МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Львівський національний університет імені Івана Франка Факультет електроніки та комп'ютерних технологій Кафедра системного проектування

Допустити до захисту
Завідувач кафедри
_____проф. Шувар Роман Ярославович
«___»_____2023 р.

Кваліфікаційна робота Бакалавр
(освітній ступінь)

ВИКОРИСТАННЯ BLOCKCHAIN ТЕХНОЛОГІЇ ДЛЯ ГЕНЕРУВАННЯ СМАРТ-КОНТРАКТУ

Виконав:
студент IV курсу групи ФЕП-41
спеціальності:
121 Інженерія програмного забезпечення
_____Коберський Данило Миколайович

Науковий керівник:
_____асист. Каськун Олег Данилович
«___»_____2023 р.

Рецензент:

_____
(підпис) (ПІБ)

Львів 2023

АНОТАЦІЯ

Ця дипломна робота присвячена дослідженню використання блокчейн технології для створення смарт-контракту. В останні роки блокчейн технологія здобула значну популярність завдяки своїм безпечним і децентралізованим характеристикам. Смарт-контракти, з іншого боку, є автоматизованими програмами, які виконують умови, записані в їх коді, без потреби довіряти посереднику.

Метою цієї дипломної роботи є дослідження можливостей і переваг використання блокчейн технології для створення смарт-контракту. Робота включає аналіз основних понять та принципів блокчейн технології та смарт-контрактів, огляд сучасних платформ блокчейн, що підтримують смарт-контракти, та дослідження їхніх особливостей.

У рамках дослідження будуть розглянуті потенційні переваги використання блокчейн технології для створення смарт-контракту, такі як безпека, надійність, швидкість та недоступність для зловмисників. Також будуть розглянуті можливі виклики і проблеми, пов'язані з цією технологією, такі як масштабованість і приватність.

На основі проведеного дослідження буде розроблений прототип системи створення смарт-контракту з використанням блокчейн технології. Результати дослідження та розроблені рекомендації можуть бути корисні для підприємств, які планують впровадження смарт-контрактів на базі блокчейн технології.

Ця дипломна робота допоможе глибше розуміти переваги і обмеження використання блокчейн технології для створення смарт-контракту та внести

свій внесок у розвиток цього перспективного напрямку в сучасному інформаційному суспільстві.

ABSTRACT

This thesis explores the use of blockchain technology for smart contract creation. In recent years, blockchain technology has gained significant popularity due to its secure and decentralized nature. Smart contracts, on the other hand, are automated programs that execute conditions written in their code without the need for intermediaries.

The objective of this thesis is to investigate the possibilities and advantages of utilizing blockchain technology for smart contract creation. The work includes an analysis of the key concepts and principles of blockchain technology and smart contracts, an overview of contemporary blockchain platforms supporting smart contracts, and an examination of their features.

The research will address the potential benefits of using blockchain technology for smart contract creation, such as security, reliability, speed, and resistance to malicious actors. It will also discuss potential challenges and issues associated with this technology, such as scalability and privacy.

Based on the conducted research, a prototype system for smart contract creation using blockchain technology will be developed. The research findings and recommended approaches can be valuable for enterprises planning to implement blockchain-based smart contracts.

This thesis aims to deepen the understanding of the advantages and limitations of utilizing blockchain technology for smart contract creation and contribute to the development of this promising field in the modern information society.

ЗМІСТ

ВСТУП

У сучасному цифровому світі, де швидкість і безпека є ключовими пріоритетами, блокчейн технологія вирізняється як інноваційний та перспективний інструмент. Вона надає можливість створювати децентралізовані, безпечні та автоматизовані системи, забезпечуючи виконання угод між сторонами без посередництва. Однією з найважливіших інтерпретацій цієї технології є використання блокчейну для створення смарт-контрактів.

Смарт-контракти, що базуються на блокчейн технології, втілюють ідею автоматизації та безпосередньої виконавчої сили угод, які базуються на умовах, визначених у їх коді. Вони дозволяють сторонам забезпечувати безпеку, довіру та виконання обіцянок без необхідності залучення традиційних посередників. Такі смарт-контракти виявляються особливо корисними в областях, де точність, незалежність та швидкість виконання є критичними факторами.

Ця дипломна робота присвячена вивченню можливостей використання блокчейн технології для створення смарт-контракту. Метою є розкриття переваг цього підходу, аналіз основних концепцій та принципів блокчейну та смарт-контрактів, а також розробка прототипу системи створення смарт-контракту на базі блокчейн технології. Результати дослідження можуть стати цінним допоміжним матеріалом для підприємств та організацій, які розглядають використання блокчейну для автоматизації угод та оптимізації бізнес-процесів.

У подальшому розділі цієї роботи будуть розглянуті основні концепції блокчейн технології, включаючи розподілену структуру, хеш-функції, консенсусні алгоритми та безпеку мережі. Також буде досліджено концепцію смарт-контрактів, їх властивості, переваги та потенційні ризики. Після цього

буде проведений аналіз різних платформ блокчейн технології, що підтримують смарт-контракти, з урахуванням їх особливостей та можливостей. На підставі здобутих знань буде розроблений прототип системи створення смарт-контракту з використанням блокчейн технології.

Ця робота відіграє важливу роль у розумінні потенціалу блокчейн технології та смарт-контрактів, а також сприяє розвитку інноваційних рішень у сфері цифрових угод та автоматизації бізнес-процесів.

Актуальність цієї роботи:

Сучасний світ швидко рухається до цифрової економіки, де автоматизація та безпека інформаційних процесів стають все важливішими. У такому контексті блокчейн технологія та смарт-контракти відіграють ключову роль, надаючи можливість автоматизованого виконання угод без необхідності довіряти посередникам.

