІНЮВАНИЙ ТОКЕН, СМАРТ-КОНТРАКТ, ВРАЗЛИВОСТІ СМАРТ-КОНТРАКТУ, ETHEREUM, SOLIDITY.

ЗМІСТ

ВСТУП

Стрімкий розвиток технології блокчейн спровокував хвилю інновацій у сфері платіжних систем. Блокчейн пропонує децентралізовану, безпечну та прозору платформу для проведення транзакцій без необхідності посередників, таких як банки чи платіжні процесори. Це надає потенціал для трансформації способів здійснення платежів і переказу вартості, пропонуючи нові можливості для створення ефективних та недорогих платіжних систем, які можуть принести користь окремим особам, компаніям і навіть цілим економікам.

Технологія блокчейн — це децентралізована та розподілена цифрова книга, яка використовується для запису транзакцій у мережі комп'ютерів. Це безпечний і прозорий спосіб зберігання та обміну даними і він має потенціал зробити революцію в багатьох галузях. У блокчейні дані зберігаються в блоках, які об'єднані разом, і кожен блок містить унікальний код, що називається хеш, який дозволяє його легко ідентифікувати та перевірити. Це робить майже неможливим для будь-кого змінити або підробити дані в блокчейні. Технологія блокчейн найчастіше асоціюється з розвитком криптовалют, таких як біткойн, але вона також має багато інших потенційних застосувань.

Однак впровадження платіжних систем, заснованих на блокчейні, все ще перебуває на початковій стадії та існує багато проблем і можливостей, які необхідно вивчити та зрозуміти. Ця дипломна робота має на меті забезпечити комплексний аналіз потенціалу та обмежень використання технології блокчейн для створення платіжної системи. Він розглядатиме поточний стан справ у цій галузі, включаючи ключових гравців та ініціативи, технічні проблеми та рішення, нормативно-правовий ландшафт та потенційний вплив на різних зацікавлених сторін.

1. ДОСЛІДЖЕННЯ МЕТОДІВ ТА ЗАСОБІВ ОРГАНІЗАЦІЇ ПЛАТЕЖІВ З ВИКОРИСТАННЯМ ТОКЕНІВ

1.1 Огляд і аналіз існуючих технологій організації платежів з використанням токенів

Платіжні рішення на блокчейні набувають популярності через зростання попиту на криптовалюту. Технологія блокчейн суттєво вплинула на фінансовий сектор з моменту своєї появи, і з того часу платіжні рішення блокчейну були вдосконалені, що призвело до ефективності та прозорості цих платіжних процесів.

Для багатьох блокчейн став кращим способом поводження з грошима, головним чином завдяки своїй ефективності [6]. Основна ідея використання технології розподіленої книги для фінансів полягає в тому, щоб відмовитися від централізованих установ, таких як банки, дозволяючи здійснювати оплату через блокчейн так само просто, як натиснути кнопку «Надіслати». Жодних періодів очікування грошових переказів або непотрібних комісій за обробку сторонніми особами. Криптовалюти на основі блокчейну можна миттєво передавати — і записувати для цілей аудиту — по всьому світу, збільшуючи ліквідність і швидкість роботи на ринках.

Крім того, система смарт-контрактів блокчейну може призупинити платежі, якщо погоджені умови порушуються, а використання криптовалюти може полегшити регуляцію, яка часто захоплює обробку міжнародних платежів. Перешкоди, такі як денні періоди очікування та високі комісії за транскордонні транзакції, зменшуються завдяки багатьом системам криптовалютних платежів.

У контексті платежу блокчейн записує інформацію, що стосується рахунків (баланс та історія транзакцій). Блок містить цифрові записи транзакцій із міткою часу. Блок не можна змінити після додавання, що робить блокчейн серією

незмінних записів. Припустімо, що дві особи є членами захищеного блокчейну та тримають розподілену книгу. Платіж можна здійснити, додавши нову транзакцію до ланцюжка з платіжною інформацією. Після цього обидві сторони підтверджують платіж і весь процес завершується за кілька секунд [7].

Платіжна система з використанням технології блокчейн — це система, яка забезпечує безпечний і прозорий переказ коштів за допомогою розподіленої книги. У платіжній системі блокчейн транзакції реєструються в мережі взаємопов'язаних комп'ютерів, а не обробляються центральним органом. Це робить систему надзвичайно безпечною та стійкою до шахрайства, оскільки транзакції неможливо змінити або видалити після їх додавання до блокчейну.

Одна з ключових переваг використання технології блокчейн для платіжних систем полягає в тому, що вона дозволяє здійснювати однорангові транзакції в режимі реального часу без необхідності стороннього посередника. Це може зробити процес надсилання й отримання платежів швидшим, дешевшим і безпечнішим.

Крім того, оскільки блокчейн-мережі є децентралізованими, до них може отримати доступ і використовувати кожен, хто має підключення до Інтернету. Це робить їх особливо придатними для використання в країнах, що розвиваються, та інших регіонах, де доступ до традиційних фінансових послуг обмежений.

Загалом, платіжні системи, які використовують технологію блокчейн, можуть революціонізувати спосіб здійснення транзакцій і переказу коштів, зробивши процес швидшим, дешевшим і безпечнішим.

Одним із прикладів платіжної системи, що використовує технологію блокчейн, є біткойн. Біткойн — це цифрова валюта, яка використовує розподілену книгу під назвою блокчейн для запису транзакцій. Кожна транзакція перевіряється учасниками мережі, відомими як майнери, і додається до блокчейну. Це гарантує, що транзакція є безпечною та не може бути змінена чи видалена.

Користувачі можуть надсилати та отримувати біткойни за допомогою цифрового гаманця, який зберігає їхній баланс у біткойнах і дозволяє їм безпечно здійснювати транзакції. Оскільки мережа є децентралізованою та не залежить від центрального органу, користувачі можуть здійснювати однорангові транзакції без допомоги банку чи іншого стороннього посередника [8].

Іншим прикладом виступають централізовані біржі, які дають можливість оформити картку та використовувати її для розрахунків (рис 1.1). Для проведення платежу біржа використовує токени, які має користувач на своєму обліковому записі, та конвертує її у валюту, в якій розплачується користувач, наприклад, євро (рис 1.2).

Використовуючи карти біржі, користувачі отримують додаткові переваги, наприклад, кешбек від будь-якої покупки, безпечність, оскільки картки використовують системи такі як Visa та MasterCard.

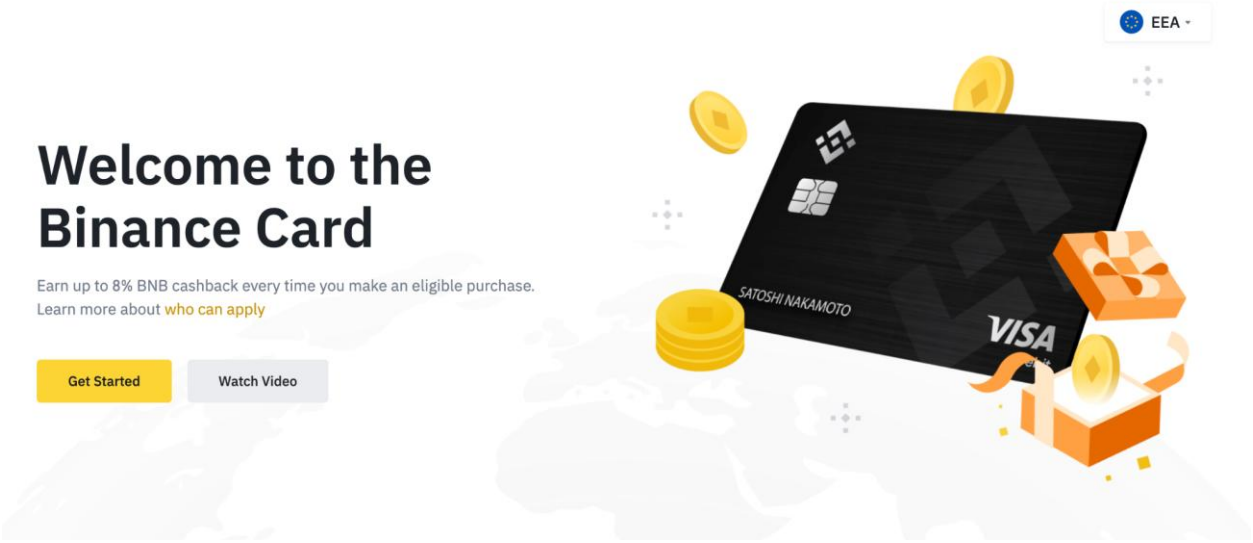

Рисунок 1.1 – Оформлення картки Visa на біржі Binance [1]

У контексті криптовалюти централізована біржа (CEX) — це тип платформи, який дозволяє користувачам купувати та продавати криптовалюти за допомогою фіатних валют або інших криптовалют. Централізованими біржами зазвичай керує

одна компанія чи організація, яка веде центральну книгу всіх транзакцій, що відбуваються на платформі. Це означає, що біржа контролює активи, якими торгують на платформі, і відповідає за підтримку безпеки та цілісності транзакцій [5].

На централізованій біржі користувачі можуть робити кілька речей, зокрема:

- купувати та продавати криптовалюти: користувачі можуть купувати та продавати широкий спектр криптовалют на централізованій біржі. Вони можуть розміщувати замовлення на покупку або продаж за певною ціною, а біржа зіставлятиме їх з іншими користувачами, які бажають торгувати за цією ціною;
- внесення та зняття коштів: користувачі можуть вносити та знімати кошти зі свого рахунку на біржі. Зазвичай це стосується як фіатної валюти (наприклад, доларів США, євро), так і криптовалюти;
- перегляд ринкових даних: користувачі можуть переглядати ринкові дані на біржі, включаючи ціни на різні криптовалюти та обсяг торгів, що відбуваються;
- маржинальна торгівля: деякі централізовані біржі дозволяють користувачам торгувати на маржі, що означає, що вони можуть позичати кошти на біржі, щоб збільшити свою купівельну спроможність. Це може бути ризикованою стратегією, оскільки вона може призвести до втрат, якщо ринок рухається проти користувача.

На відміну від цього, децентралізовані біржі (DEX) не мають центрального органу чи бухгалтерської книги. Натомість вони використовують технологію блокчейн, щоб уможливити однорангову торгівлю криптовалютами без необхідності центрального посередника. Децентралізовані біржі часто вважаються більш безпечними та приватними, ніж централізовані біржі, але вони також можуть

бути менш зручними для користувачів і можуть пропонувати більш обмежений діапазон торгових опцій [9].

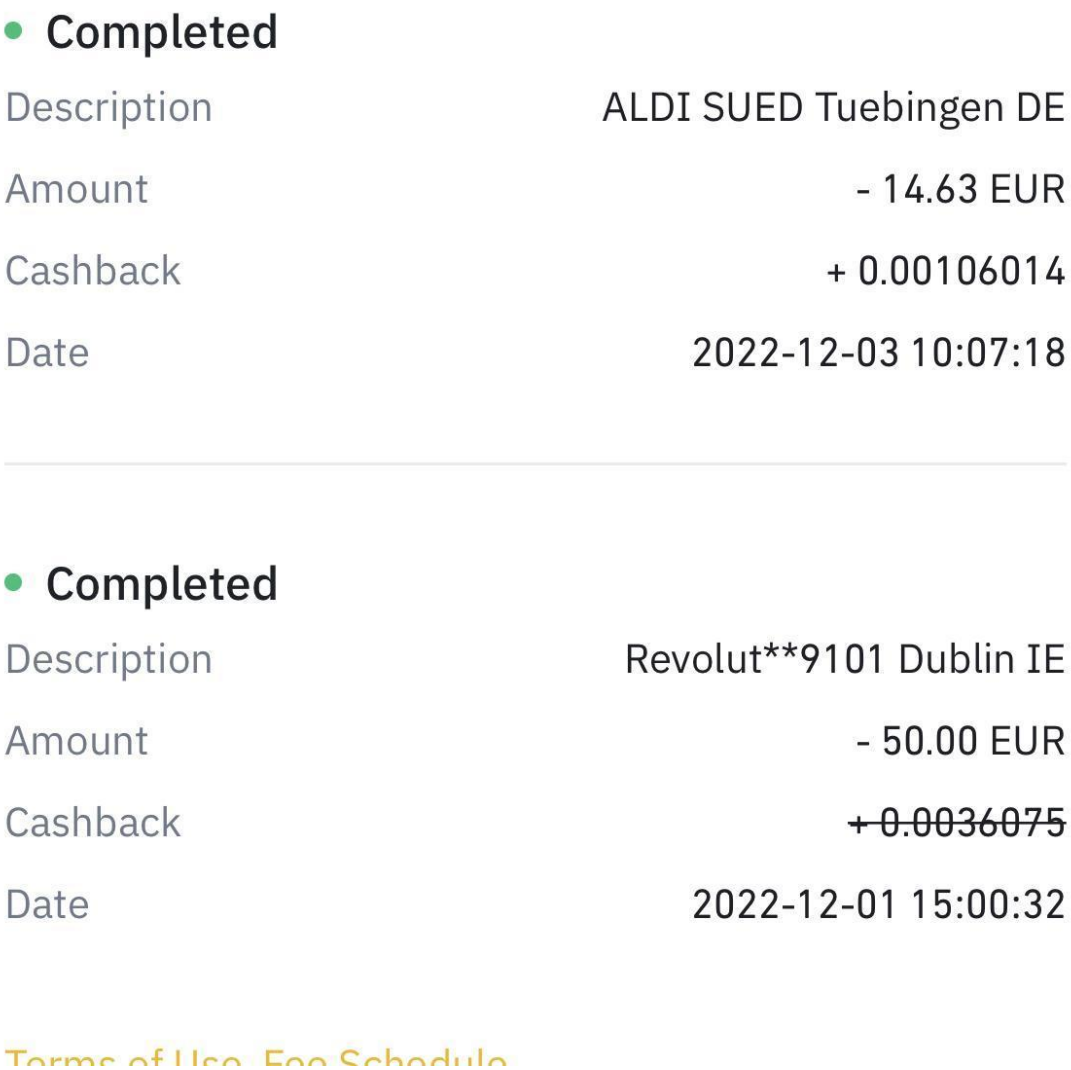

Рисунок 1.2 – Приклад транзакції з використанням біржової картки

Розглянемо стандартні кроки, які зазвичай притаманні для роботи децентралізованої біржі:

1. користувачі створюють облікові записи в децентралізованій біржі, генеруючи відкритий і закритий ключ. Відкритий ключ використовується для

ідентифікації облікового запису користувача в мережі, а закритий ключ використовується для підписання транзакцій і доступу до коштів користувача;

2. користувачі вносять свої криптовалюти в децентралізовану біржу, де вони зберігаються в смарт-контракті на блокчейні. Смарт-контракт діє як цифрове депонування, утримуючи кошти до завершення торгівлі;
3. користувачі можуть шукати та знаходити інших користувачів, які готові торгувати криптовалютами, які вони хочуть купити або продати. Коли вони знайдуть відповідний збіг, вони можуть ініціювати торгівлю, відправивши транзакцію на смарт-контракт децентралізованої біржі;
4. смарт-контракт перевіряє транзакцію та гарантує, що обидві сторони мають необхідні кошти для завершення торгівлі. Якщо транзакція дійсна, вона виконується, а кошти перераховуються від одного користувача до іншого;
5. операція реєструється в блокчейні і обидві сторони можуть переглядати деталі транзакції на платформі децентралізованої біржі або через блокчейн-провідник.

Приклад децентралізованої біржі зображений на рисунку 1.3.

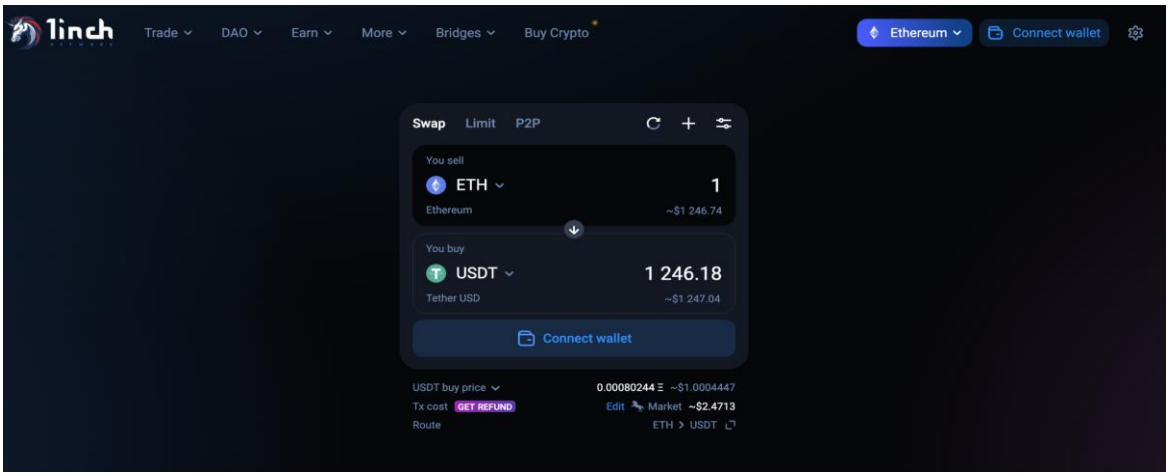

Рисунок 1.3 – Приклад сайту децентралізованої біржі [10]

Інша можливість використовувати криптовалюту як платіж – це створити сайт, на якому буде генеруватися QR код, за допомогою якого можна перевести криптовалюту зі свого гаманця до гаманця продавця [12]. Гарним прикладом є гаманець Phantom, через який можна реалізувати цей інтерфейс.