Застосування блокчейн технології для створення смарт-контрактів має ряд переваг. Воно забезпечує безпеку, аутентифікацію та недоступність для зловмисників, завдяки децентралізованій природі блокчейну. Крім того, він пропонує надійність та непідробнуваність угод, завдяки використанню криптографічних алгоритмів. Такі переваги особливо актуальні в областях, де потрібно гарантувати точність та виконання угод безпосередньо між сторонами, наприклад, у фінансовому секторі, логістичних компаніях, ланцюжку постачання та багатьох інших сферах.

Однак, впровадження блокчейн технології та смарт-контрактів не є безвідсотковим. Є виклики, пов'язані з масштабованістю, приватністю та енергоефективністю цієї технології. До того ж, розуміння потенціалу та

обмежень блокчейн технології для створення смарт-контрактів є обмеженим, і вимагає більш детального дослідження.

У цьому контексті дана дипломна робота є актуальною. Вона спрямована на вивчення можливостей використання блокчейн технології для створення смарт-контрактів, а також на розробку прототипу системи, яка демонструє практичність та ефективність цього підходу. Результати цього дослідження можуть бути використані підприємствами та організаціями, які розглядають впровадження блокчейн технології для автоматизації своїх бізнес-процесів, а також сприятимуть подальшому розвитку цієї області.

Об'єкт дослідження цієї роботи:

Об'єктом дослідження даної роботи є використання блокчейн технології для створення смарт-контрактів. Робота фокусується на аналізі та дослідженні можливостей, переваг і обмежень використання блокчейн технології для створення автоматизованих та безпосередніх угод між сторонами без посередництва.

Об'єкт дослідження включає різні аспекти блокчейн технології, такі як децентралізована структура, хеш-функції, консенсусні алгоритми та безпека мережі. Також вивчаються особливості смарт-контрактів, їх властивості, переваги та потенційні ризики. Об'єктом дослідження є також розгляд різних платформ блокчейн технології, що підтримують смарт-контракти, з метою визначення їх можливостей та пристосування до потреб бізнесу.

Результати дослідження і розробка прототипу системи створення смарт-контракту на базі блокчейн технології також включаються в об'єкт дослідження. Це дозволяє вивчити практичну реалізацію використання

блокчейн технології для створення смарт-контрактів і оцінити його ефективність та придатність для використання в різних сферах бізнесу.

Отже, об'єктом дослідження є використання блокчейн технології для створення смарт-контрактів, що включає аналіз, вивчення та практичну реалізацію цього підходу.

Предмет дослідження цієї роботи:
Предметом дослідження цієї роботи є використання блокчейн технології для створення смарт-контрактів. Робота спрямована на розуміння і дослідження можливостей, переваг і обмежень цього підходу, а також на розробку прототипу системи створення смарт-контракту на базі блокчейн технології.

Предмет дослідження включає аналіз концепцій блокчейн технології, таких як розподілена структура, хеш-функції, консенсусні алгоритми та безпека мережі. Також досліджуються концепції смарт-контрактів, їх властивості, переваги та потенційні ризики. Предметом дослідження є також розгляд різних платформ блокчейн технології, що підтримують смарт-контракти, з метою визначення їх можливостей та вибору оптимального варіанта для реалізації прототипу.

Розробка прототипу системи створення смарт-контракту на базі блокчейн технології також включається до предмету дослідження. Це дозволяє перевірити практичну придатність і ефективність використання блокчейн технології для створення смарт-контрактів і оцінити його потенційну працездатність в різних сферах бізнесу.

Отже, предметом дослідження є використання блокчейн технології для створення смарт-контрактів, включаючи аналіз, дослідження та розробку прототипу системи, що демонструє практичну реалізацію цього підходу.

Методи дослідження, використані в даній роботі, включають:

1. Літературний огляд: Проведення систематичного аналізу академічних джерел, наукових статей, книг і онлайн-документів, що стосуються блокчейн технології та смарт-контрактів. Цей метод дозволяє отримати теоретичну базу, ознайомитися зі сучасними дослідженнями та визначити ключові концепції, проблеми та напрямки розвитку області.

2. Аналіз використаних платформ: Проведення детального аналізу різних платформ блокчейн технології, що підтримують смарт-контракти. Це включає дослідження їхньої архітектури, особливостей, масштабованості, безпеки, протоколів консенсусу та інших функціональних аспектів. Такий аналіз допомагає вибрати оптимальну платформу для реалізації прототипу системи створення смарт-контракту.

3. Розробка прототипу: Розробка програмного прототипу системи створення смарт-контракту з використанням обраної блокчейн платформи. Цей метод включає проектування архітектури системи, розробку смарт-контракту, інтеграцію з блокчейн мережею та тестування прототипу. Результатом цього методу є практична реалізація системи, яка дозволяє оцінити функціональність та ефективність використання блокчейн технології для створення смарт-контрактів.

4. Аналіз результатів: Проведення аналізу результатів використання блокчейн технології для створення смарт-контракту. Це включає оцінку ефективності, безпеки, швидкодії та масштабованості системи, порівняння результатів з існуючими рішеннями та визначення потенційних переваг і обмежень використання даного підходу. Аналіз результатів дозволяє зробити висновки

щодо придатності блокчейн технології для створення смарт-контрактів та вказати можливі напрямки подальшого дослідження.

Отже, в даній роботі використовуються методи літературного огляду, аналізу використаних платформ, розробки прототипу та аналізу результатів з метою досягнення поставлених цілей дослідження.
Сфера застосування даної роботи включає широкий спектр галузей і секторів, де можливе використання смарт-контрактів на базі блокчейн технології.
Деякі з основних сфер застосування включають:

1. Фінанси та банківська справа: Використання смарт-контрактів у фінансових операціях, таких як автоматизовані платежі, мікрофінансування, емісія цифрових активів та облік фінансових договорів. Блокчейн технологія забезпечує безпеку, прозорість та швидкість проведення фінансових транзакцій.

2. Логістика та постачання: Використання смарт-контрактів для автоматизації ланцюжка постачання, відстеження товарів та забезпечення дотримання угод між різними сторонами. Це дозволяє ефективно керувати запасами, знижувати витрати та покращувати прозорість угод.

3. Нерухомість: Використання смарт-контрактів для автоматизації процесів купівлі-продажу нерухомості, реєстрації власності та управління орендою. Це може забезпечити безпеку та довіру між учасниками угоди, а також спростити та прискорити процеси нерухомісного бізнесу.

4. Управління ланцюжком постачання: Використання смарт-контрактів для відстеження та підтвердження доставок, контролю якості та розрахунків між різними етапами ланцюжка постачання. Це може покращити ефективність, прозорість та безпеку управління ланцюжком постачання.

5. Інтелектуальна власність та авторські права: Використання смарт-контрактів для автоматизованої реєстрації та управління авторськими правами, ліцензуванням та розподілом винагороди. Це дозволяє створити довіру та прозорість у відносинах між авторами, виконавцями та правовласниками.

Це лише кілька прикладів сфер застосування блокчейн технології та смарт-контрактів. Ці технології мають потенціал революціонізувати багато галузей бізнесу та суспільства, забезпечуючи безпеку, ефективність та автоматизацію угод та процесів.

РОЗДІЛ 1 АНАЛІЗ СТАНУ ПРОБЛЕМНОЇ ОБЛАСТІ

Аналіз стану проблемної області використання блокчейн технології для створення смарт-контрактів розкриває наступні важливі аспекти:

1. *Обмеженість технологічної зрілості:* Блокчейн технологія, хоч і вже здобула значну популярність та впроваджується в різних галузях, все ще перебуває на стадії розвитку. Існує необхідність вдосконалення шкали, швидкості та безпеки блокчейн мережі для більш широкого застосування смарт-контрактів.

2. *Безпека та конфіденційність:* Однією з основних проблем є забезпечення безпеки та конфіденційності смарт-контрактів на блокчейні. Вразливості у розробці контрактів та можливість зловживання через кодові дефекти можуть призвести до фінансових втрат та порушень домовленостей.

3. *Скалабельність:* Розширення масштабів блокчейн мережі є викликом, оскільки швидкість обробки транзакцій та розмір блоків можуть обмежувати кількість смарт-контрактів, які можуть бути виконані одночасно. Це особливо актуально в сферах з великим обсягом транзакцій, наприклад, фінансовій сфері.

4. *Легалітет та регулювання:* Правовий статус смарт-контрактів є ще не повністю визначеним. В багатьох юрисдикціях відсутні конкретні законодавчі рамки, що регулюють смарт-контракти, що може створювати невизначеність та правові ризики.

5. *Потреба відкритих стандартів:* Відсутність загальноприйнятих відкритих стандартів у сфері смарт-контрактів унеможливлює взаємодію різних блокчейн платформ та обмежує переносимість контрактів між різними системами.

Незважаючи на ці виклики, потенціал блокчейн технології та смарт-контрактів є значним. Поступове розвиток технологій, збільшення зрілості блокчейн мереж та встановлення відповідних правових рамок можуть сприяти широкому впровадженню та використанню смарт-контрактів у багатьох сферах діяльності.

1.1 Користь даної роботи

1. Впровадження новаторських технологій: Дослідження використання блокчейн технології для створення смарт-контрактів дозволяє впровадити інноваційні рішення в різних галузях, що може покращити ефективність та надійність угод та процесів.

2. Автоматизація та ефективність: Використання смарт-контрактів дозволяє автоматизувати виконання угод та процесів, що сприяє зниженню витрат часу та ресурсів. Це особливо актуально в сферах, де велика кількість транзакцій та угод потребує швидкого та точного виконання.

3. Підвищення довіри та безпеки: Блокчейн технологія надає високий рівень безпеки та довіри, оскільки угоди, здійснені за допомогою смарт-контрактів, є недійсними без погодження всіх сторін та їхньої підтримки в блокчейн мережі. Це дозволяє запобігати шахрайству та втратам, а також підвищує довіру між учасниками угод.

4. Прозорість та аудит: Блокчейн технологія забезпечує прозорість угод та транзакцій, оскільки записи зберігаються у розподіленій базі даних і доступні

для перегляду всім учасникам мережі. Це сприяє проведенню аудиту та перевірці дотримання умов угод.

5. Потенційні економічні переваги: Впровадження смарт-контрактів на базі блокчейн технології може привести до зменшення витрат на посередників, виключення необхідності використання традиційних інструментів, а також до прискорення процесів виконання угод, що може сприяти зростанню продуктивності та зниженню витрат.

Отже, дана робота має потенціал покращити ефективність, безпеку та довіру угод та процесів за допомогою використання блокчейн технології та смарт-контрактів, що може призвести до значних економічних та інноваційних переваг у різних сферах діяльності.

1.2 Методи застосування:

Використання блокчейн технології для створення смарт-контрактів має різноманітні методи застосування. Ось кілька прикладів:

1. Фінансовий сектор: Блокчейн технологія може використовуватись для створення смарт-контрактів у фінансових угодах, таких як перекази грошей, позики, страхування та торгівля цінними паперами. Це дозволяє автоматизувати процеси, знизити витрати та забезпечити швидше та безпечне виконання угод.

2. Логістика та постачання: Смарт-контракти можуть бути використані для автоматизації та відстеження логістичних процесів, включаючи доставку товарів, митний контроль та платежі. Це спрощує взаємодію між різними сторонами та покращує прозорість та ефективність логістичних ланцюжків.

3. Інтелектуальна власність: Блокчейн технологія може бути використана для реєстрації та управління правами на інтелектуальну власність, таку як авторські права, патенти та товарні знаки. Смарт-контракти дозволяють автоматизувати процеси ліцензування та розподілу винагороди між правовласниками.