Гаманець — це цифрова платформа для зберігання та керування криптовалютними активами [11]. Гаманець зазвичай складається з двох основних компонентів: закритого ключа та публічної адреси. Приватний ключ — це секретний код, який дозволяє власнику отримувати доступ до своїх активів у криптовалюті та керувати ними, а публічна адреса — це унікальний ідентифікатор, який можна використовувати для отримання транзакцій від інших користувачів.

Існує кілька різних типів гаманців, які можна використовувати в блокчейні. Деякі приклади:

- програмні гаманці: це гаманці, встановлені на комп’ютері або мобільному пристрої користувача. Зазвичай вони вважаються менш безпечними, ніж інші типи гаманців, оскільки можуть бути вразливими до злому, якщо пристрій користувача не захищено належним чином;
- апаратні гаманці: це фізичні пристрої, призначені для безпечного зберігання криптовалют. Вони часто вважаються найбезпечнішим типом гаманця, оскільки вони не підключені до Інтернету і тому менш вразливі для злому;
- паперові гаманці: це фізичні документи, які містять відкритий і закритий ключі користувача, які можна використовувати для доступу до їхніх криптовалют. Вони вважаються дуже безпечними, оскільки не зберігаються в цифровому вигляді і тому не вразливі для злому;
- веб-гаманці: це онлайн-гаманці, доступ до яких здійснюється через веб-браузер. Вони, як правило, вважаються менш безпечними, ніж інші типи гаманців, оскільки ключі користувача зберігаються на сервері постачальника гаманців і тому вразливі до злому;

- гаманці біржі. Це гаманці, які надаються біржами криптовалют і використовуються для зберігання криптовалют, які купуються та продаються на біржі. Вони, як правило, вважаються менш безпечними, ніж інші типи гаманців, оскільки біржа контролює ключі користувача і тому є вразливою до злому.

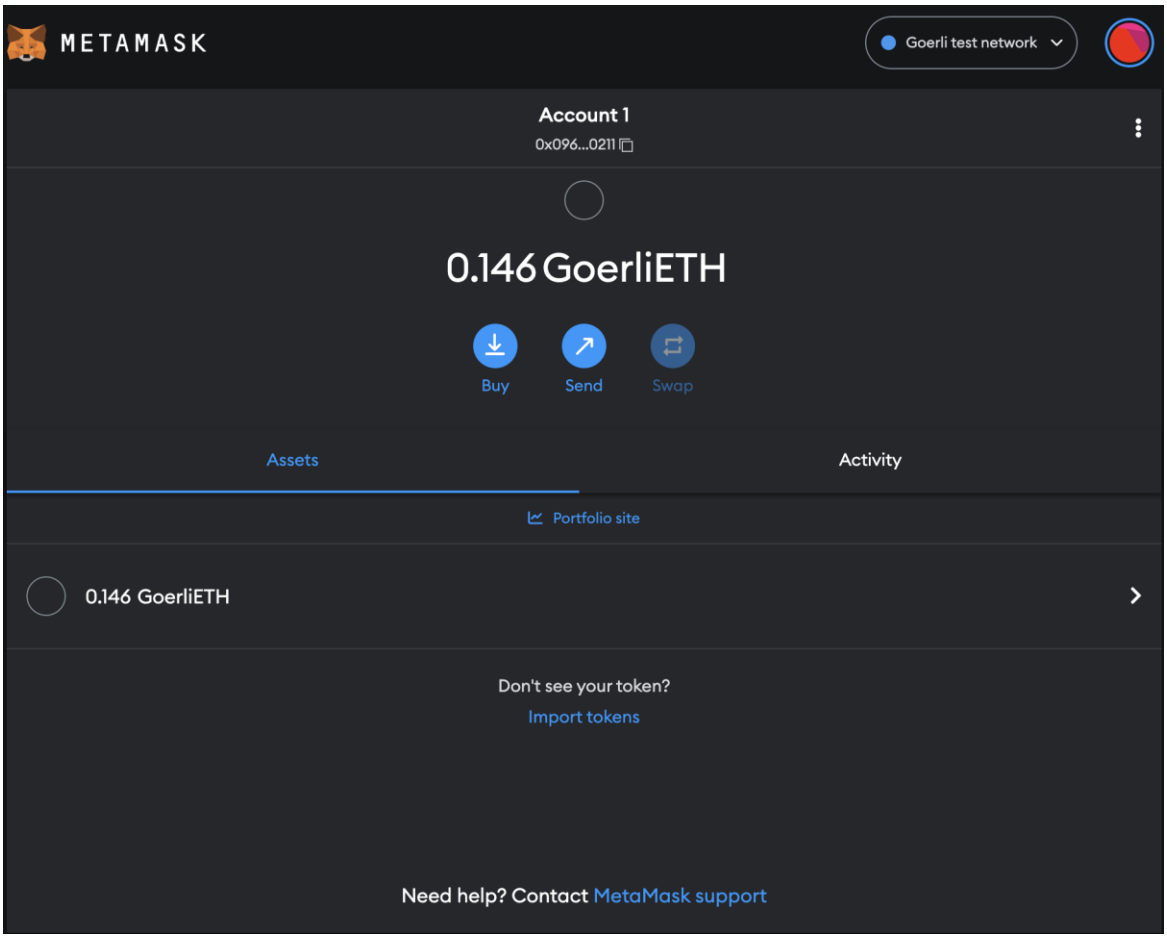

Рисунок 1.4 – Приклад гарячого гаманця

Криптовалютні гаманці можна розділити на дві основні категорії: гарячі гаманці (рис 1.4) та холодні гаманці (рис 1.5). Гарячі гаманці підключені до Інтернету, тому вони більш доступні, але вони також більш уразливі до злому та інших загроз безпеці. Холодні гаманці, з іншого боку, не підключені до Інтернету, тому є більш безпечними, але вони також менш зручні у використанні [4].

Рисунок 1.5 – Приклад холодного гаманця

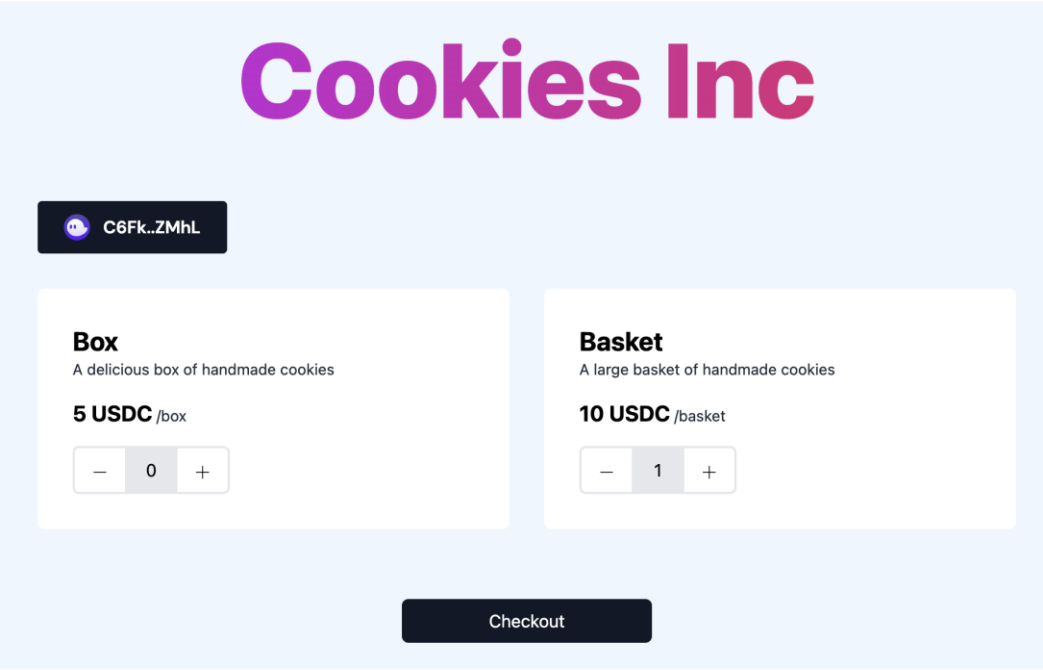

Рисунок 1.6 – Приклад сайту для оплати товару за допомогою гаманця Phantom

Окрім зберігання та керування криптовалютними активами деякі гаманці також дозволяють користувачам надсилати та отримувати транзакції, переглядати історію своїх транзакцій та керувати своїми відкритими та закритими ключами.

Деякі популярні приклади криптовалютних гаманців включають MyEtherWallet, Ledger і Trezor.

Рисунок 1.7 – Створення QR-коду для транзакції

Рисунок 1.8 – Сканування QR-коду за допомогою телефону

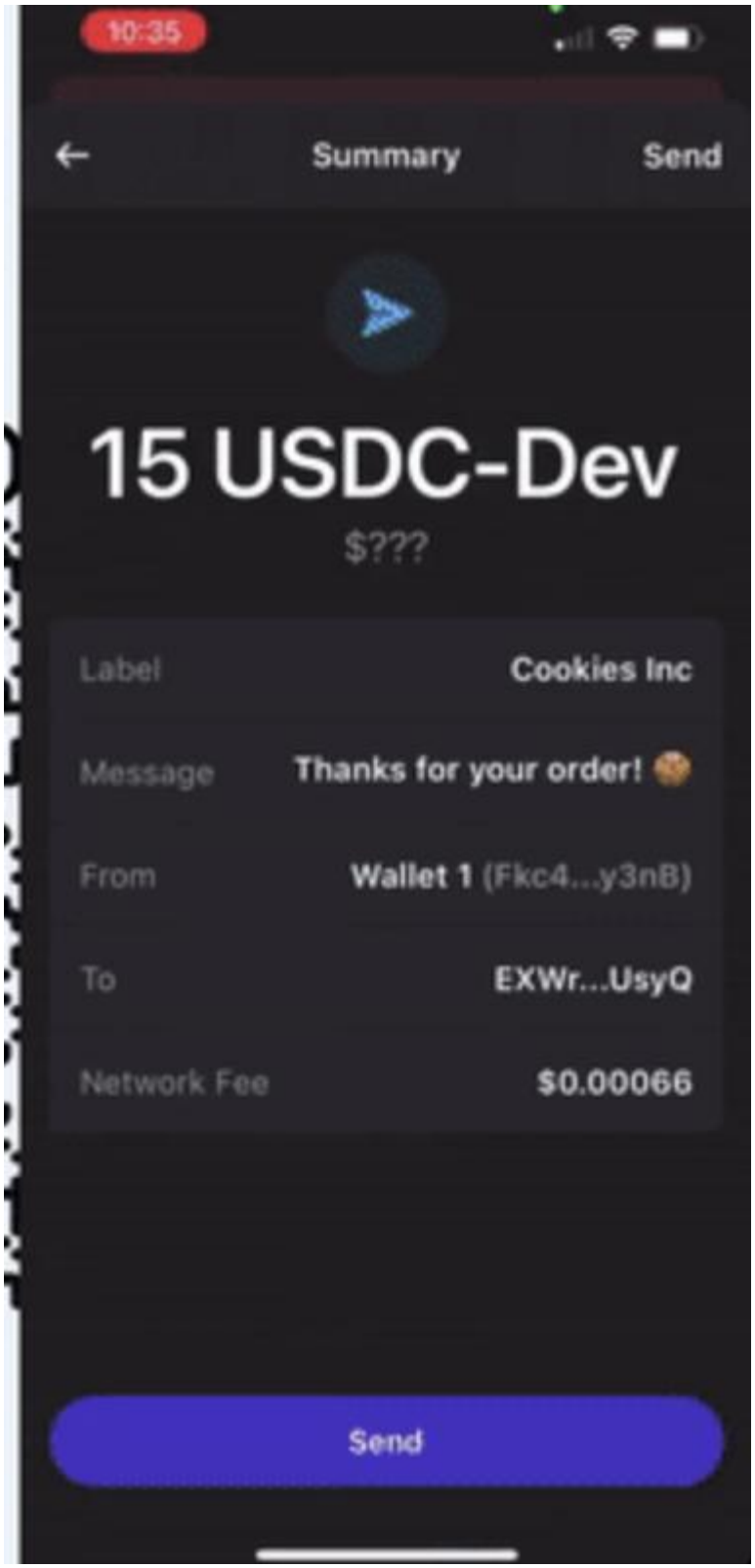

Рисунок 1.9 – Отримання транзакції для оплати товару

1.2 Аналіз проблем реалізації смарт-контрактів

Написання смарт-контрактів є основною задачею для використання блокчейн технології. Смарт-контракти – це програми, що зберігаються в блокчейні, та які запускаються, коли виконуються заздалегідь визначені умови. Зазвичай вони використовуються для автоматизації виконання угоди, щоб усі учасники могли бути миттєво впевнені в результаті, без участі будь-якого посередника чи втрати часу. Вони також можуть автоматизувати робочий процес, запускаючи наступну дію, коли умови виконуються [13].

Але є проблеми, з якими стикаються розробники смарт-контрактів, а саме:

- складність: смарт-контракти часто складні, їх важко розробити, реалізувати та перевірити. Вони вимагають глибокого розуміння базової платформи блокчейну, а також конкретних мов програмування та інструментів, які використовуються для їх створення. Це може ускладнити їх створення і керування для нефахівців, а також збільшити ризик помилок і вразливостей;
- масштабованість: Блокчейн-платформи зазвичай розроблені для обробки обмеженої кількості транзакцій за секунду, що може ускладнити масштабування смарт-контрактів, яким потрібно обробляти великі обсяги даних або обробляти великі обсяги трафіку. Це може призвести до затримок, перевантажень та інших проблем з продуктивністю, які можуть вплинути на зручність використання та надійність;
- безпека: смарт-контракти вразливі до різних типів загроз безпеці, таких як злом, шахрайство та цензура. Ними можуть скористатися зловмисники, які намагаються вкрасти кошти, маніпулювати даними або порушити роботу контракту. Це може поставити користувачів під загрозу втрати своїх активів і може підірвати довіру до смарт-контракту та базової платформи блокчейну;

- взаємодія: різні блокчейн-платформи мають власні унікальні функції, протоколи та стандарти, що може ускладнити взаємодію з іншими платформами або створення крос-ланцюгових смарт-контрактів. Це може обмежити гнучкість і універсальність смарт-контрактів та не дати їм скористатися перевагами кількох мереж блокчейну;
- регулювання: у багатьох юрисдикціях правовий статус смарт-контрактів досі нечіткий і немає встановлених правил чи рамок для регулювання їх використання та функціонування. Це може створити невизначеність і плутанину для користувачів, розробників і регуляторів, а також може перешкодити прийняттю та розвитку смарт-контрактів.

Вивчити технологію блокчейн і стати розробником Solidity може бути складним завданням. Solidity — це мова програмування високого рівня, яка використовується для написання смарт-контрактів для блокчейну Ethereum. Вона має власний синтаксис і правила, які потрібно вивчити [14]. Крім того, сама технологія блокчейн є складною, що створює складнощі для вивчення та розуміння.

Перш ніж вивчати технологію блокчейн, важливо зрозуміти деякі ключові поняття та технології, які з нею пов’язані. Ось деякі аспекти, які користувачі повинні знати, перш ніж користуватися блокчейн [15]:

- криптографія: це практика використання математичних алгоритмів для шифрування та захисту даних. Технологія блокчейн використовує криптографію для захисту транзакцій, які відбуваються в мережі;
- технологія розподіленої книги: це базова технологія, на якій базується блокчейн. Це тип бази даних, яка поширюється в мережі комп’ютерів і дозволяє безпечно та прозоро записувати транзакції;
- механізми консенсусу: це алгоритми та протоколи, які використовуються для досягнення консенсусу щодо стану блокчейну. Різні блокчейн-мережі

використовують різні механізми консенсусу, і розуміння того, як вони працюють, є важливим для розуміння того, як працює блокчейн загалом;

- розумні контракти: це самовиконувані контракти з умовами угоди між покупцем і продавцем, які безпосередньо записані в рядках коду. Вони є важливою частиною багатьох додатків блокчейну, і розуміння того, як вони працюють, має важливе значення для розуміння того, як можна використовувати блокчейн;
- децентралізація: це ключова особливість технології блокчейн. Це стосується того факту, що мережа не контролюється одним суб'єктом, а натомість підтримується мережею користувачів, у кожного з яких є копія книги. Розуміння децентралізації є важливим для розуміння переваг і обмежень технології блокчейн.

Загалом, вивчення цих концепцій і технологій забезпечить міцну основу для розуміння того, як працює блокчейн і як його можна використовувати.

1.3 Постановка задачі

На сьогодні платіжні системи, які використовують блокчейн технології, роблять лише оплату криптовалютою, без використання додаткових можливостей для стимулювання користувачів використовувати альтернативний варіант оплати. У цій роботі буде представлено систему, за допомогою якої бізнес може підключити можливість створювати та обробляти свої платежі, використовуючи токени на системі блокчейн.

За допомогою невзаємозамінюваного токена бізнес може відстежувати та видавати або робити додаткові знижки для активних користувачів, або ж, якщо бізнес має свій токен, то відправляти відсоток від суми кожної покупку.

Система буде налічувати в собі контракт токену, контракт-фабрика, за допомогою якого бізнес створює колекцію з представленого контракту невзаємозамінюваного токену, та контракт оплати, який бізнес буде використовувати для створення транзакцій для оплати. Схема роботи представлена на рисунку 1.10.