4. Управління ланцюгами постачання: Використання смарт-контрактів у ланцюгах постачання може покращити відстеження та автоматизацію процесів, включаючи контроль якості, підписання угод, оплату та доставку товарів. Це забезпечує більшу прозорість, ефективність та безпеку у ланцюгах постачання.

5. Галузь нерухомості: Смарт-контракти можуть використовуватись у сфері нерухомості для автоматизації угод, здійснення переказу власності та управління правами на нерухомість. Це спрощує процеси купівлі-продажу та забезпечує безпеку та прозорість угод.

Це лише кілька прикладів методів застосування блокчейн технології та смарт-контрактів. Завдяки їхній гнучкості та автоматизації, вони можуть бути використані в різних галузях, де потрібні надійні та ефективні механізми угод та контрактів.

1.3 Веб-застосунки які можуть використовувати дану роботу:

Дана робота може бути застосована для розробки різноманітних веб-застосунків, які використовують блокчейн технологію та смарт-контракти. Ось кілька прикладів таких веб-застосунків:

1. Фінансові платформи: Веб-застосунки можуть бути розроблені для створення фінансових платформ, що дозволяють користувачам здійснювати перекази грошей, позики та інвестування за допомогою смарт-контрактів. Такі платформи можуть забезпечувати швидкі та безпечні фінансові операції, уникати посередників та зменшувати витрати.

2. Електронна комерція: Веб-застосунки для електронної комерції можуть використовувати смарт-контракти для автоматизації процесів замовлення, оплати, доставки та повернення товарів. Це дозволяє забезпечити безпеку та прозорість угод між продавцем та покупцем.

3. Управління ланцюгом постачання: Веб-застосунки можуть бути розроблені для відстеження та керування ланцюгами постачання за допомогою блокчейн технології та смарт-контрактів. Це дозволяє покращити прозорість, швидкість та надійність процесів постачання, від контролю за рівнем запасів до взаєморозрахунків між постачальниками та покупцями.

4. Інтелектуальна власність та авторські права: Веб-застосунки можуть бути створені для реєстрації та управління правами на інтелектуальну власність, використовуючи блокчейн технологію. Це забезпечує автоматизований процес реєстрації, захисту та ліцензування авторських прав, патентів та товарних знаків.

5. Угоди та контракти: Веб-застосунки можуть надавати можливість створювати та виконувати різні види угод та контрактів за допомогою смарт-контрактів. Це може бути від простих угод купівлі-продажу до складних бізнес-угод. Веб-застосунки дозволяють сторонам автоматизувати процеси, забезпечити безпеку та ефективність виконання угод.

Це лише кілька прикладів веб-застосунків, які можуть використовувати предмет даної роботи. З блокчейн технологією та смарт-контрактами можна реалізувати різноманітні інноваційні проекти, які покращують ефективність та надійність різних сфер діяльності.

1.4 Сучасні технічні можливості:

Сучасні технічні можливості в контексті використання блокчейн технології для створення смарт-контрактів включають:

1. Розподілена база даних: Блокчейн технологія дозволяє створювати розподілену базу даних, яка зберігається на багатьох комп'ютерах у мережі. Це забезпечує високу надійність, доступність та стійкість до змін, оскільки інформація розподіляється і реплікується між вузлами мережі.

2. Криптографічна безпека: Блокчейн технологія використовує криптографічні алгоритми для забезпечення безпеки даних та угод. Це включає шифрування даних, цифрові підписи та механізми перевірки та підтвердження транзакцій, що робить систему майже недоступною для зловмисників.

3. Смарт-контракти: Смарт-контракти є програмами, які автоматизують та виконують угоди на блокчейні. Вони можуть автоматично контролювати, виконувати та управляти умовами угод, що дозволяє ефективно та безпечно виконувати операції без потреби посередників.

4. Скалкість: Останні технологічні вдосконалення у блокчейн технології, такі як розвиток протоколів Proof-of-Stake (PoS) та Proof-of-Authority (PoA), дозволяють покращити швидкість обробки транзакцій та масштабованість

мережі. Це робить блокчейн технологію більш придатною для широкого кола застосувань.

5. Інтеграція з іншими технологіями: Блокчейн може бути інтегрований з іншими сучасними технологіями, такими як штучний інтелект (AI), Інтернет речей (IoT) та хмарні обчислення. Це відкриває шлях до розробки нових рішень та послуг, які комбінують силу різних технологій для досягнення більшої ефективності та інноваційності.

Ці сучасні технічні можливості створюють незліченні можливості для використання блокчейн технології та смарт-контрактів у різних сферах діяльності, приводячи до автоматизації процесів, забезпечення безпеки та створення нових інноваційних рішень.

РОЗДІЛ 2 ОГЛЯД ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ

2.1 Середовище розробки - Visual Studio Code

Visual Studio Code (VS Code) є популярним середовищем розробки, розробленим компанією Microsoft. Воно є відкритим та безкоштовним редактором, який надає широкі можливості для програмістів і розробників у різних мовах програмування.

Особливості Visual Studio Code:

1. Множина мов програмування: VS Code підтримує багато мов програмування, включаючи популярні такі, як JavaScript, Python, Java, C++, HTML/CSS, PHP і багато інших. Це дозволяє розробникам працювати з різними проектами в єдиному редакторі.

2. Розширення та плагіни: VS Code надає потужну систему розширень, що дозволяє розширити його функціональність та налаштувати під власні потреби. Існує велика кількість доступних розширень, які додають підтримку конкретних мов програмування, забезпечують інтеграцію з системами контролю версій, надають інструменти для форматування коду, налагодження та багато іншого.

3. Інтегрована консоль: VS Code має вбудовану консоль, яка дозволяє виконувати команди та запускати програми безпосередньо з редактора. Це зручно для тестування та налагодження коду без переходу до зовнішнього терміналу.

4. Підсвічування синтаксису та автодоповнення: VS Code автоматично визначає мову програмування використовуваного файлу і надає підсвічування

синтаксису, що полегшує читання та редагування коду. Також, він пропонує автодоповнення коду, що прискорює процес розробки та допомагає уникнути помилок.

5. Інтеграція з системами контролю версій: VS Code має підтримку популярних систем контролю версій, таких як Git. Це дозволяє розробникам здійснювати коміти, відстежувати зміни, порівнювати та об'єднувати гілки безпосередньо в редакторі.

6. Налаштування та зручний інтерфейс: VS Code надає багато налаштувань, що дозволяють адаптувати редактор під власні потреби. Він має інтуїтивний та зручний інтерфейс, з можливістю розташування редакторів біч-бічно або вертикально, розділенням вкладок, швидким перемиканням між файлами та багато іншого.

В цілому, Visual Studio Code є потужним та гнучким середовищем розробки, яке надає зручні інструменти для програмістів у різних мовах програмування. Його розширюваність та активна спільнота розробників роблять його популярним вибором для багатьох професіоналів.

2.2 Мова програмування JavaScript та її особливості

JavaScript є однією з найпопулярніших мов програмування, яка використовується для розробки веб-додатків та веб-сторінок. Вона є скриптовою мовою програмування, що виконується безпосередньо в браузері клієнта.

Особливості JavaScript:

1. Клієнтська мова: JavaScript використовується для створення динамічних та інтерактивних елементів на веб-сторінках. Вона дозволяє керувати подіями, змінювати вміст сторінок, створювати анімацію та взаємодіяти з користувачем без необхідності перезавантаження сторінки.

2. Широке застосування: JavaScript може використовуватися як на стороні клієнта (Front-End), так і на стороні сервера (Back-End). На стороні клієнта, вона використовується для розробки веб-додатків, взаємодії з DOM (Document Object Model) та роботи з AJAX (Asynchronous JavaScript and XML). На стороні сервера, JavaScript використовується за допомогою платформи Node.js для створення швидких та масштабованих веб-серверів.

3. Простота вивчення: JavaScript має простий і зрозумілий синтаксис, що дозволяє новим розробникам швидко вивчити мову. Вона базується на об'єктному підході та використовує прототипне наслідування. Багато розробників, які вже знайомі з іншими мовами програмування, можуть легко перейти до JavaScript.

4. Розширюваність: JavaScript має велику кількість бібліотек та фреймворків, що розширюють його функціональність та спрощують розробку веб-додатків. Популярні фреймворки, такі як React, Angular і Vue.js, дозволяють розробникам будувати складні і масштабовані клієнтські додатки.

5. Підтримка веб-стандартів: JavaScript активно підтримується браузерами та оновлюється з урахуванням останніх веб-стандартів. Це дозволяє розробникам використовувати нові функції та можливості, що полегшують роботу з мовою та забезпечують сумісність з різними браузерами.

JavaScript є потужним і розширюваним інструментом для розробки веб-додатків. Вона дозволяє створювати інтерактивні, динамічні та високопродуктивні веб-застосунки, забезпечуючи зручний інтерфейс та покращений користувацький досвід.

2.3 Мова програмування Solidity та її особливості:

Solidity - це мова програмування, спеціально розроблена для розуміння та реалізації смарт-контрактів на платформі Ethereum. Вона є високорівневою, об'єктно-орієнтованою мовою, що базується на синтаксисі JavaScript.

Особливості Solidity:

1. Спрощена розробка смарт-контрактів: Solidity надає потужні можливості для розробки смарт-контрактів, що використовуються на платформі Ethereum. Вона дозволяє визначати структури даних, функції та події, що забезпечують логіку роботи контракту. Solidity також підтримує успадкування та багато інших парадигм програмування.

2. Розуміння блокчейн-логіки: Solidity дозволяє програмістам взаємодіяти з розподіленою блокчейн-мережею. Вона має вбудовані функції для доступу до інформації про поточний блок, адреси контракту та розподіленого реєстру. Це дозволяє легко реалізувати умови, валідацію та контроль доступу в смарт-контрактах.

3. Безпека: Solidity надає інструменти та механізми безпеки для розробки смарт-контрактів. Вона має вбудовані механізми для запобігання вразливостям,

таким як переповнення цілочисельних значень, атаки на заморозку та рекурсивні атаки.

4. Інтеграція з Ethereum Virtual Machine (EVM): Solidity компілюється в машинний код, який може бути виконаний на Ethereum Virtual Machine (EVM). Це дозволяє смарт-контрактам, написаним на Solidity, виконуватись на платформі Ethereum та взаємодіяти з іншими контрактами та користувачами.

5. Розширюваність: Solidity підтримує бібліотеки, які дозволяють розробникам повторно використовувати код та функціональність. Це сприяє покращенню ефективності та зменшенню витрат при розробці смарт-контрактів.

Solidity є основною мовою програмування для розробки смарт-контрактів на платформі Ethereum. Вона забезпечує потужні можливості для визначення та реалізації логіки контрактів та взаємодії з блокчейн-мережею.

2.4 Програмна платформа Node.js та її особливості:

Node.js - це відкрите середовище виконання, побудоване на двигуні V8 JavaScript, який раніше використовувався в браузері Google Chrome. Воно дозволяє виконувати JavaScript на стороні сервера і забезпечує швидке та ефективне виконання веб-додатків.

Особливості Node.js:

1. Асинхронна та подієва модель: Node.js використовує подієву та асинхронну модель програмування, що дозволяє ефективно обробляти багато запитів

одночасно. Це особливо важливо для розробки серверних додатків, де можуть бути одночасні запити від багатьох користувачів.

2. Висока продуктивність: Завдяки використанню двигуна V8 JavaScript, Node.js має високу продуктивність. Він виконує оптимізацію коду, що дозволяє досягти швидкого виконання програм.