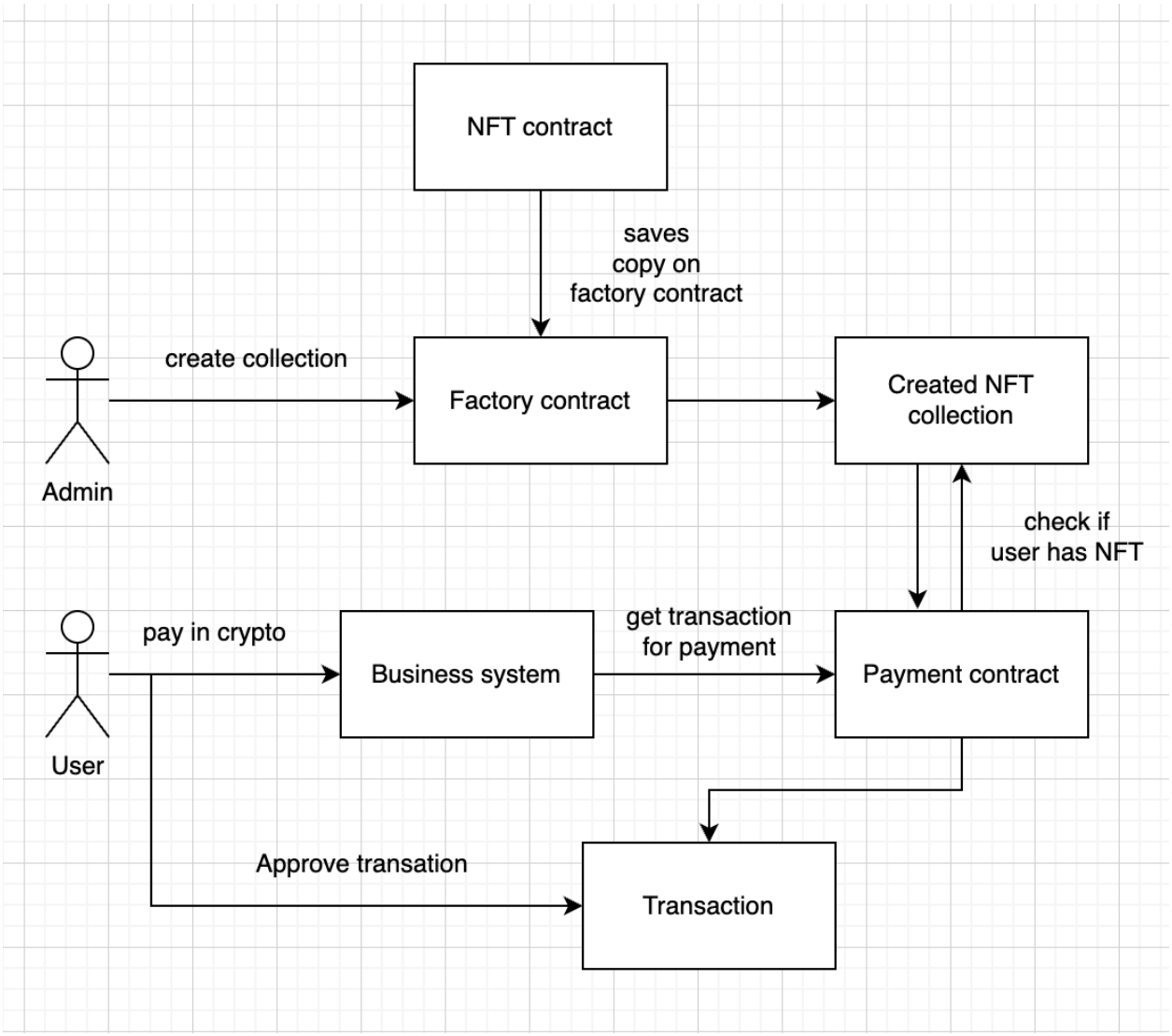

Рисунок 1.10 – Схема роботи представленої системи

Висновки по розділу І

Технологія блокчейн може революціонізувати платіжні системи, зробивши їх швидшими, безпечнішими та ефективнішими. Блокчейн — це децентралізована розподілена книга, яка записує транзакції на кількох комп'ютерах, тому запис не можна змінити заднім числом без згоди мережі. Це робить його дуже безпечним і прозорим способом відстеження транзакцій. У платіжній системі, що використовує технологію блокчейн, транзакції будуть записуватися та перевірятися на блокчейні, що дозволить майже миттєво переказувати кошти без необхідності в посередниках, таких як банки. Це може значно зменшити комісію за транзакції та зробити процес оплати ефективнішим. Крім того, оскільки записи блокчейну є незмінними та прозорими, вони можуть забезпечити додатковий рівень безпеки та запобігання шахрайству.

Смарт-контракт — це комп'ютерна програма, яка автоматично виконує умови контракту, коли виконуються певні умови. У мережі блокчейн розумні контракти можна використовувати для сприяння, перевірки та забезпечення виконання переговорів або виконання контракту. Однак створення та впровадження смарт-контрактів на блокчейні може бути складним завданням з кількох причин.

Однією з головних проблем є те, що смарт-контракти часто дуже складні та вимагають високого ступеня точності в кодуванні. Це може ускладнити їх написання та тестування, і навіть невелика помилка в коді може мати серйозні наслідки. Крім того, оскільки смарт-контракти незмінні та не можуть бути змінені після розгортання в блокчейні, важливо переконатися, що вони пройшли ретельне тестування та перегляд перед розгортанням.

Ще одна проблема з використанням смарт-контрактів у блокчейні полягає в тому, що ця технологія все ще є відносно новою та розвивається. Це означає, що ще

немає чітких стандартів або найкращих практик для створення та використання смарт-контрактів, а також бракує досвідчених розробників і ресурсів підтримки. Крім того, різні блокчейн-платформи мають різні можливості та обмеження, що може ускладнити розробку смарт-контрактів, які можна використовувати на кількох платформах.

Загалом, незважаючи на те, що використання смарт-контрактів у блокчейні має потенціал для значного підвищення ефективності та безпеки договірних угод, перед тим, як його можна буде широко впровадити, необхідно вирішити значні проблеми.

2. РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПРОВЕДЕННЯ РОЗРАХУНКІВ ЗА ТОВАРИ ТА ПОСЛУГИ З ВИКОРИСТАННЯМ НЕВЗАЄМОЗАМІНЮВАНОГО ТОКЕНА

2.1 Концепція проведення розрахунків за товари та послуги з використанням невзаємозамінюваного токена

Платіжна система, яка використовує криптовалюту як активи та незамінні токени (NFT) як засіб ідентифікації та надання переваг, працюватиме, дозволяючи користувачам заробляти та використовувати NFT як винагороду за використання платіжної системи. У представленій системі користувачі можуть отримувати знижку та заробляти бали, якщо вони мають токен на своєму гаманці. Щоб здійснити платіж, користувач просто конвертує свою криптовалюту в потрібну форму оплати (наприклад, біткойн, ефір тощо), а система перевіряє, чи має користувач потрібний токен, щоб отримати будь-які доступні переваги. Ця система пропонує позитивний і корисний досвід для користувачів, а також дозволяє створити децентралізовану платіжну мережу. Однак важливо зазначити, що використання NFT у такий спосіб все ще є відносно новою технологією, що розвивається, тому можуть виникнути деякі проблеми та обмеження, про які слід знати [16].

Забігаючи наперед, обмеження у цій системі представлені тим, що для бізнесу воно не пропонує розширення своїх можливостей. Але кожна компанія може побачити код контракту та переписати його під свій лад, щоб ті можливості, яких немає в основному контракті, були відображені.

Структура запропонованої системи складається з трьох основних контрактів:

- Контракт невзаємозамінюваного токена **NftBase**
- Контракт-фабрика **Factory**
- Контракт для отримання оплати **Payment**

Принцип роботи системи дуже простий і не потребує великих знань у сфері блокчейн. Перший етап (рис 2.1) – це створення колекції, яку бізнес буде використовувати. Для цього довірча особа повинна на контракті-фабрики викликати функцію для створення колекції, записавши туди назву та символ колекції, гаманець адміністратора, який зможе керувати контрактом токену, та гаманець, який буде отримувати плату за товар. Адміністратор має можливість змінити адрес, на який будуть відбуватися перекази.

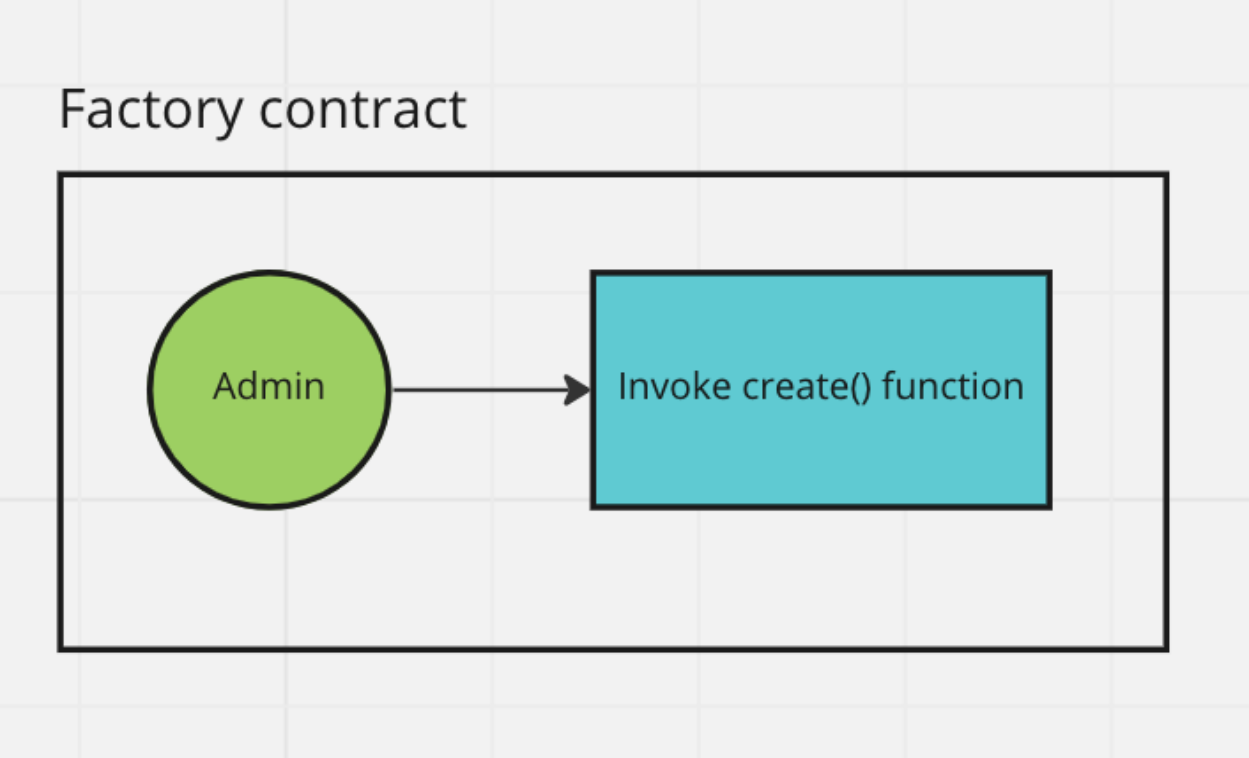

Рисунок 2.1 – Схема першого етапу

Другим етапом (рис 2.2) є підключення контрактів до бізнес системи. Це може бути сайт, на якому користувач підключається за допомогою свого гаманця, а коли підходить час до оплати, то система всередині викликає функцію в контракті для отримання оплати, де є номер замовлення, ціна у токені, який вибрав користувач, адрес контракту-колекції та адрес токену, якщо це ненативний токен блокчейну. Після цього користувач отримає транзакцію, яку він повинен підтвердити, тоді

контракт знімає плату з користувача та передає токени до адреси, яку адміністратор вказав на контракті токену, щоб отримувати плату.

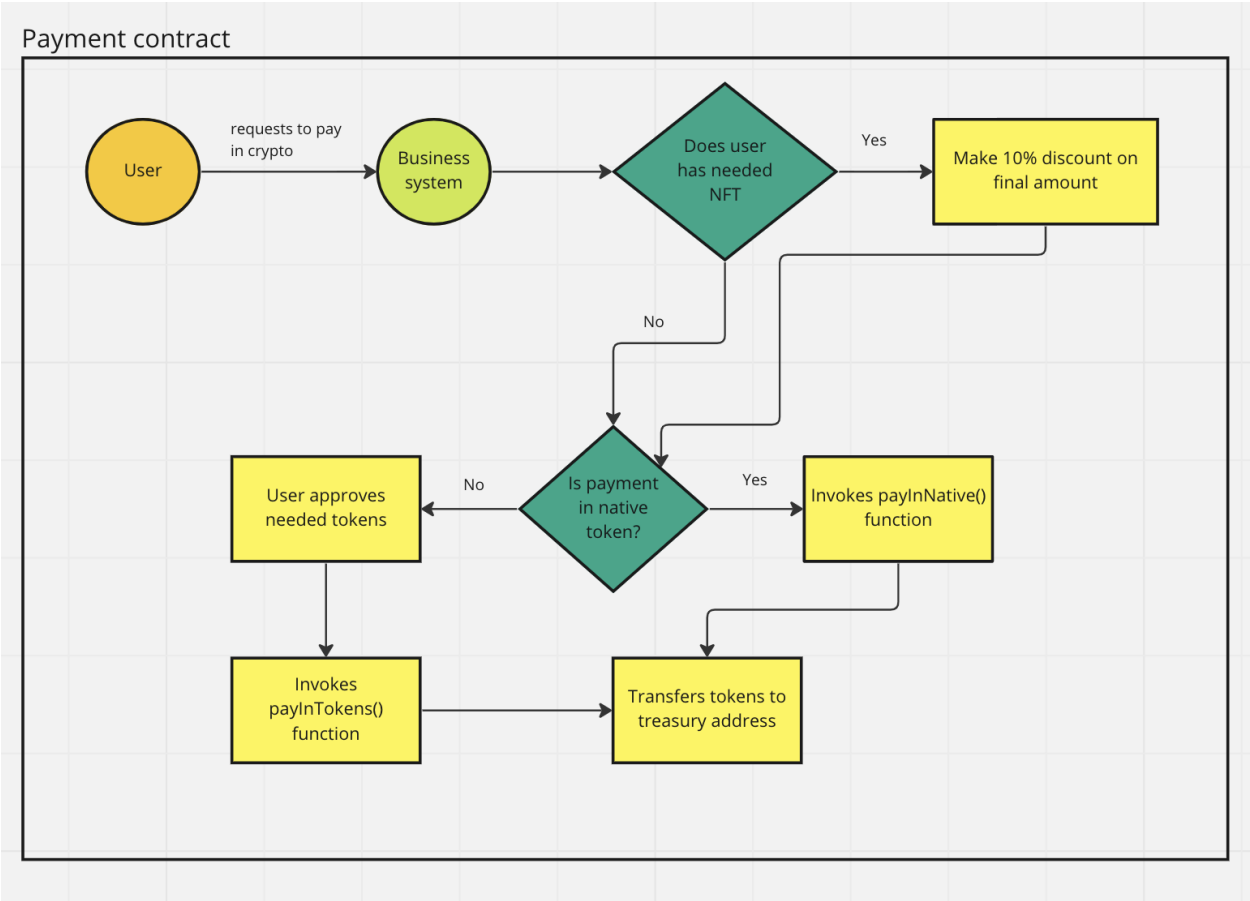

Рисунок 2.2 – Схема другого етапу

2.2 Реалізація смарт-контракту невзаємозамінюваного токена для використання у платежах

Для початку треба створити контракт невзаємозамінюваного токена, який буде використовуватись на контракті-фабрики. Основним для цього контракту буде протокол ERC721.

ERC721 — це стандарт для невзаємозамінюваних токенів у блокчейні Ethereum [17]. Ці токени являють собою унікальні активи, які не можна розділити або відтворити, наприклад, предмети колекціонування або унікальні цифрові

активи. ERC721 токени унікальні та мають індивідуальні властивості та характеристики. Це робить їх придатними для використання в програмах, які включають унікальні активи, такі як цифрові предмети колекціонування або внутрішньоігрові предмети комп'ютерних ігор.

У кожного стандарту є вимоги, яким має відповідати контракт, щоб його можна було вважати таким, який відповідає стандарту і належить до контрактів його групи. У випадку невзаємозамінюваного токена ці вимоги включають підтримку унікальних ідентифікаторів для кожного токена, можливість передачі токенів між адресами та можливість перевірки балансу токенів, що належать певній адресі. Щоб не прописувати та не робити помилок для створення основних вимог у контракті, використаємо готові контракти з платформи OpenZeppelin.

OpenZeppelin — це платформа з відкритим вихідним кодом для створення смарт-контрактів на блокчейні Ethereum [3]. Він надає набір попередньо написаних, протестованих і перевірених компонентів смарт-контракту, які розробники можуть використовувати як відправну точку для своїх власних контрактів. Це може допомогти заощадити час і зменшити ризик уразливостей, дозволяючи розробникам будувати свої контракти на основі безпечного перевіреного коду, а не починати з нуля. OpenZeppelin також надає інструменти для тестування та розгортання смарт-контрактів, а також форум спільноти для обговорення найкращих практик та обміну знаннями.

За допомогою OpenZeppelin легко створити свій токен, систему голосування або можливість оновлювати контракти. На платформі є велика документація по кожному контракту, який можна використовувати в різних цілях, починаючи від простих по типу стандартів та закінчуючи криптографією та безпекою (рис 2.3).