3. Розширюваність: Node.js має велику кількість модулів, доступних через систему керування пакетами npm. Це дозволяє розробникам легко використовувати готові модулі та бібліотеки для розширення функціональності їхніх додатків.

4. Єдина мова програмування: Node.js дозволяє використовувати JavaScript як на стороні сервера, так і на стороні клієнта, що спрощує розробку та обмін кодом між клієнтською та серверною частинами додатків.

5. Широке співтовариство: Node.js має активну та велику спільноту розробників. Це означає, що є багато ресурсів, документації, модулів та інструментів, які підтримуються спільнотою та сприяють швидкому розвитку додатків.

Node.js є потужним середовищем розробки для створення серверних додатків та веб-сервісів. Його асинхронна та подієва модель дозволяє створювати ефективні та масштабовані додатки, а велика спільнота розробників забезпечує підтримку та інновації у цій області.

2.5 Фреймворк Truffle та його осоливості:

Truffle - це фреймворк для розробки, тестування та впровадження смарт-контрактів на платформі Ethereum. Він надає набір інструментів і середовище для спрощення процесу розробки децентралізованих додатків із використанням смарт-контрактів.

Особливості Truffle:

1. Компіляція та управління контрактами: Truffle забезпечує зручний інтерфейс для компіляції смарт-контрактів написаних на мові Solidity. Він автоматично виявляє зміни в контрактах і надає можливість оновлення контрактів безпосередньо на блокчейні.

2. Розробка та управління міграціями: Truffle дозволяє легко розробляти та керувати міграціями смарт-контрактів на блокчейн. Це дозволяє зручно впроваджувати та оновлювати контракти на різних етапах розробки.

3. Тестування смарт-контрактів: Truffle надає вбудовану систему тестування для смарт-контрактів. З його допомогою можна створювати автоматизовані тести для перевірки функціональності, безпеки та правильності роботи контрактів.

4. Інтерактивна консоль: Truffle надає інтерактивну консоль, яка дозволяє взаємодіяти з контрактами та блокчейном безпосередньо з командного рядка. Це дозволяє швидко перевіряти функціональність та взаємодіяти з контрактами під час розробки.

5. Інтеграція з іншими інструментами: Truffle можна легко інтегрувати з іншими популярними інструментами, такими як Ganache (локальний блокчейн), Metamask (розширення для браузера) та іншими, що полегшує розробку та взаємодію з додатками на базі Ethereum.

Truffle є потужним інструментом для розробки смарт-контрактів, який спрощує процес розробки та тестування децентралізованих додатків на платформі Ethereum. Він надає зручність, ефективність та безпеку у розробці смарт-контрактів.

2.6 Бібліотека Web3 API та його особливості:

Web3 API - це набір інструментів та бібліотек, які дозволяють взаємодіяти з блокчейнами на основі технології Web3, зокрема Ethereum. Цей API надає розробникам доступ до функцій блокчейну, таких як взаємодія з смарт-контрактами, читання та запис даних на блокчейні, керування гаманцями та багато іншого.

Особливості Web3 API:

1. Взаємодія з смарт-контрактами: Web3 API дозволяє взаємодіяти з смарт-контрактами, що базуються на блокчейні. З його допомогою можна викликати функції контракту, передавати параметри та отримувати результати виконання контракту.

2. Керування транзакціями: Web3 API надає можливість створювати та відправляти транзакції на блокчейн. Це дозволяє виконувати операції, такі як переказ ефірів, запис даних на блокчейні та виконання функцій смарт-контрактів.

3. Робота з гаманцями: З допомогою Web3 API можна створювати та керувати гаманцями. Він дозволяє генерувати ключі, створювати адреси гаманців, підписувати транзакції та перевіряти баланси рахунків.

4. Читання та запис даних на блокчейні: Web3 API дозволяє читати дані з блокчейну, такі як стан смарт-контрактів, історію транзакцій та іншу інформацію, що зберігається на блокчейні. Також він надає можливість записувати дані на блокчейн, змінювати стан смарт-контрактів та взаємодіяти з додатками на блокчейні.

5. Підтримка різних блокчейнів: Web3 API може працювати з різними блокчейнами, а не лише з Ethereum. Він підтримує взаємодію з блокчейнами, які підтримують технологію Web3, що розширює його можливості і дозволяє розробникам працювати з різними блокчейнами з використанням єдиного API.

Web3 API є важливим інструментом для розробки децентралізованих додатків та взаємодії з блокчейнами. Він дозволяє розробникам ефективно використовувати можливості блокчейну та інтегрувати їх у свої додатки та сервіси.

РОЗДІЛ 3 ДОДАТКОВІ ВІДОМОСТІ ПРО ТЕХНОЛОГІЮ

3.1 Блокчейн, загальні відомості

Історія блокчейну починається зі створення першої криптовалюти - Bitcoin, в 2008 році, людиною або групою людей під псевдонімом Сатоші Накамото. Bitcoin використовував новаторську технологію, відому як блокчейн, для реалізації децентралізованої та безпечної системи електронних платежів.

Суть блокчейну полягає в тому, що він є розподіленою базою даних, яка зберігається на безлічі комп'ютерів, відомих як вузли мережі. Кожен вузол має копію всіх транзакцій, що сталися на мережі. Ці транзакції групуються в блоки, які зв'язані між собою за допомогою криптографічних хеш-функцій. Кожен блок містить хеш попереднього блоку, що утворює ланцюг блоків, відомий як блокчейн.

Основні властивості блокчейну включають:

1. Децентралізація: Блокчейн не має центральної влади або сервера, що контролює його. Він базується на розподіленій мережі вузлів, які спільно підтримують та підтверджують транзакції.

2. Безпека: Блокчейн використовує криптографію для захисту від фальсифікації та змін даних. Кожен блок містить хеш попереднього блоку, що утворює ланцюг, що не може бути легко змінений без перерахунку всіх наступних блоків.

3. Прозорість: Усі транзакції, що сталися на блокчейні, є публічно доступними та перевіряемими. Кожен учасник мережі може переглядати історію транзакцій, що робить систему прозорою та довіренною.

4. Незмінність: Коли блок додається до блокчейну, він стає незмінним. Це означає, що він не може бути вилучений або змінений без консенсусу всіх учасників мережі.

Після створення Bitcoin блокчейн знайшов застосування в інших галузях, включаючи фінанси, логістику, ланцюжки постачання та громадські реєстри. Технологія блокчейну розширилася на інші криптовалюти та платформи, такі як Ethereum, що дозволяють створювати смарт-контракти та додатки з використанням блокчейну. Сьогодні блокчейн продовжує розвиватися та знаходити нові застосування у багатьох галузях, змінюючи спосіб, яким ми спілкуємося, торгуємо та зберігаємо дані.

3.2 Смарт-контракти:

Смарт-контракти є центральною складовою технології блокчейну, яка дозволяє автоматизувати та виконувати угоди або умови без посередництва та довіряючи програмному коду. Вони є саморегульованими програмами, які працюють на блокчейні та виконуються автоматично без необхідності сторонньої втручаності.

Особливості смарт-контрактів:

1. Самовиконання: Смарт-контракти виконуються автоматично, як тільки виконуються певні задані умови. Вони не потребують посередників або центральних органів, що забезпечує швидке та безпечне виконання угод.

2. Безпека: Смарт-контракти побудовані на базі криптографії та блокчейну, що забезпечує високий рівень безпеки. Вони неможливі для вилучення, зміни або порушення без згоди всіх учасників мережі.

3. Децентралізація: Смарт-контракти працюють на розподіленій мережі вузлів блокчейну, що дозволяє уникнути залежності від централізованих органів контролю. Вони можуть бути запущені та виконані безпосередньо між учасниками мережі.

4. Програмованість: Смарт-контракти можуть бути написані за допомогою програмного коду, що дозволяє визначати різні умови та дії. Це дає можливість створювати різноманітні логіки та функціональність угод, які автоматично виконуються при виконанні певних умов.

Смарт-контракти знаходять застосування в різних галузях, включаючи фінанси, логістику, нерухомість, громадські реєстри та багато інших. Вони можуть використовуватися для автоматизації процесів, управління цифровими активами, забезпечення безпеки та надійності угод, а також для створення децентралізованих додатків (DApps). Смарт-контракти є ключовим інструментом у розвитку блокчейн-технологій та відкривають нові можливості для ефективної та безпечної автоматизації угод.

РОЗДІЛ 4 РОЗРОБКА ТА РЕАЛІЗАЦІЯ СМАРТ-КОНТРАКТУ ЗА ДОПОМОЮ ТЕХНОЛОГІЇ БЛОКЧЕЙН

4.1 Загальні відомості:

Найменування проекту: «NFT коллекція».

Проект є кросплатформенним та може бути запущенним під управлінням операційних систем MS WINDOWS, Linux, Mac OS. Для написання використано мови программування: Solidity, JavaScript.

Програма використовує бібліотеки chai та mocha для тестування проекту, фреймворк Node.js, станднарти смарт контрактів OpenZeppelin та стандартні бібліотеки Solidity.

Проект працює з вхідними даними користувача. Після того, як користувач ввів усі дані, він запускає скрипт, який публікує програму в мережі Ethereum та може взаємодіяти з нею за допомогою Minting page. Вихідними даними є результат публікації програми та “мінтинг”(створення) токенів.

4.2 Функціональне призначення:

4.2.1 Постановка задачі.

Програма призначена для створення NFT коллекції.

Користувач повинен вставити посилання на сховище данних NFT(картинки та їх метаданні).

Далі роботу програмного засобу проілюструємо за допомогою опису роботи відповідних процедур.

Вхідні дані використовуються для відображення метаданних токена, його картинки на торговому майданчику та публікації програми до мережі Ethereum.

Після завершення публікації програми користувач може “замінтить” свій унікальний токен. Він буде єдиним власником цього токена, що можна буде довести.

Функціональні обмеження: всього можна “замінтить” тільки 100 токенів, максимально можна отримати тільки 3 токена за транзакцію.

4.2.2 Алгоритм створення NFT коллекції.

1. Знаходячись в папці проекту в консолі потрібно прописати “npm install”, щоб встановити всі необхідні бібліотеки.

2. Далі в файлі .link потрібно вставити посилання на сховище NFT метадати.

3. В файлі .secret потрібно вставити мнемонічну фразу криптогаманця(це потрібно для того, щоб з цього гаманця опублікувати програму до мережі).

4. Далі в консолі потрібно викликати скрипт deploy.js. Пишем в консолі “npm run deploy”.

5. Далі запускаєм minting page і “мінтим” наш токен

6. На сайті бачимо повідомлення, що ми створили токен.

7. Заходим на торговий майданчик https://testnets.opensea.io/ , авторизуємось і бачимо наш токен разом із колекцією яку ми створили.

4.3 Опис логічної структури

Розроблена програма використовує наступні алгоритми:

− алгоритм створення коллекції;

− алгоритм зчитування інформації з полів для введення даних;

Проект складається з файлів: DragonEye.sol, Migrations.sol, deploy.js, 1_initial_migrations.js, 2_deploy_contracts.js, index.html, style.css, web3.js

У файлі DragonEye.sol прописаний алгоритм створення токена та взаємодії з ним по стандарту протокола ERC-721, у файлі deploy.js написаний скрипт який компілює весь код і публікує його до тестової мережі, у файлах index.html, style.css, web3.js написаний Minting page.