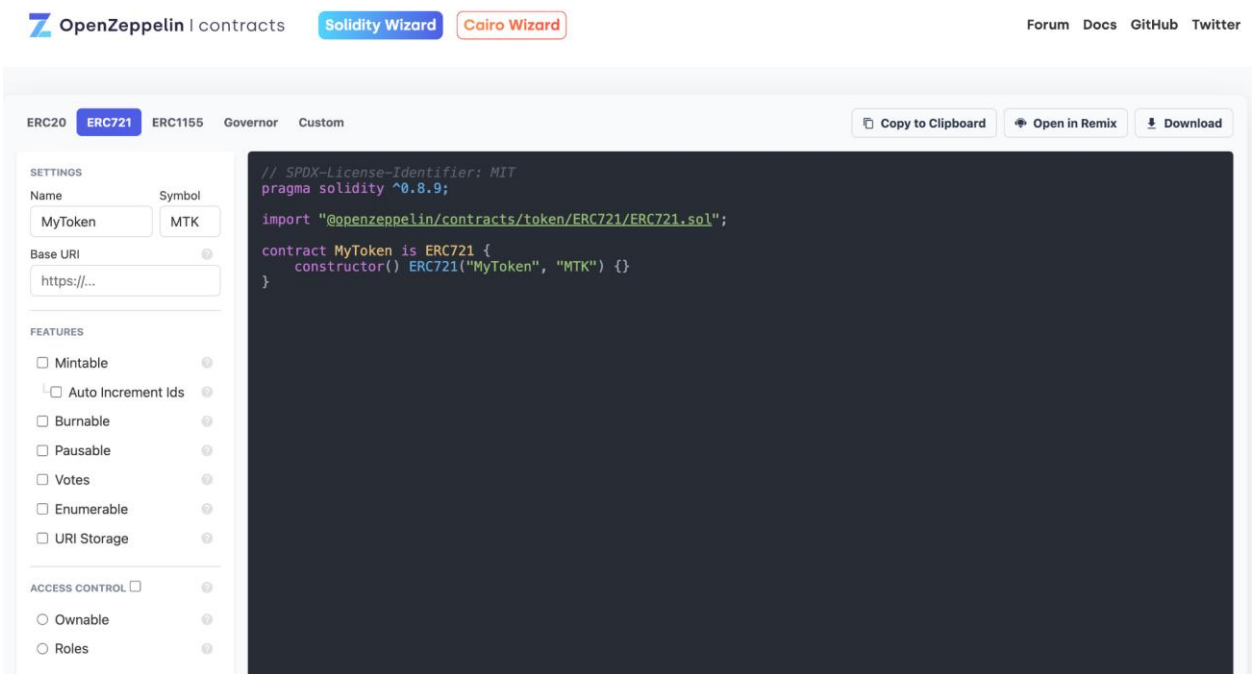

Рисунок 2.3 – Приклад конструктора на платформі OpenZeppelin [2]

Маючи основу на базі стандарту ERC721, треба додати функціонал, який включає у себе те, що користувач може мати тільки один токен, неможливість передавати токен іншим особам та запис адреси, на яку будуть передаватись гроші за транзакцію.

Для реалізації логіки того, що користувач може мати тільки один токен можна записати таким чином:

```
function safeMint() public {
    if (balanceOf(msg.sender) == 1) revert OnlyOneForUser();
    uint256 tokenId = _tokenIdCounter.current();
    _tokenIdCounter.increment();
    _safeMint(msg.sender, tokenId);
}
```

Функція, яка називається safeMint, призначена для того, щоб створювати токен, який будемо використовувати при оплаті товару. Щоб користувач зміг його

отримати, він повинен викликати цю функцію та підтвердити транзакцію. Після цього виконується функція, яка:

- перевіряє, чи має вже користувач цей токен. Якщо має, то транзакція не пройде та видасть помилку OnlyOneForUser і користувач не отримає новий токен;
- якщо користувач немає токену, то береться ідентифікатор токену, який зберігається на контракті, та створюється токен за допомогою функції _safeMint.

У разі успішної транзакції користувач повинен отримати токен на свій гаманець.

Реалізація неможливості передачі токену полягає у тому, щоб переписати функцію трансферу, заборонивши будь-які спроби викликати її. Приклад коду:

```
function _transfer(
    address from,
    address to,
    uint256 tokenId
) internal override {
    revert NotTransferable();
}
```

Можна побачити ключове слово override, що означає перепис функції, яка наслідується з контракту ERC721. Все, що додаємо у цю функцію, це помилку NotTransferable, що не дає можливості виконати транзакцію.

Останній функціонал, який залишився, це зміна або запис адреси, на яку будуть приходити кошти від оплати. Фінальна функція виглядає ось так:

```
function setTreasury(address payable _treasury) external onlyOwner {
    if (_treasury == address(0)) revert ZeroAddress();
    treasury = _treasury;
```

```
        emit NewTreasury(_treasury);
    }
```

Основний момент полягає у модифікаторі onlyOwner, який являє собою перевірку того, щоб цю функцію міг викликати тільки власник контракту. Далі йде перевірка, щоб адреса, яку власник хоче записати, не була нульовою, бо тоді через це всі кошти будуть просто зніматися з користувача та знищуватись.

Готовий контракт токену виглядає таким чином:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.17;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/Counters.sol";

error NotTransferable();
error OnlyOneForUser();
error ZeroAddress();

contract NftBase is ERC721, Ownable {
    using Counters for Counters.Counter;

    address public collectionAddr;
    address payable public treasury;

    Counters.Counter private _tokenIdCounter;

    event NewTreasury(address payable _treasury);
```

```
    constructor(string memory _name, string memory _symbol, address _owner,
address payable _treasury) ERC721(_name, _symbol) {
        _transferOwnership(_owner);
        collectionAddr = address(this);
        treasury = _treasury;
    }

    function safeMint() public {
        if (balanceOf(msg.sender) == 1) revert OnlyOneForUser();
        uint256 tokenId = _tokenIdCounter.current();
        _tokenIdCounter.increment();
        _safeMint(msg.sender, tokenId);
    }

    function setTreasury(address payable _treasury) external onlyOwner {
        if (_treasury == address(0)) revert ZeroAddress();
        treasury = _treasury;

        emit NewTreasury(_treasury);
    }

    function getTreasury() public view returns(address) {
        return treasury;
    }

    function _transfer(
```

```
        address from,
        address to,
        uint256 tokenId
    ) internal override {
        revert NotTransferable();
    }
}
```

Після створення бази токену, можна перейти до створення контракту-фабрики. Основний функціонал контракту полягає у тому, щоб створювати копії контракту, який передається у функції. Є два варіанти створити копію контракту. Перший – це за допомогою ключового слова new. Але тоді не буде відома адреса, яку отримає контракт. Другим варіантом є використання операції create2.

Код операції create2 використовується для створення смарт-контракту в блокчейні Ethereum [18]. Це дозволяє укласти договір за конкретною адресою, яку можна визначити заздалегідь. Це може бути корисно з різних причин, наприклад, дозволити створення контракту з передбачуваною адресою. Це може бути корисно з метою безпеки, оскільки зловмисникам буде складніше передбачити адресу контракту та потенційно скомпрометувати його. Крім того, використання create2 також може допомогти заощадити газ, оскільки його можна використовувати, щоб уникнути проблеми порожнього облікового запису, коли контракт створюється за адресою, яка раніше використовувалася, але тепер порожня. Це може призвести до більш ефективного створення контракту, оскільки це дозволяє уникнути необхідності чистити порожній рахунок. Готова функція виглядає так:

```
    function create2(
        string memory _name,
        string memory _symbol,
        address _owner,
```

```
        address payable _treasury,
        bytes32 _salt
    ) public {
        NftBase collection = (new NftBase){salt: _salt}(_name, _symbol, _owner,
_treasury);
        collections.push(collection);
    }
```

Основним у цій функції є передача всіх аргументів, які потрібні для створення колекції та так званий «синтаксичний цукор», через який можна отримати заздалегідь адресу контракту. Адресу записуємо у змінну під назвою collections, що являє собою список всіх адрес створених контрактів.

Фінальний код контракту-фабрики:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.17;

import "./NftBase.sol";

contract Factory {
    NftBase[] public collections;

    function create2(
        string memory _name,
        string memory _symbol,
        address _owner,
        address payable _treasury,
        bytes32 _salt
    ) public {
```

```
        NftBase collection = (new NftBase){salt: _salt}(_name, _symbol, _owner,
_treasury);
        collections.push(collection);
    }

    function getCollection(uint _index)
        public
        view
        returns (
            address owner,
            address collectionAddr
        )
    {
        NftBase collection = collections[_index];

        return (collection.owner(), address(collection));
    }
}
```

Отже, на даний момент маємо створені основні контракти, тепер залишилось створити контракт, який буде приймати оплату. Оплату можна буде приймати як у нативному токені, так і токені стандарту ERC20.

ERC20 — це стандарт для створення токенів у блокчейні Ethereum [20]. Він визначає набір правил, яким має відповідати контракт на токен, щоб вважатися токеном ERC20. Ці правила включають такі речі, як те, що токен може бути переданий з однієї адреси на іншу, правила керування загальним запасом токенів і надання можливостей користувачам отримати доступ до інформації про токен, такої як його назва, символ і баланс.

Щоб мати можливість взаємодіяти з ERC20 стандартом, треба імпортувати в контракт інтерфейс IERC20, за допомогою якого можна буде викликати функції токенів, такі як трансфер. Так само треба зробити й для невзаємозамінюваного токена, який використовується при оплаті.

Буде створено дві функції, одна з яких буде відповідати за отримання нативного токену, а інша – за стандарт ERC20 токенів. Аргументами для функцій будуть слугувати адреса невзаємозамінюваного токена, кількість токенів, які треба передати, якщо це ERC20 стандарт, то адресу токену та ідентифікатор квитанції, яка генерується системою. Функції виглядають таким чином:

```
function payInNative(address _nft, uint256 _amount, uint256 receitId) public payable {
        if (msg.value != _amount) revert WrongAmount();
        address _treasury = NftBase(_nft).getTreasury();

        if (NftBase(_nft).balanceOf(msg.sender) == 1) {
            _amount = _amount * 90 / 100;
            uint256 returnAmount = _amount * 10 / 100;
            payable(_treasury).transfer(_amount);
            payable(msg.sender).transfer(returnAmount);
        } else {
            payable(_treasury).transfer(_amount);
        }

        emit Payed(_nft, address(0), _amount, receitId, block.timestamp);
    }
```

```
        function payInTokens(address _nft, address _token, uint256 _amount,
uint256 receitId) public {
            if (IERC20(_token).allowance(msg.sender, address(this)) < _amount)
revert LowAllowance();

            if (NftBase(_nft).balanceOf(msg.sender) == 1) {
                _amount = _amount * 90 / 100;
                IERC20(_token).safeTransferFrom(msg.sender,
NftBase(_nft).getTreasury(), _amount);
            } else {
                IERC20(_token).safeTransferFrom(msg.sender,
NftBase(_nft).getTreasury(), _amount);
            }

            emit Payed(_nft, _token, _amount, receitId, block.timestamp);
        }
```

Опишемо кожну функцію. Перша з назвою payInNative відповідає за оплату у нативному токені. Йде перевірка, щоб кількість токенів відповідала кількості, яка буде передана до контракту у змінній msg.value. Далі отримуємо адресу, на яку треба буде відправити токени, та записуємо її як _treasury. Після цього перевіряємо, чи є у користувача невзаємозамінюваний токен, який використовується при оплаті. Якщо є, то надаємо знижку у розмірі 10% та передаємо цю суму до адреси, яку записали у змінну _treasury.

Друга функція з назвою payInTokens відповідає за оплату у токенах стандарту ERC20. Логіка схожа на payInNative функцію, тільки у цій функції перевіряється дозвіл на передачу токенів від користувача, тому що для того, щоб контракт міг отримати токени від користувача, користувач на контракті токену повинен дати

дозвіл на використання його грошей цим контрактом. Якщо дозвіл є, то знову ж таки, перевіряється, чи є у користувача невзаємозамінюваний токен. Після цього токени передаються до адреси, яка записана у невзаємозамінюваному токені для передачі оплати.

У кінці всіх маніпуляцій створюємо подію під назвою Payed, за допомогою якої у майбутньому можна буде простежити за транзакціями, які були виконані на контракті оплати. У події зберігається інформація, така як адреса невзаємозамінюваного токену, адреса токену, яким оплачували, ідентифікатор квитанції та час, коли була проведена транзакція. Повний код контракту:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.17;

import "./NftBase.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";

error LowAllowance();
error WrongAmount();

contract Payment {
    using SafeERC20 for IERC20;

    event Payed(address _nft, address _token, uint256 _amount, uint256
indexed receitId, uint256 timestamp);

    function payInNative(address _nft, uint256 _amount, uint256 receitId)
public payable {
```

```
        if (msg.value != _amount) revert WrongAmount();
        address _treasury = NftBase(_nft).getTreasury();

        if (NftBase(_nft).balanceOf(msg.sender) == 1) {
            _amount = _amount * 90 / 100;
            uint256 returnAmount = _amount * 10 / 100;
            payable(_treasury).transfer(_amount);
            payable(msg.sender).transfer(returnAmount);
        } else {
            payable(_treasury).transfer(_amount);
        }

        emit Payed(_nft, address(0), _amount, receitId, block.timestamp);
    }
    function payInTokens(address _nft, address _token, uint256 _amount,
uint256 receitId) public {
        if (IERC20(_token).allowance(msg.sender, address(this)) < _amount)
revert LowAllowance();

        if (NftBase(_nft).balanceOf(msg.sender) == 1) {
            _amount = _amount * 90 / 100;
            IERC20(_token).safeTransferFrom(msg.sender,
NftBase(_nft).getTreasury(), _amount);
        } else {
            IERC20(_token).safeTransferFrom(msg.sender,
NftBase(_nft).getTreasury(), _amount);
        }
```

```
        emit Payed(_nft, _token, _amount, receitId, block.timestamp);
    }
}
```

2.3 Проектування та реалізація програмного додатку для проведення платежів із використання розробленого контракту. Тестування інформаційної технології

Запропонована система працює таким чином, що у додатку може бути два способи оплати: стандартний за допомогою банківської картки або за допомогою криптовалюти. Якщо користувач обирає оплатити за допомогою криптовалюти, то система конвертує суму у той токен, який вибрав користувач, та відправляє інформацію до контракту-платежу, де створюється транзакція на оплату. Схема роботи розробленої системи представлено на рисунку 2.4.

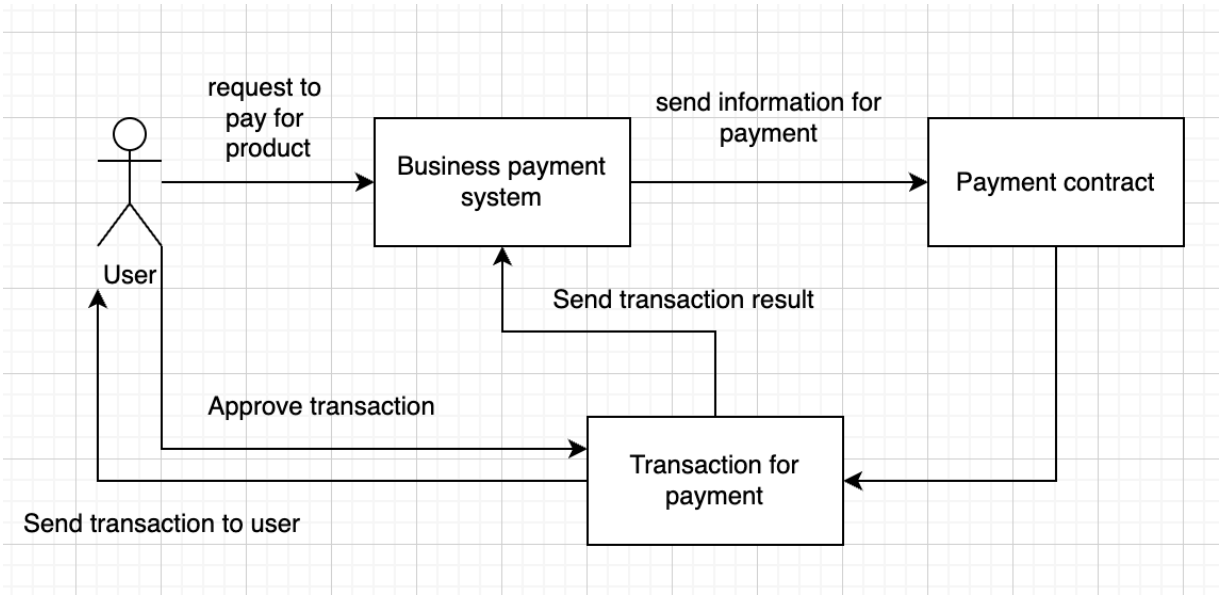

Рисунок 2.4 – Схема роботи системи платежів з використанням розробленого контракту

Наприклад, якщо користувач хоче придбати ноутбук, який коштує дві тисячі доларів, та хоче оплатити у токені Ethereum, який, наприклад, торгується за ціною тисячу двісті доларів за один токен, то система робить конвертацію 2000/1200 = 1.66, та відправляє транзакцію покупцю.

Після розробки контрактів, їх треба протестувати перед тим, як використовувати у реальних сценаріях. Бо якщо десь логіка не відповідає вимогам, які були надані, то це може призвести до краху всієї системи.

Для тестування використовується фреймворк hardhat, у якому є вбудований фреймворк mocha – тестування на мові JavaScript. Гарною практикою є те, щоб кожний контракт протестувати окремо, а потім створити різні сценарії, через які, потенційно, контракти можуть поводити себе не так, як того треба.

Після написання всіх тестових варіантів, треба перевірити, чи весь контракт покритий тестами та чи всі функції були викликані. Список тестів, які були проведені:

Factory

Basic

✓ Create contract (62ms)

NftBase

Basic

✓ Mint token

✓ Should revert if user try to transfer token

✓ Should revert if user try to mint second token

✓ Should revert if user try to set treasury address

✓ Should revert if owner try to set treasury address to zero

Payment

User scenario

✓ Create contract

✓ Add nft contract, set treasury and mint one nft

✓ Pay in Native token

✓ Pay in ERC20 token (44ms)

✓ Revert when try to pay in native with less amount of tokens

✓ Revert when try to pay in tokens with low allowance

Після запуску команди npx hardhat coverage, отримуємо табличку з покриттям (рис 2.5), щоб перевірити себе, чи всі функції та частини контакту були перевірені.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
Factory.sol	100	100	100	100	
NftBase.sol	100	100	100	100	
Payment.sol	100	100	100	100	
contracts/mock/	100	100	100	100	
Token.sol	100	100	100	100	
All files	100	100	100	100	

Рисунок 2.5 – Таблиця покриття наданих контрактів

Коли все протестовано та розробник впевнений, що все правильно працює, контракти треба розгорнути у блокчейні. Для цього використовується той самий фреймворк hardhat. Для розгортання контрактів треба написати скрипт з усіма потрібними аргументами. У випадку системи з трьох контрактів розгортаються тільки два, це контракт-фабрика та контракт для отримання оплат, оскільки контракт невзаємозамінюваного токена використовується як копія, яку контракт-фабрика буде використовувати для створення.

Для розгортання буде використовуватись тестова мережа блокчейну Goerli. Після написання скрипту, ми отримуємо адреси контрактів:

```
npx hardhat run scripts/deploy.ts --network goerli
Factory deployed to 0xD2AaC2594673315b075F7fbdE6609E742098bBDa
Payment deployed to 0x13316652C51E522cE4Fb9AEe4dc13bE77e14f03D
```

Для перевірки того, що контракти дійсно знаходяться на блокчейні, використаємо сайт EtherScan. Etherscan — це блокчейн-провідник для блокчейну Ethereum [20]. Він дозволяє користувачам шукати, переглядати та перевіряти транзакції та іншу інформацію в блокчейні. Etherscan можна використовувати для перегляду інформації про конкретну адресу гаманця, транзакцію або блокування в блокчейні Ethereum. Це корисний інструмент для тих, хто хоче дослідити блокчейн Ethereum або для розробників, які працюють зі смарт-контрактами на платформі Ethereum.

Якщо зайти на адресу, через яку розгорталися контракти, то можна побачити перелік, представлений на рис. 2.5.

Рисунок 2.5 – Транзакції створення контрактів

За допомогою Etherscan можна взаємодіяти з контрактами, що дає можливість перевірити, чи правильно все працює. Спробуємо створити колекцію, яку будемо використовувати для платежів. Для цього треба перейти до контракту Factory та зайти у розділ Contract -> Write Contract (рис 2.6). Для взаємодії треба під'єднати гаманець, яким будемо підписувати транзакцію.

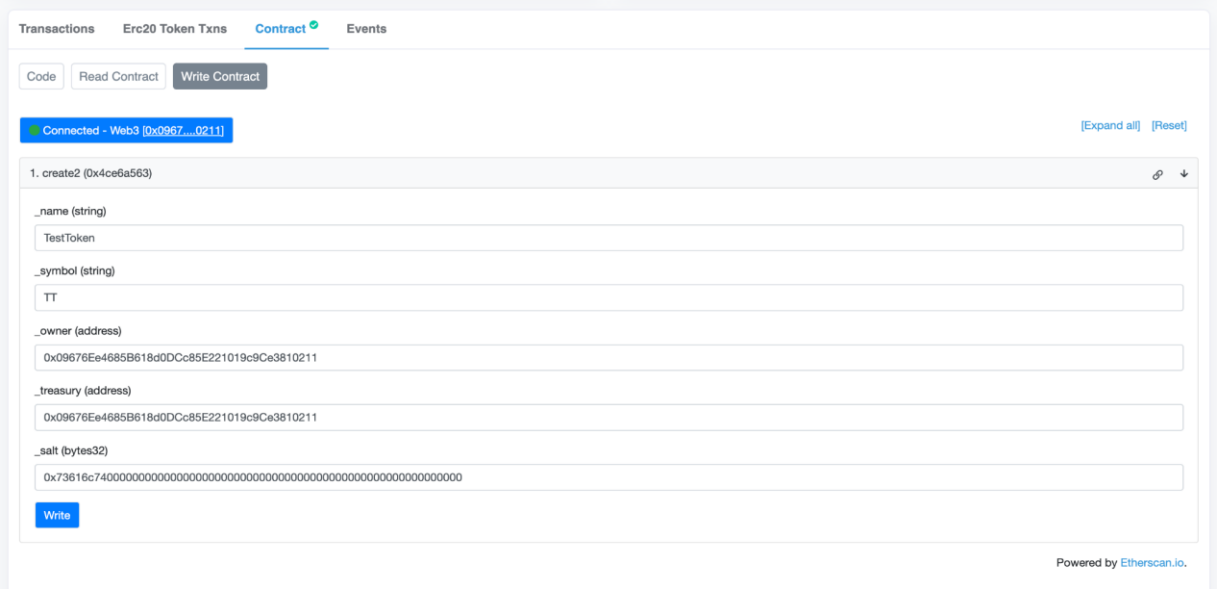

Рисунок 2.6 – Взаємодія з контрактом-фабрикою для створення колекції

Після запису всіх потрібних аргументів, натискаємо кнопку Write та отримуємо транзакцію, яку треба підписати (рис 2.7). Виконану транзакцію можна побачити на рисунку 2.8.

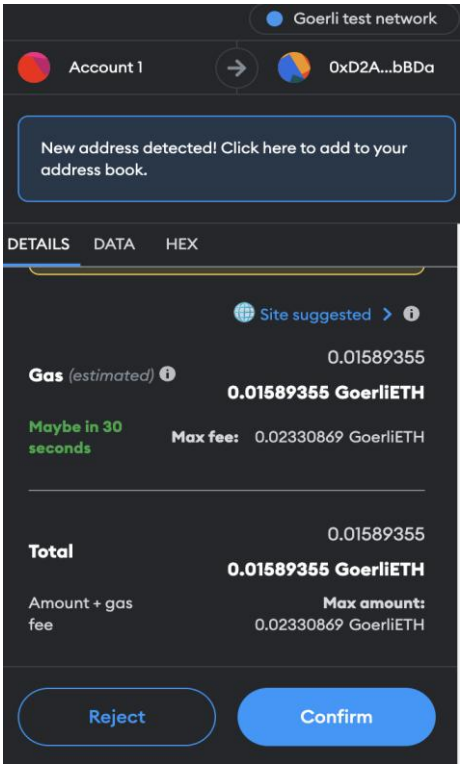

Рисунок 2.7 – Транзакція на створення колекції

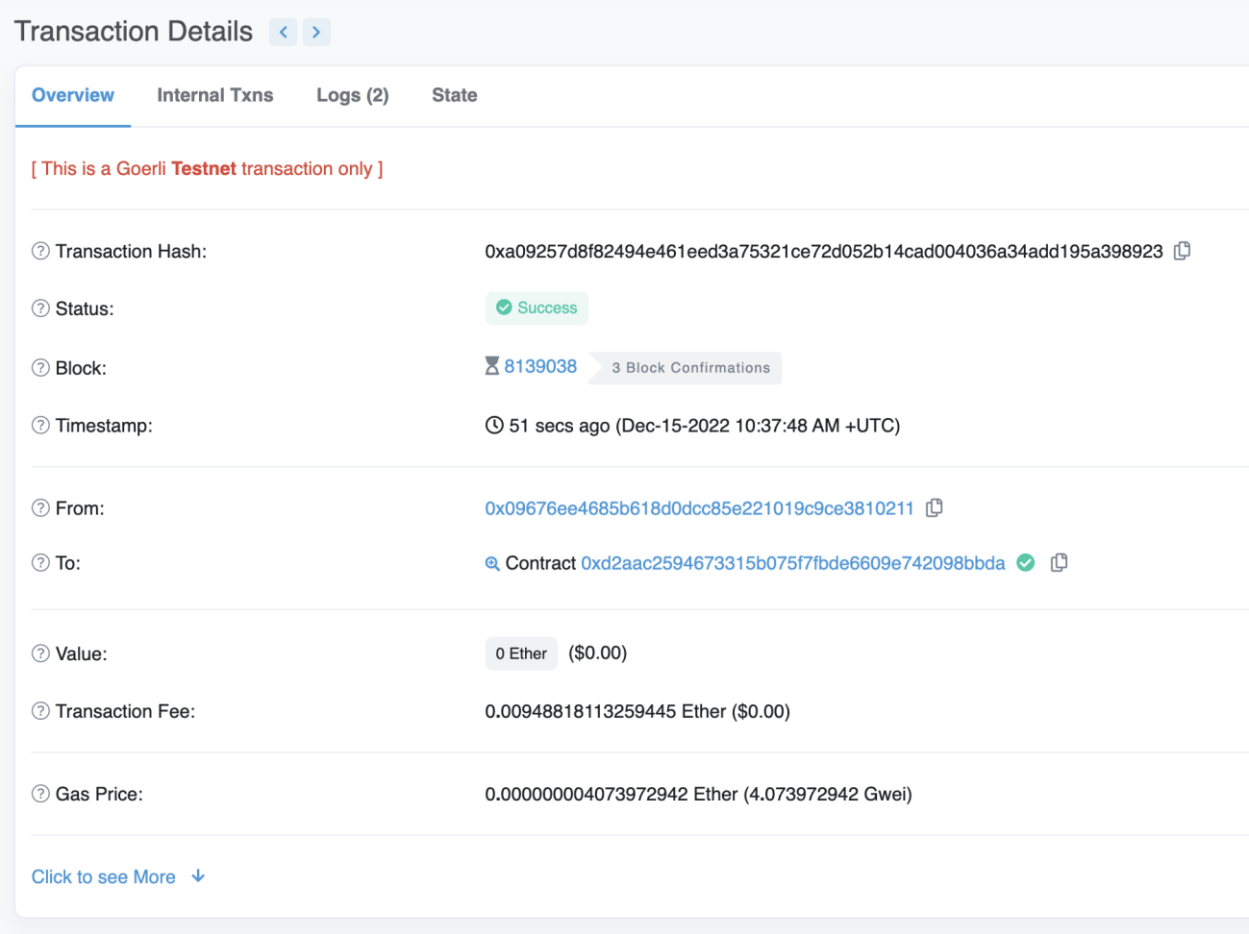

Рисунок 2.8 – Виконана транзакція на створення колекції

Транзакція пройшла успішно, а значить колекція була створена. Є дві можливості отримати адресу контракту, який було створено. Перший, це зайти до Logs у транзакції, і там буде адреса нашого контракту. Або зайти назад до контракту-фабрики, зайти до Contract -> Read Contract та викликати функцію getCollection (рис 2.9), де аргументом є індекс. Так як це перший контракт, який було створено, то адреса знаходиться під індексом 0.

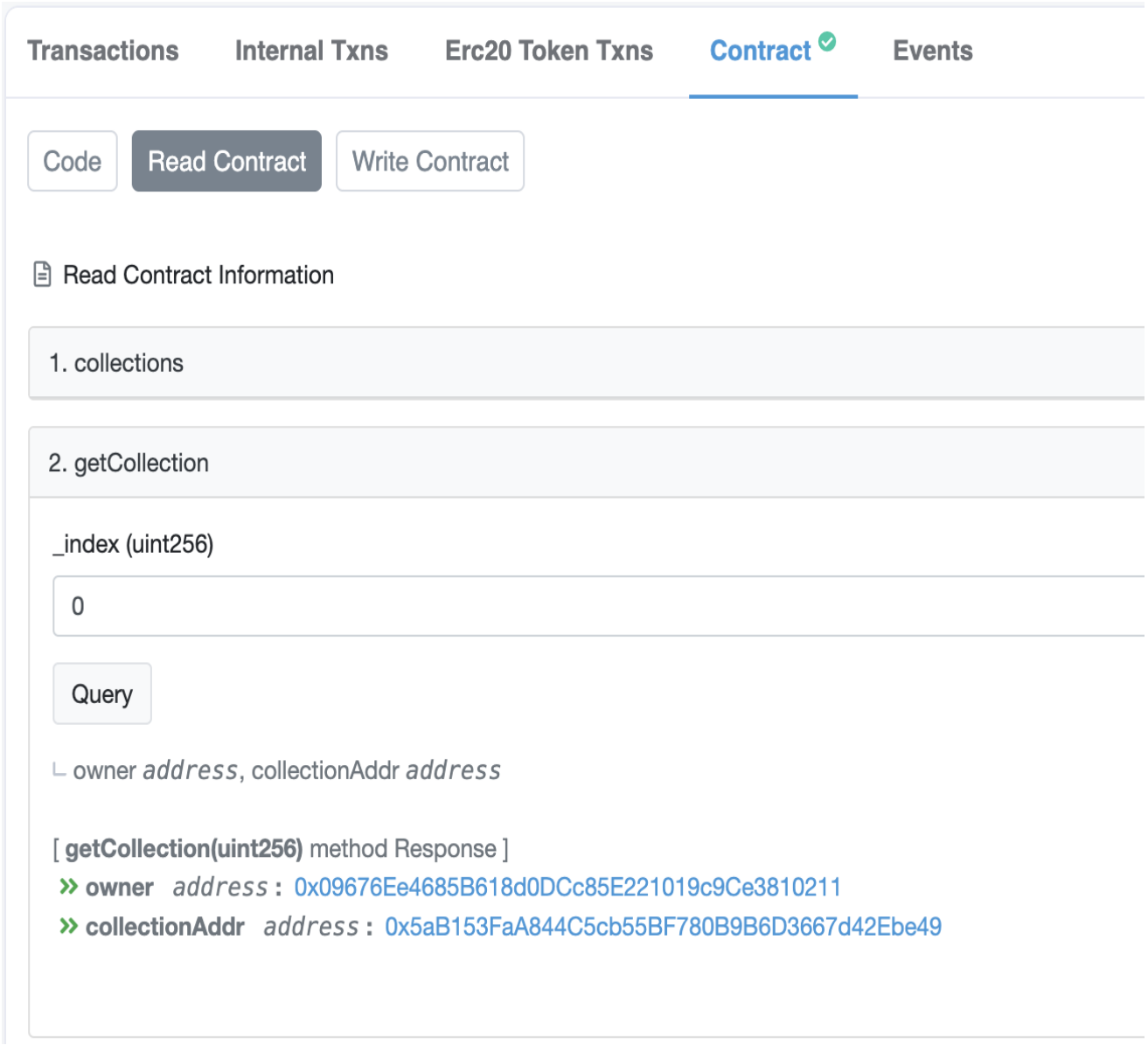

Рисунок 2.9 – Виклик функції getCollection для адреси колекції

Як бачимо, 0x5aB153FaA844C5cb55BF780B9B6D3667d42Ebe49 є адреса колекції, яку було створено через контракт-фабрику. Переходимо до контракту невзаємозамінюваного токена та створюємо собі токен (рис 2.10).

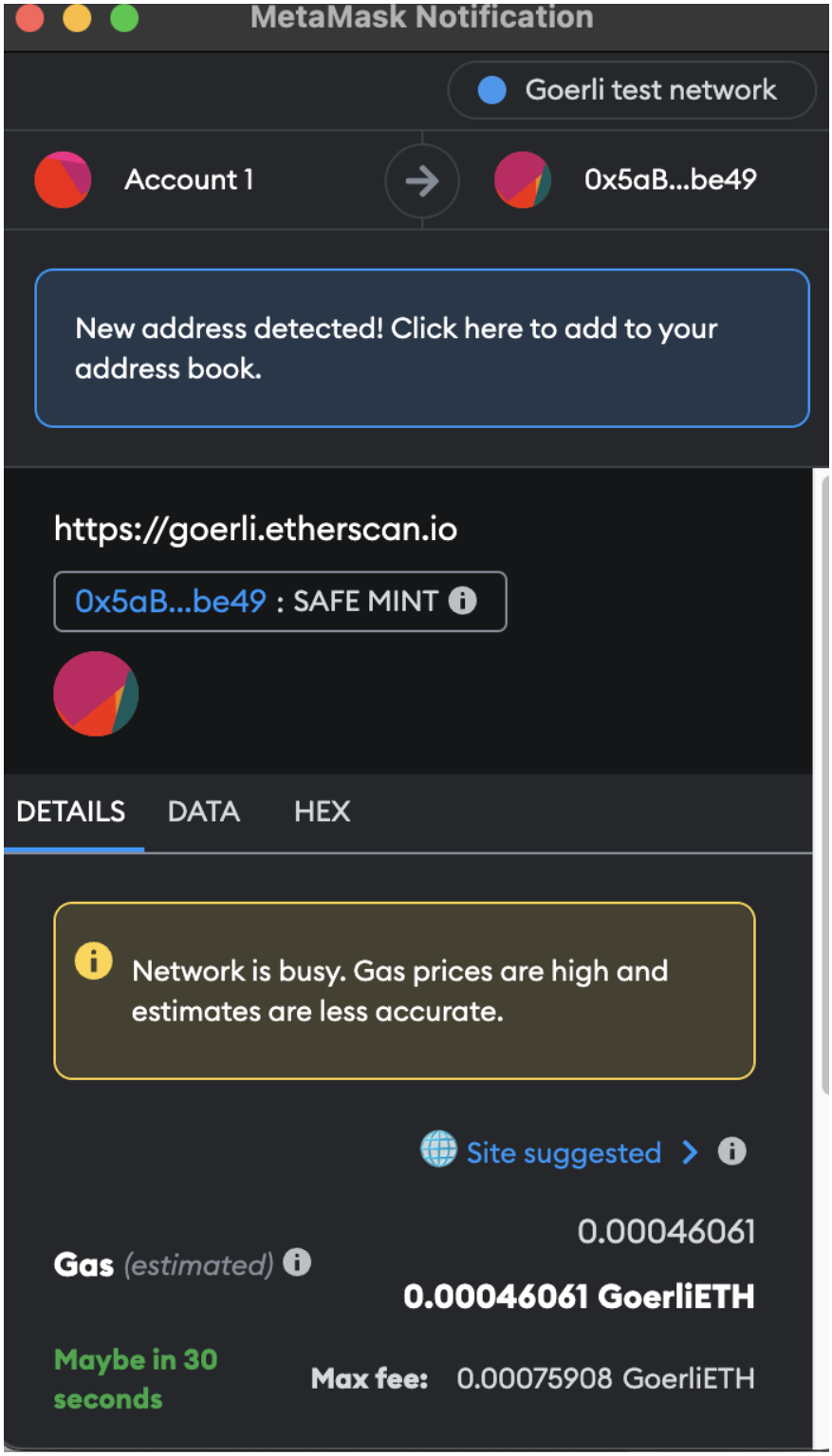

Рисунок 2.10 – Транзакція на створення невзаємозамінюваного токена

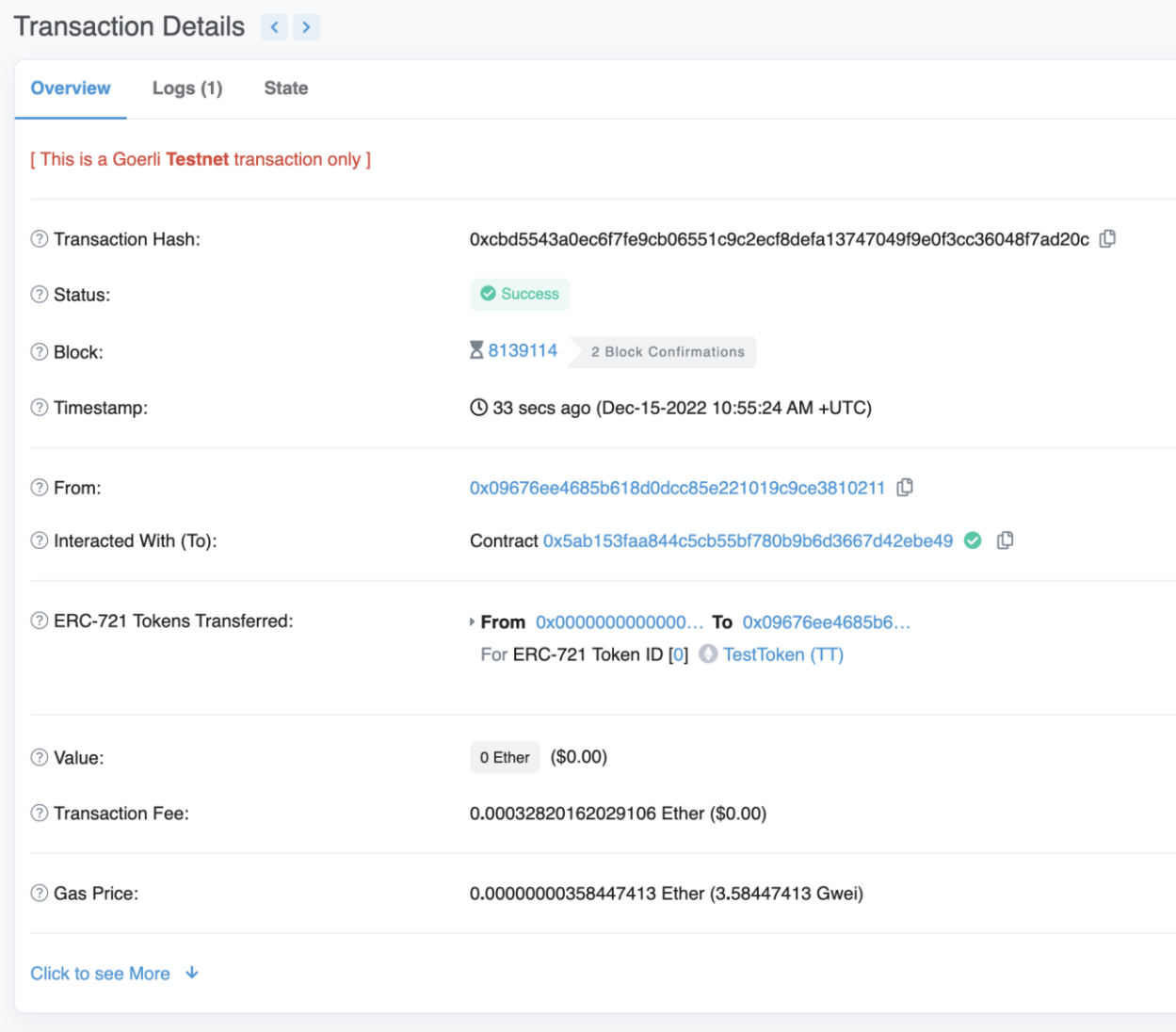

Рисунок 2.11 – Виконана транзакція на створення невзаємозамінюваного токена

Як можна побачити на рисунку 2.11, було отримано токен з індексом 0 на адресу, якою підтверджували транзакцію. Для того, щоб перевірити, що акаунт дійсно володіє цим токеном, можна зайти на сайт OpenSea та подивитись, що у нього є цей токен.

OpenSea — це платформа для купівлі та продажу цифрових товарів у блокчейні Ethereum [21]. Він часто використовується для купівлі та продажу унікальних цифрових активів, таких як предмети колекціонування, віртуальна нерухомість і цифрове мистецтво. Оскільки ці транзакції реєструються в блокчейні

Ethereum, вони безпечні та прозорі. OpenSea є популярним місцем для тих, хто цікавиться зростаючим світом цифрових активів на основі блокчейну. Створену колекцію через контракт-фабрику можна побачити на рисунку 2.12.

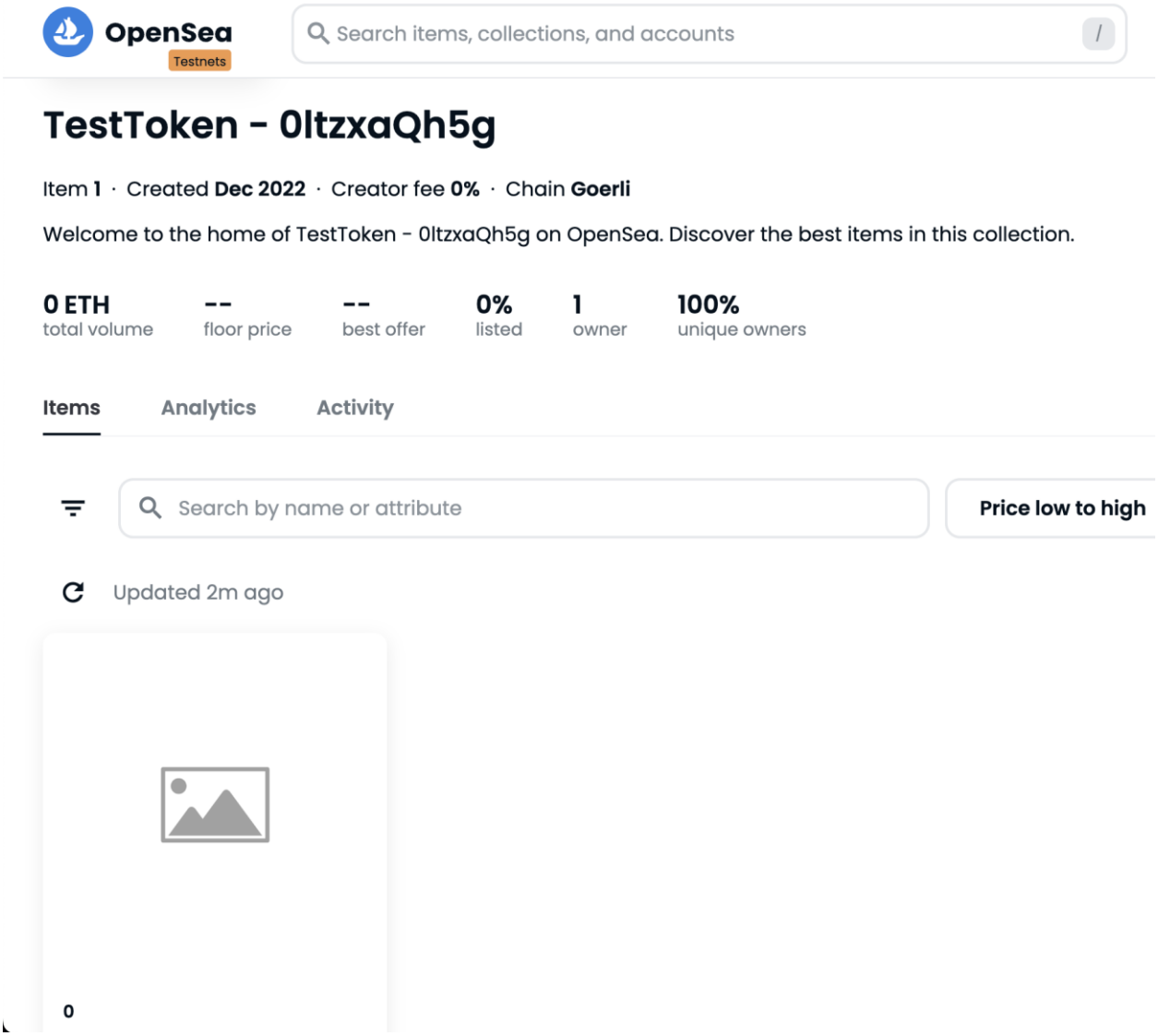

Рисунок 2.12 – Створена колекція на маркетплейсі OpenSea

Колекцію створено, а значить можна протестувати як працює платіжна система. Для цього створюємо новий адрес, на який будуть приходити токени, та запишемо його до контракту колекції. Адреса, яка буде використовуватись:

0x4e730bef6b7d5f69c3802aD0a1Bbd72A37911a63. Для підтвердження, що на цю адресу будуть приходити кошти, подивимось на функцію getTreasury() на контракті колекції (рис 2.13).

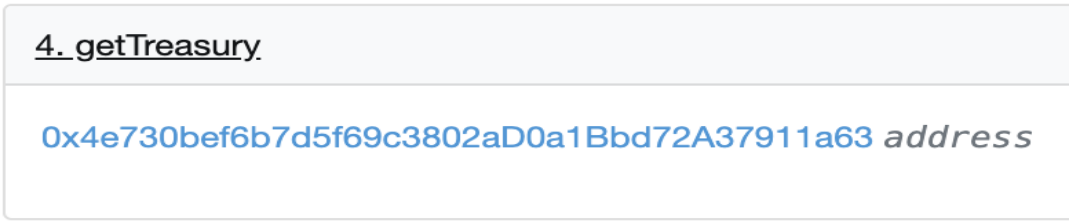

Рисунок 2.13 – Результат функції getTreasury

Переходимо до контракту для оплати та створюємо транзакцію, яку будемо оплачувати (рис 2.14). Створимо тестовий токен, за допомогою якого будемо оплачувати. Сума, яку будемо оплачувати буде сто токенів.

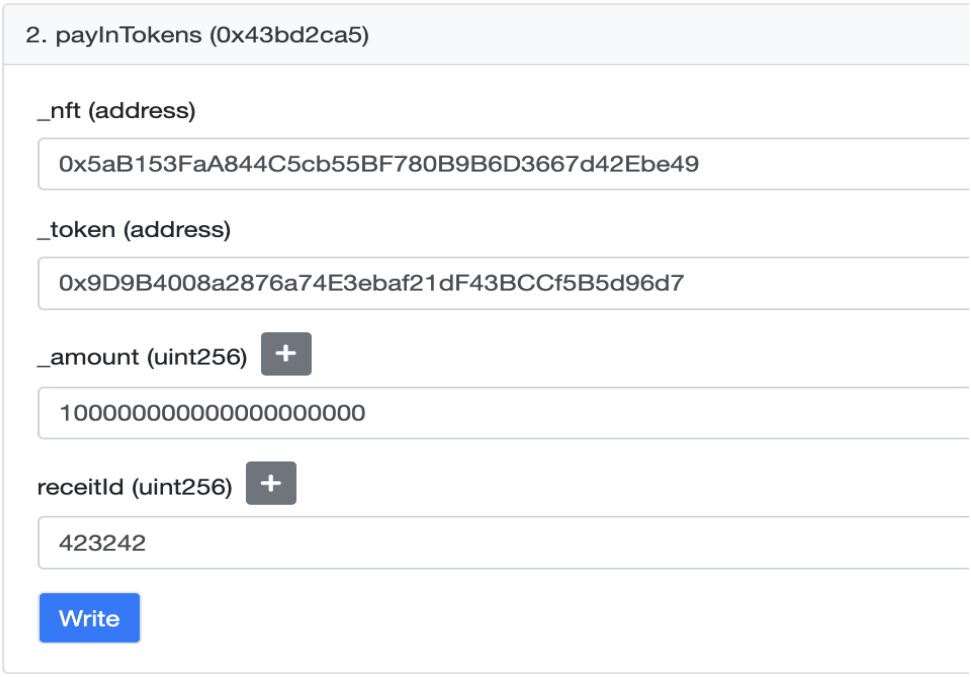

Рисунок 2.14 – Створення транзакції для оплати

Якщо спробувати підписати транзакцію, то вийде помилка, тому що не надано контракту доступу до токенів (рис 2.15). Через це і буде помилка.

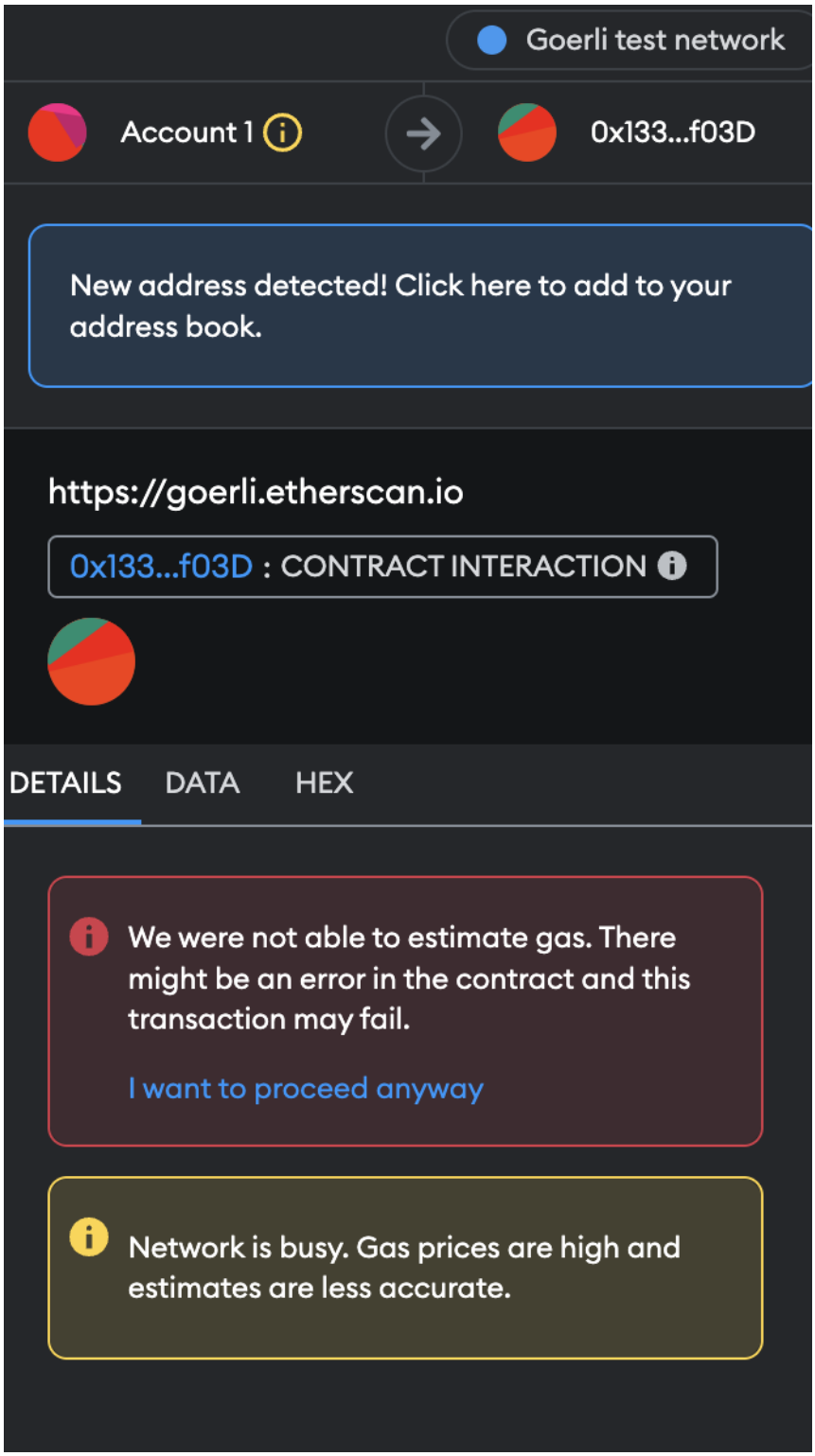

Рисунок 2.15 – Невдала транзакція оплати

Якщо надати доступ, то транзакція пройде. Для цього треба зайти на контракт токену та викликати функцію approve(), де аргументами виступають контракт,

якому надають доступ, та кількість токенів. Якщо знову викликати транзакцію оплати та підтвердити її, то отримуємо результат.

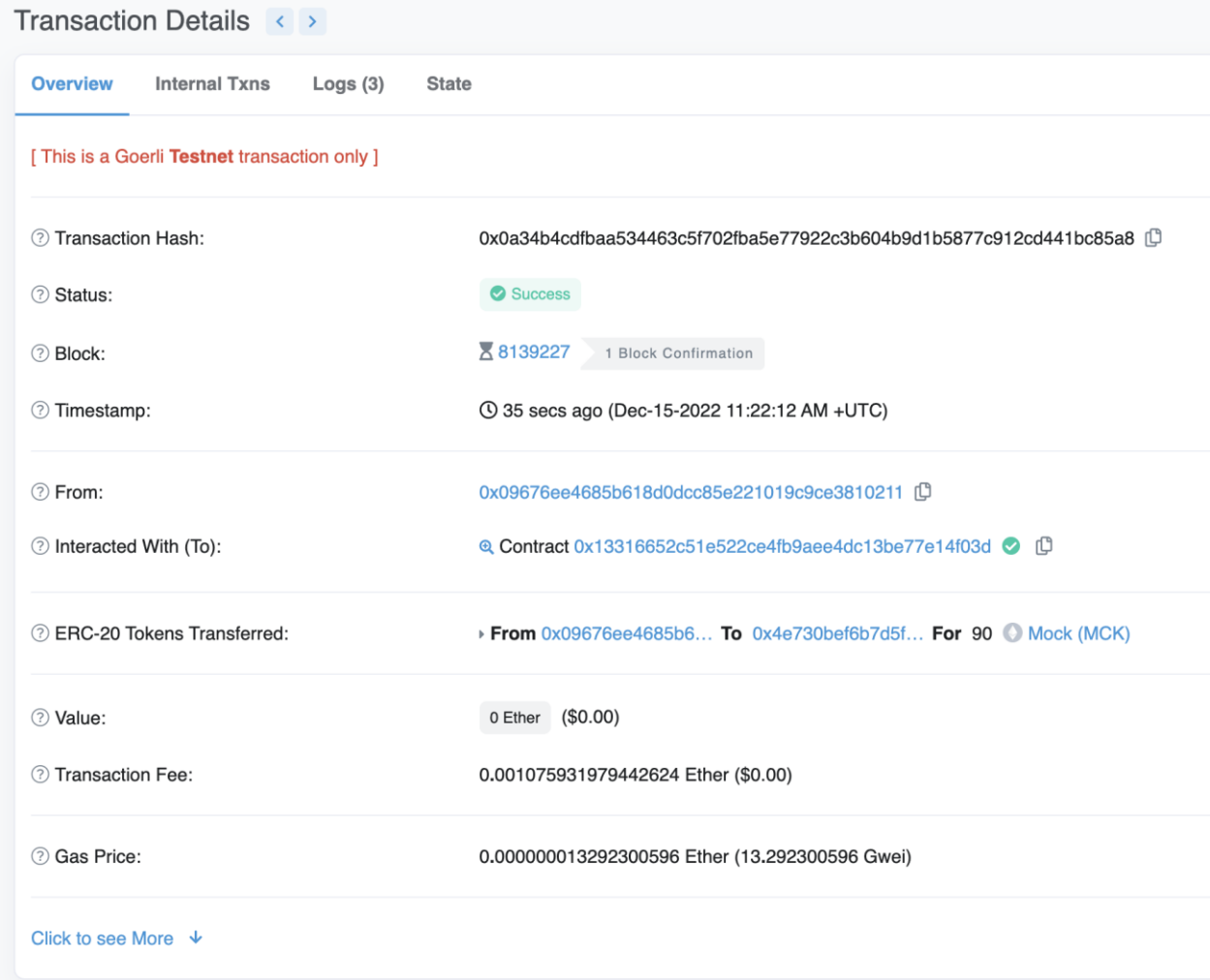

Рисунок 2.16 – Виконана транзакція оплати товару

Як видно на рисунку 2.16, транзакція пройшла. Так як у гаманці був невзаємозамінюваний токен, ми отримали знижку 10%, через те відправлено 90 токенів замість 100. За допомогою EtherScan перевіримо, чи дійсно токени є на адресі (рис 2.17).

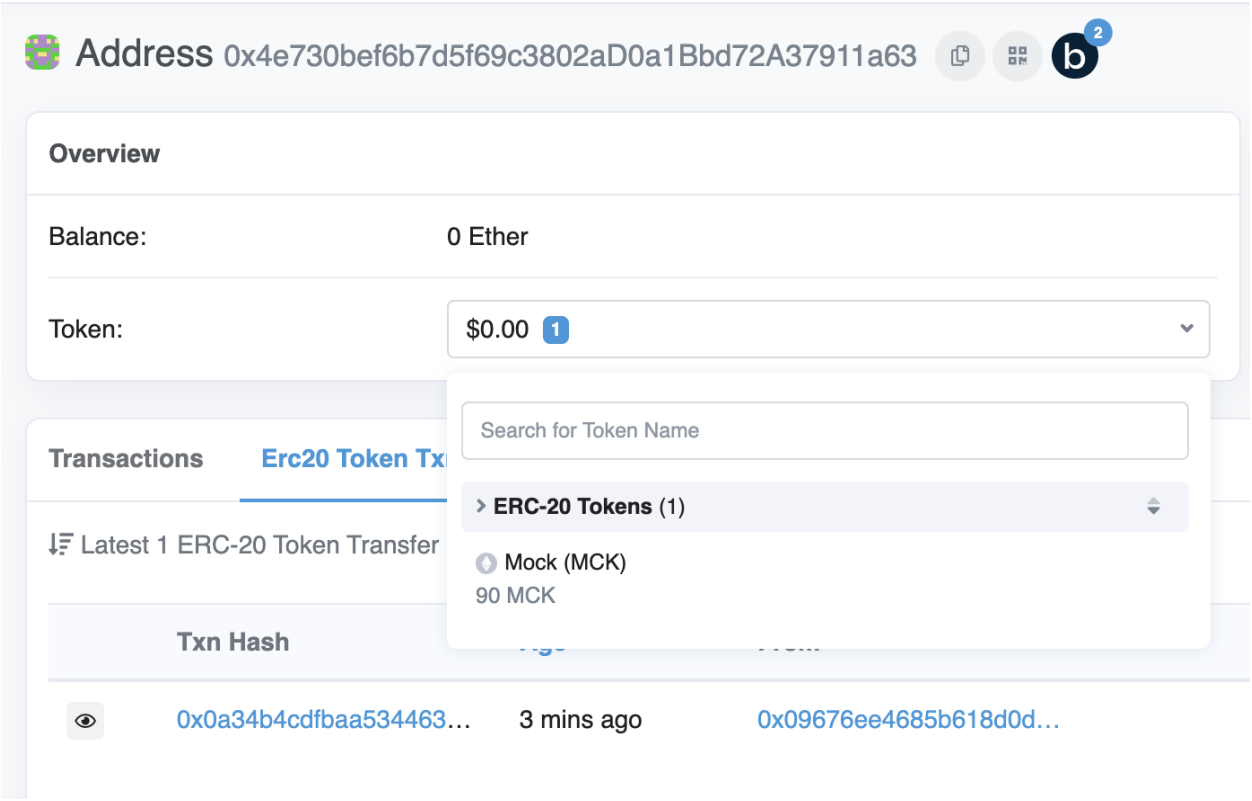

Рисунок 2.17 – Баланс токену в адреси

У адреси дійсно є 90 токенів, які ми передали, а значить система працює правильно.

Висновки по розділу II

У розділі розглянуто концепцію, проектування та реалізацію смарт-контрактів для отримання оплат.

Були використані основні стандарти для створення контрактів, таких як ERC20 та ERC721. Написані контракти під стандарти були створені за допомогою платформи OpenZeppelin.

Контракт NftBase був створений за допомогою стандарту ERC721 та додані функціонал, який буде використовуватись у платежах, такі як адреса, на яку будуть

приходити токени після оплати та неможливість передачі невзаємозамінюваного токена третій особі. Контракт Factory налічує у собі функцію, за допомогою якої можна створювати копії контракту NftBase. Після створення копії, контракт передає право власності до особи, яка викликала функцію створення. Контракт Payment створений для обробки платежів, перевірки у користувача наявності невзаємозамінюваного токена та передачі токенів до адреси, яка знаходиться на контракті NftBase.

Було розглянуто та протестовано контракти як за допомогою скриптів, так і розгорнуті на блокчейні, що дає змогу зрозуміти, що контракти готові до використання. Метод тестування полягав у покритті всього функціоналу контрактів тестами, а також використанні сценарії, які будуть відбуватися постійно, такі як створення копії контрактів, перевірка, чи дійшли токени до адреси та отримання невзаємозамінюваного токена на адресу користувача.

3. АНАЛІЗ СМАРТ-КОНТРАКТІВ НА ПРЕДМЕТ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ

3.1 Фреймворки для виявлення вразливостей у смарт-контрактах Ethereum

Основна проблема при створені смарт-контракту є безпека. Розробнику потрібно з нуля створювати структуру для того, щоб функціонал, який задумувався для створення, працював належним чином. Не повинно бути можливості у кожного користувача забрати токени з контракту або викликати функції, які впливають на роботу контракту.

Існує кілька потенційних уразливостей, які можуть існувати в смарт-контрактах на блокчейн-платформах [22]. Деякі з найпоширеніших включають:

- відсутність належного тестування та перевірки коду контракту, що може призвести до помилок і вразливостей, якими можуть скористатися зловмисники;
- погано розроблений або реалізований код смарт-контракту, що може призвести до вразливостей, які дозволяють зловмисникам викрасти кошти або іншим чином маніпулювати контрактом;
- використання незахищених або застарілих бібліотек або коду в контракті, що може створити вразливості, якими можуть скористатися зловмисники;
- використання наданого користувачем введення в коді контракту без належної перевірки чи валідації7, що може призвести до вразливостей, таких як атаки «повторного входу»;
- відсутність процесу аудиту або перевірки коду смарт-контракту, через що вразливості можуть залишатися непоміченими та невирішеними.

Усім, хто використовує або розробляє смарт-контракти на блокчейн-платформах, важливо знати про ці потенційні вразливості та вживати заходів для їх усунення. Це може включати такі речі, як належне тестування та перевірка коду контракту, уникнення використання незахищених бібліотек або коду та впровадження надійних заходів безпеки в контракті.

Кожен розробник має можливість перевірити, чи є у розробленому контракті вразливість, яка може вплинути на користувачів та їх токени. За допомогою інструментів, таких як фазінг тестування та додатків можна знайти основні проблеми, які частіше всього зустрічаються при розробці смарт-контрактів. Розглянемо деякі з них.

Slither — це структура статичного аналізу Solidity, написана на мові Python. Вона запускає набір детекторів уразливостей, друкує візуальну інформацію про деталі контракту та надає API для легкого написання власного аналізу. Slither дозволяє розробникам знаходити вразливості, покращувати розуміння коду та швидко створювати прототипи спеціального аналізу [23].

За допомогою Slither розробник може перевірити свої контракти на основні вразливості (рис 3.1-3.2). Під кожну вразливість є її опис та рекомендації по усуненню (рис 3.3). Розробник повинен розуміти, що такий аналізатор не завжди може видавати правильну інформацію або ж не бачити лімітацій, які виставленні у функції для запобігання того чи іншого сценарію.

```
Payment.payInNative(address,uint256,uint256) (contracts/Payment.sol#16-30) sends eth to arbitrary user
        Dangerous calls:
        - address(_treasury).transfer(_amount) (contracts/Payment.sol#23)
        - address(msg.sender).transfer(returnAmount) (contracts/Payment.sol#24)
        - address(_treasury).transfer(_amount) (contracts/Payment.sol#26)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
```

Рисунок 3.1 – Приклад вразливості високого рівня

Рисунок 3.2 – Результат аналізу за допомогою розширення у редакторі коду VSCode

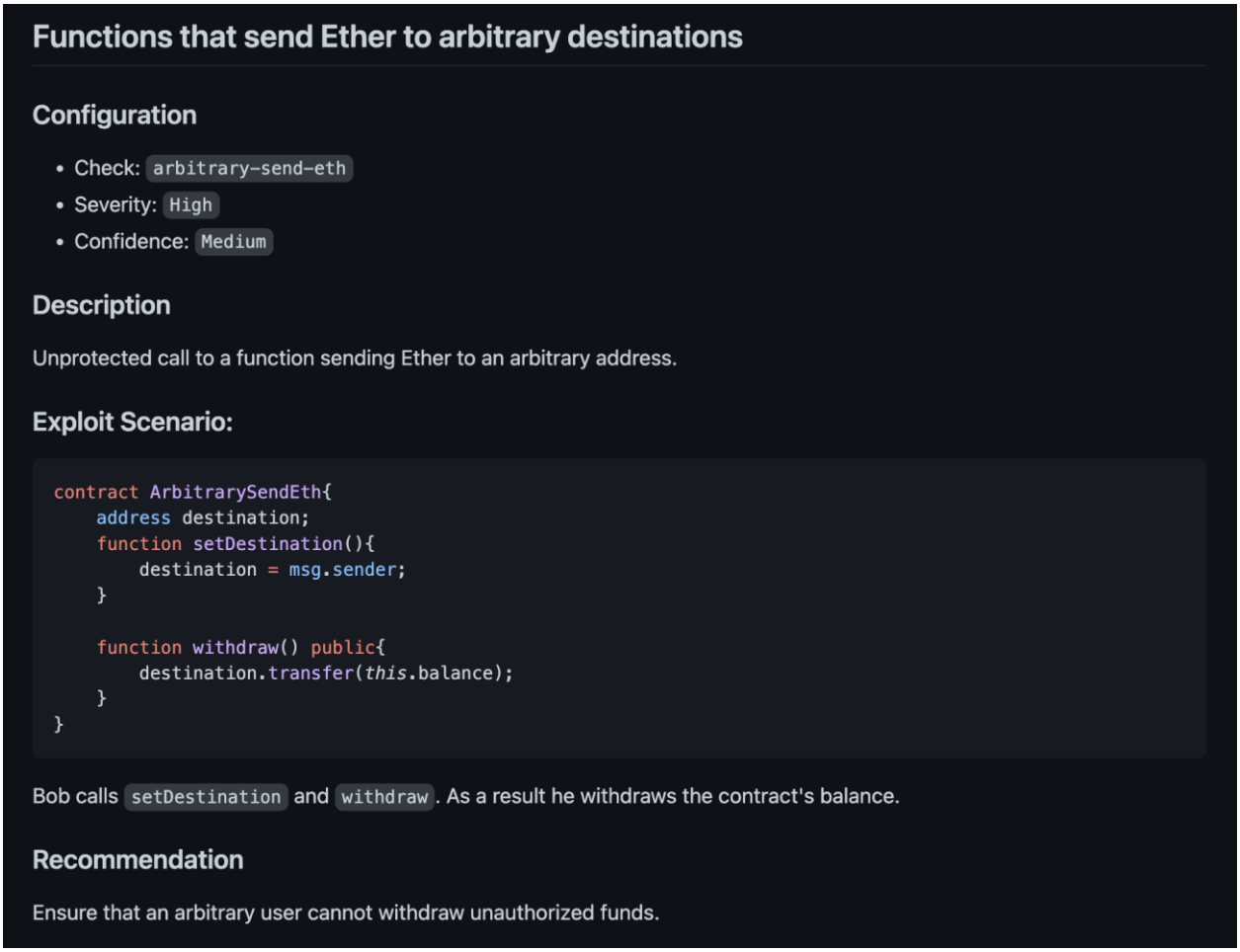

Рисунок 3.3 – Приклад опису вразливості та рекомендації щодо її усунення

Інший приклад є Echidna – програма на Haskell, розроблена для тестування смарт-контрактів Ethereum на основі фаззингу/власності [24]. Він використовує складні кампанії фаззингу на основі граматики на основі контрактного ABI для фальсифікації визначених користувачем предикатів або тверджень Solidity. Echidna розроблена з урахуванням модульності, тому її можна легко розширити, щоб включити нові мутації або перевірити певні контракти в конкретних випадках.

Фаззинг або фазз-тестування — це автоматизована техніка тестування програмного забезпечення, яка передбачає надання недійсних, неочікуваних або випадкових даних як вхідних даних для комп’ютерної програми [25]. Програма

відстежується на наявність виключень, таких як збої, помилки вбудованого коду або потенційні витоки пам’яті.

Для розуміння як працює Echidna, створимо контракт, який будемо перевіряти на вразливість. Контракт складається з конструктора, у який приймається пароль, який буде використовуватись, щоб розблокувати контракт та функція розблокування. Код контракту представлений на рисунку 3.4.

```
// SPDX-License-Identifier: MIT

// All data on-chain can be read! Don't store sensitive information
// Even if you make it private
// Level from Ethernaut: https://ethernaut.openzeppelin.com/level/0xf94b476063B6379A3c8b6C836efB8B3e10eDe188
// Slighly modified

pragma solidity ^0.8.0;

contract Vault {
    bool public s_locked;
    bytes32 private s_password;

    constructor(bytes32 password) {
        s_locked = true;
        s_password = password;
    }

    function unlock(bytes32 password) external {
        if (s_password == password) {
            s_locked = false;
        }
    }
}

```

Рисунок 3.4 – Код контракту для тестування

Тепер треба створити контракт, який Echidna буде використовувати для тестування. Для цього треба наслідувати контракт, який був створений раніше, та створити функцію, яка буде отримувати пароль та дивитись, чи збігається він з тим, який знаходиться у контракті (рис 3.5).

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.7;
3
4  import "../../Vault.sol";
5
6  contract VaultFuzzTest is Vault {
7      constructor() Vault("123asd123") {}
8
9      function echidna_test_find_password() public view returns (bool) {
10         return s_locked == true;
11     }
12 }
```

Рисунок 3.5 – Код контракту для тестування

Коли смарт-контракт для тестування готовий, треба запустити Echidna для тестування. Для цього треба мати образ Echidna на Docker та створити конфігураційний файл для запуску тесту. Кінцева команда для тестування буде: echidna-test “шлях до контракту” --contract VaultFuzzTest --config “шлях до конфігураційного файлу”. Після цієї команди отримуємо результат аналізу.

```
Echidna 1.7.2
Tests found: 1
Seed: 8904217114430560841
Unique instructions: 157
Unique codehashes: 1
Corpus size: 1
Tests
echidna_test_find_password: FAILED!

Call sequence:
1.unlock("123asd123\NUL\NUL\NUL\NUL\NUL\NUL\NUL\NUL\NUL\NUL\NUL\NUL\NUL\NUL\NUL\NUL\NUL\NUL\NUL\NUL\N
Campaign complete, C-c or esc to exit
```

Рисунок 3.6 – Результат тестування

Як видно з рисунка 3.6, Echidna знайшла результат, за допомогою якого можна розблокувати контракт, не знаючи пароль. Основна проблема представленого

контракту полягає у тому, що змінна паролю хоч і приватна, але на блокчейні вона зберігається незашифрованою у комірці пам’яті, до якої можна отримати доступ, відправивши запит на одну з нод. Одна з основних практик створення смарт-контрактів є те, що ніколи не треба створювати приватні змінні, у яких буде зберігатися чутлива інформація, наприклад, пароль.

3.2 Розробка рекомендацій щодо усунення вразливостей смарт-контрактів

Майже кожного місяця у сфері блокчейн технології зустрічаються історії про те, що якийсь протокол зламали на мільйони доларів. Все через те, що розробники припустили помилки при розробці, що призвело до виведення грошей у користувачів. Статистика за третій квартал 2022 року представлений на рисунку 3.7.

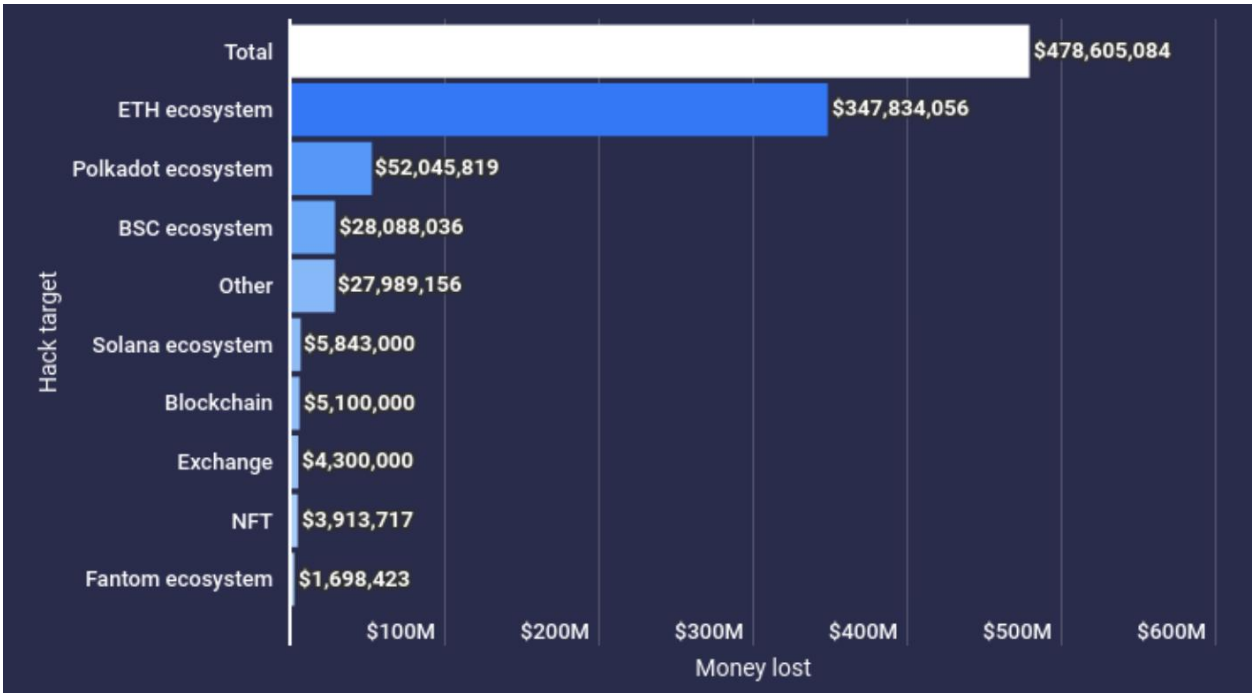

Рисунок 3.7 – Кількість вкрадених грошей на різних блокчейнах за третій квартал 2022 року [26]

Є кілька ключових речей, які можна зробити, щоб допомогти усунути вразливості в смарт-контрактах на блокчейні [27]. Основні рекомендації:

- слід використовувати безпечну мову програмування: вибір мови програмування, яка відома своїми функціями безпеки, може допомогти зменшити ймовірність уразливостей у смарт-контрактах. Деякі приклади безпечних мов програмування для смарт-контрактів включають Solidity і Vyper;
- завжди треба проводити ретельне тестування: важливо ретельно протестувати свої смарт-контракти перед розгортанням їх у блокчейні. Це може допомогти виявити та виправити будь-які потенційні вразливості до того, як їх використають;
- важливо використовувати сторонні послуги аудиту: є компанії, які спеціалізуються на перевірці смарт-контрактів на наявність уразливостей. Ці служби можуть забезпечити додатковий рівень безпеки для смарт-контрактів, визначаючи потенційні вразливості, які могли бути допущені при розробці контрактів та пропущені з якихось причин на етапі тестування;
- необхідно дотримуватися найкращих практик. Існують найкращі практики написання безпечних смарт-контрактів, яких слід дотримуватися, щоб зменшити ймовірність уразливості. Деякі з цих передових практик включають використання безпечних бібліотек і уникнення використання складного коду, коли це можливо.

В Інтернеті є багато прикладів вразливостей та їх усунення. Топові компанії, які надають послуги у сфері безпеки, розробили курси та фреймворки, які можна використовувати, щоб здобувати нові знання та покращувати свої навички як спеціаліста.

Аудити, які проводять компанії частіше всього є у відкритому доступі, що дає можливість дивитися, як не треба писати контракти та основні проблеми, з якими

могли зіштовхнутися розробники. Чим більше дивитися різних аудитів та пояснень до вразливостей, тим краще та кваліфікованіше стає розробник

Висновки по розділу III

Вразливості на смарт-контрактах можуть призвести до втрат коштів користувачів, які користувалися протоколом. Наразі розробники повинні бути висококваліфікованими, щоб робити правильні рішення для створення безпечних контрактів. У цьому можуть допомагати фреймворки для аналізу контрактів на вразливості. Одними з популярніших є Slither та Echidna.

Slither — це інструмент із відкритим кодом для аналізу контрактів Solidity. Він розроблений, щоб допомогти визначити потенційні вразливості в смарт-контрактах і надати рекомендації щодо їх усунення. Slither працює шляхом статичного аналізу коду смарт-контракту та порівняння його з набором відомих шаблонів безпеки. Це дозволяє виявити потенційні вразливості та надати пропозиції щодо їх усунення. Розробники можуть використовувати Slither для покращення безпеки своїх смарт-контрактів і зниження ймовірності вразливості.

Echidna — це інструмент з відкритим кодом для тестування смарт-контрактів. Він розроблений, щоб допомогти визначити потенційні вразливості в смарт-контрактах шляхом автоматичної генерації тестових випадків для певного контракту. Echidna працює за допомогою техніки під назвою фазз-тестування, яка передбачає надання великої кількості випадкових вхідних даних у контракт і спостереження за його поведінкою. Це може допомогти виявити потенційні вразливості, які можуть бути неочевидними під час ручної перевірки коду контракту. Echidna особливо корисний для виявлення вразливостей у контрактах, які мають складну або дуже взаємопов’язану логіку.

Ключем до усунення вразливостей у смарт-контрактах на блокчейні є ретельний і старанний підхід. Приділивши час ретельному написанню, тестуванню та перевірці розроблених смарт-контрактів, можна переконатися, що вони безпечні та вільні від уразливостей.

ВИСНОВКИ

Технологія блокчейн може революціонізувати спосіб здійснення та обробки платежів. Безпека те ефективність є одним з основних переваг використання блокчейну. Система прозорості дозволяє легко отримати інформацію про той чи інший актив, який має користувач, або інформацію щодо виконаних транзакцій.

Є багато прикладів використання блокчейну як способу оплати, наприклад, Біткоїн. Криптобіржі пропонують оформлювати картки, за допомогою яких можна оплачувати покупки так само як і з банківською карткою. Також є можливість використовувати свій смартфон, на якому завантажений гаманець, та, сканувавши QR-код та підтвердивши транзакцію, можна робити оплати товарів напряму у блокчейні.

Мета розробки системи оплати із використанням невзаємозамінюваного токена полягає у тому, щоб реалізувати додаткові можливості для бізнесу у заохочуванні користувачів, а для компаній забезпечити менші витрати на створення транзакцій.

У роботі представлено три розроблені контракти. Перший — це контракт на основі стандарту ERC721, який і є по суті реалізацією невзаємозамінюваного токену, який буде використовуватись при оплаті. Основний функціонал полягає у тому, що тільки один користувач може мати один токен, передача токену неможлива, а також забезпечується зберігання адреси, на яку буде приходити криптовалюта після виконання транзакції. Другий контракт є фабрикою, за допомогою якого створюються копії першого контракту. Третій контракт реалізує механізм, за допомогою якого можна створювати транзакції на основі традиційної системи оплати. У користувача є можливість виконувати транзакції на основі стандарту ERC20 або нативного токену блокчейну.

Також у роботі проаналізовано проблеми розробки смарт-контрактів, оскільки це порівняно нова технологія і на даний момент часу стрімко розвивається. Основна проблема розробки смарт-контрактів є безпека. Тому розглянуто декілька можливостей для перевірки контрактів на вразливості. Використання аналізатора допомагає виявити основні проблеми та вразливості, які представлені у контракті. Прикладом аналізатора є Slither. Фазінг є одним з методів для більш глибокої перевірки контрактів на безпеку. Прикладом фазінг-інструменту є Echidna. За допомогою цього інструмента є можливість перевірити функції контракту з використанням різних технік виклику та наборів аргументів, які можуть “зламати” логіку самого контракту.

Для створення більш безпечних контрактів розробники повинні бути обізнані у основах блокчейну, шифруванні та особливостях мови, на якій створюються контракти.

ПЕРЕЛІК ПОСИЛАНЬ

1. Binance Crypto Card [Електронний ресурс]. – Режим доступу: https://www.binance.com/en/cards
2. Openzeppelin contracts [Електронний ресурс]. – Режим доступу: https://wizard.openzeppelin.com/
3. Openzeppelin documentation [Електронний ресурс]. – Режим доступу: https://docs.openzeppelin.com/
4. Cold Storage: What It Is, How It Works, Theft Protection [Електронний ресурс]. – Режим доступу: https://www.investopedia.com/terms/c/cold-storage.asp
5. What is a Centralized Crypto Exchange (CEX)? [Електронний ресурс]. – Режим доступу: https://www.babypips.com/crypto/learn/what-is-a-centralized-exchange-cex
6. Benefits of blockchain [Електронний ресурс]. – Режим доступу: https://www.ibm.com/topics/benefits-of-blockchain
7. Blockchain In Payment: Accelerating Payment Services [Електронний ресурс]. – Режим доступу: https://101blockchains.com/blockchain-in-payment/
8. How do Bitcoin transactions work? [Електронний ресурс]. – Режим доступу: https://www.bitcoin.com/get-started/how-bitcoin-transactions-work/
9. What Is a DEX (Decentralized Exchange)? [Електронний ресурс]. – Режим доступу: https://blog.chain.link/dex-decentralized-exchange/
10. 1inch - DeFi / DEX aggregator on Ethereum [Електронний ресурс]. – Режим доступу: https://app.1inch.io/#/1/unified/swap/ETH/DAI
11. What is a crypto wallet? [Електронний ресурс]. – Режим доступу: https://n26.com/en-eu/blog/what-is-a-crypto-wallet
12. Taking Payments IRL with Solana Pay [Електронний ресурс]. – Режим доступу: https://www.pointer.gg/tutorials/solana-pay-irl-payments/944eba7e-82c6-4527-b55c-5411cdf63b23

13. Smart Contract Challenges [Електронний ресурс]. – Режим доступу: https://hedera.com/learning/smart-contracts/smart-contract-challenges
14. Solidity documentation [Електронний ресурс]. – Режим доступу: https://docs.soliditylang.org/en/v0.8.17/
15. Prerequisites to Learn Blockchain Technology: It's Not What You Think It Is [Електронний ресурс]. – Режим доступу: https://www.upgrad.com/blog/prerequisites-to-learn-blockchain/
16. NFTs, explained [Електронний ресурс]. – Режим доступу: https://www.theverge.com/22310188/nft-explainer-what-is-blockchain-crypto-art-faq
17. EIP-721: Non-Fungible Token Standard [Електронний ресурс]. – Режим доступу: https://eips.ethereum.org/EIPS/eip-721
18. Getting the most out of CREATE2 [Електронний ресурс]. – Режим доступу: https://blog.openzeppelin.com/getting-the-most-out-of-create2/
19. ERC-20 TOKEN STANDARD [Електронний ресурс]. – Режим доступу: https://ethereum.org/en/developers/docs/standards/tokens/erc-20/
20. What is Etherscan and how do you use it? [Електронний ресурс]. – Режим доступу: https://www.moonpay.com/blog/what-is-etherscan
21. What Is Opensea and How Does It Work? [Електронний ресурс]. – Режим доступу: https://www.makeuseof.com/what-is-opensea-and-how-does-it-work/
22. Smart Contract Vulnerabilities [Електронний ресурс]. – Режим доступу: https://hacken.io/discover/smart-contract-vulnerabilities/
23. Slither: The Leading Static Analyzer for Smart Contracts [Електронний ресурс]. – Режим доступу: https://blog.trailofbits.com/2019/05/27/slither-the-leading-static-analyzer-for-smart-contracts/
24. Using Echidna to test a smart contract library [Електронний ресурс]. – Режим доступу: https://blog.trailofbits.com/2020/08/17/using-echidna-to-test-a-smart-contract-library/

25. What Is Fuzz Testing and How Does It Work? [Електронний ресурс]. – Режим доступу: https://www.synopsys.com/glossary/what-is-fuzz-testing.html
26. Blockchain Hackers Stole Nearly Half a Billion in Q3 2022 [Електронний ресурс]. – Режим доступу: https://beincrypto.com/blockchain-hackers-stole-nearly-half-a-billion-in-q3-2022/
27. Top 7 Strategies for Finding Smart Contract Vulnerabilities [Електронний ресурс]. – Режим доступу: https://blog.chain.link/smart-contract-bug-hunting/

ДОДАТОК А

ВМІСТ ФАЙЛУ NFTBASE.SOL

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.17;
```

```
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/Counters.sol";

error NotTransferable();
error OnlyOneForUser();
error ZeroAddress();

contract NftBase is ERC721, Ownable {
    using Counters for Counters.Counter;
    address public collectionAddr;
    address payable public treasury;

    Counters.Counter private _tokenIdCounter;

    event NewTreasury(address payable _treasury);

    constructor(string memory _name, string memory _symbol, address
_owner, address payable _treasury) ERC721(_name, _symbol) {
        _transferOwnership(_owner);
        collectionAddr = address(this);
        treasury = _treasury;
    }

    function safeMint() public {
        if (balanceOf(msg.sender) == 1) revert OnlyOneForUser();
        uint256 tokenId = _tokenIdCounter.current();
```

```
        _tokenIdCounter.increment();
        _safeMint(msg.sender, tokenId);
    }

    function setTreasury(address payable _treasury) external onlyOwner
{
        if (_treasury == address(0)) revert ZeroAddress();
        treasury = _treasury;

        emit NewTreasury(_treasury);
    }

    function getTreasury() public view returns(address) {
        return treasury;
    }

    function _transfer(
        address from,
        address to,
        uint256 tokenId
    ) internal override {
        revert NotTransferable();
    }
}
```

ДОДАТОК Б

ВМІСТ ФАЙЛУ FACTORY.SOL

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.17;
```

```
import "./NftBase.sol";

contract Factory {
    NftBase[] public collections;

    function create2(
        string memory _name,
        string memory _symbol,
        address _owner,
        address payable _treasury,
        bytes32 _salt
    ) public {
        NftBase collection = (new NftBase){salt: _salt}(_name,
_symbol, _owner, _treasury);
        collections.push(collection);
    }

    function getCollection(uint _index)
        public
        view
        returns (
            address owner,
            address collectionAddr
        )
    {
```

```
        NftBase collection = collections[_index];


        return (collection.owner(), address(collection));
    }
}
```

ДОДАТОК С

ВМІСТ ФАЙЛУ PAYMENT.SOL

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.17;
```

```
import "./NftBase.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";

error LowAllowance();
error WrongAmount();

contract Payment {
    using SafeERC20 for IERC20;

    event Payed(address _nft, address _token, uint256 _amount, uint256
indexed receitId, uint256 timestamp);

    function payInNative(address _nft, uint256 _amount, uint256
receitId) public payable {
        if (msg.value != _amount) revert WrongAmount();
        address _treasury = NftBase(_nft).getTreasury();

        if (NftBase(_nft).balanceOf(msg.sender) == 1) {
            _amount = _amount * 90 / 100;
            uint256 returnAmount = _amount * 10 / 100;
            payable(_treasury).transfer(_amount);
            payable(msg.sender).transfer(returnAmount);
        } else {
            payable(_treasury).transfer(_amount);
        }
```

```
        emit Payed(_nft, address(0), _amount, receitId,
block.timestamp);

    }

    function payInTokens(address _nft, address _token, uint256
_amount, uint256 receitId) public {

        if (IERC20(_token).allowance(msg.sender, address(this)) <
_amount) revert LowAllowance();



        if (NftBase(_nft).balanceOf(msg.sender) == 1) {

            _amount = _amount * 90 / 100;

            IERC20(_token).safeTransferFrom(msg.sender,
NftBase(_nft).getTreasury(), _amount);

        } else {

            IERC20(_token).safeTransferFrom(msg.sender,
NftBase(_nft).getTreasury(), _amount);

        }



        emit Payed(_nft, _token, _amount, receitId, block.timestamp);

    }

}
```