Логічна структура програми представлена UML-діаграмами:

4.4 Симтемні вимоги:

Для роботи програми необхідне використання персонального комп’ютера. Мінімальні системні вимоги:

− процесор: Intel Celeron 2 ГГц;

− ОЗУ: 200 Мб;

− Місце на диску: 20 Мб;

− Операційна система: Linux/Windows 7/8/10/Mac OS.

4.5 Виклик і завантаження

Для роботи програми необхідні наступні файли: Truffle, OpenZeppelin Contracts, Node.js, бібліотеки: chai, mocha, HTML 5, CSS, Web3 API

Для початку роботи необхідно скачати усі необхідні бібліотеки та запустити скрипт deploy.js.

4.6 Вхідні та вихідні дані:

Вхідними даними програми є:

–URL до сховища метадати NFT;

– Назву та символ коллекції.

Вихідними даними програми є:

– Смарт-контракт коллекції

–Готова коллекція, яка буде відображена на найпопулярнішому маркетплейсі

–Minting page на якому можна буде “замінтить” токен.

Вимоги до проекту

Проєкт призначений для створення NFT коллекції.
Обсяг проєкту зі всіма текстами 179 Мб.
Працездатний обсяг проєкту 100 Мб.
При написанні програми використана операційна система MS WINDOWS, Visual Studio Code, Тестова мережа Ethereum під назвою Goerli,

Photoshop, скрипт генерації NFT images під назвою HashLips Art Generator, Truffle, OpenZeppelin Contracts, Node.js, бібліотеки: chai, mocha, HTML 5, CSS, Web3 API. Мови програмування: Solidity, JavaSript.

Мінімальні вимоги до апаратного забезпечення:
− процесор: Intel Celeron 2 ГГц;
− ОЗУ: 200 Мб;
− Місце на диску: 20 Мб;
− Операційна система: Linux/Windows 7/8/10/Mac OS.

Вхідними даними програми є:

–URL до сховища метадати NFT;

– Назву та символ коллекції.

Результатом роботи проєкту є готова NFT колекція з усім її можливим функціоналом,тю яка буде відображена на тестовій версії самого популярного маркетплейса під назвою OpenSea та Minting Page для “мінтингу” NFT.

Вимоги до інсталяції проєкту: потрібно скачати node.js і в корінній папці проекту вставити всі необхідні файли(написати npm install).

Вимоги до з розповсюдження проєкту: програма є вільною для росповсюдження.

Контроль правильності забезпечується на рівні тестового прикладу.

4.7 Призначення та умови застосування:

Проєкт призначений для створення NFT коллекції
Вхідними даними програми є:

–URL до сховища метадати NFT;

– Назву та символ коллекції.

Функціональні обмеження: користувач не може замінтить більше 3-х NFT за транзакцію, обмежений доступ до деяких функцій(є доступ тільки власнику програми).

Вихідними даними програми є:

– Смарт-контракт коллекції

–Готова коллекція, яка буде відображена на найпопулярнішому маркетплейсі

–Minting page на якому можна буде “замінтити” токен.

Вихідні данні відображаються в консолі та на Minting Page.

Мінімальні системні вимоги для функціонування програми: персональний комп’ютер з процесором Intel Celeron 2 ГГц, ОЗУ 200 Мб, ПЗУ 20 Мб, операційна система Linux/Windows 7/8/10/Mac OS.

Умови застосування – вільний доступ.

4.8 Структура програми:

Основний код програми розміщений у файлі: DragonEye.sol.

Програма складається з модулів:

1. ERC721
2. ERC721Enumerable

Модуль ERC721 складається з наступних методів:

balanceOf(owner) - Повертає кількість токенів в обліковому записі власника.

ownerOf(tokenId) - Повертає власника маркера tokenId.

safeTransferFrom(from, to, tokenId) - Безпечно передає токен tokenId від до до.

transferFrom(from, to, tokenId) - Передає токен tokenId від до до.

approve(to, tokenId) - Надає дозвіл на передачу токена tokenId до іншого облікового запису. Схвалення очищається, коли маркер передається.

setApprovalForAll(operator, _approved) - Затвердити або видалити оператора як оператора для абонента. Оператори можуть викликати transferFrom або safeTransferFrom для будь-якого токена, що належить абоненту.

getApproved(tokenId) - Повертає обліковий запис, затверджений для токена tokenId.

isApprovedForAll(owner, operator) - Повертає, якщо оператору дозволено керувати всіма активами власника.

Модуль ERC721Enumerable складається з наступних методів:

tokenOfOwnerByIndex(owner, index) -

totalSupply() - Повертає загальну кількість токенів, збережених контрактом.

tokenByIndex(index) - Повертає ідентифікатор маркера, що належить власнику, за заданим індексом його списку маркерів. Використовуйте разом із balanceOf, щоб перерахувати всі токени власника

4.9 Звернення до програми:

Для роботи програми необхідні наступні файли: Truffle, OpenZeppelin Contracts, Node.js, бібліотеки: chai, mocha, HTML 5, CSS, Web3 API

Для початку роботи необхідно скачати усі необхідні бібліотеки та запустити скрипт deploy.js.

4.10 Перевірка програми:

1) Оновлюєм всі необхідні файли:

```
D:\learn\2 course\mycourse>npm i

up to date, audited 635 packages in 3s

83 packages are looking for funding
  run `npm fund` for details

4 moderate severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```

Рис.2.1-Результат оновлення необхідних файлів для того, щоб смарт-контракт на актуальній версії можна було опубліквати в мережі.

2) Публікуєм смарт-контрак в мережі:

```
> my-course@1.0.0 deploy
> node deploy.js

  Deploying and updating contracts...

Compiling your contracts...
===========================
√ Fetching solc version list from solc-bin. Attempt #1
√ Fetching solc version list from solc-bin. Attempt #1
> Compiling .\contracts\DragonEye.sol
> Compiling .\contracts\DragonEye.sol
> Compiling .\contracts\Migrations.sol
> Artifacts written to D:\learn\2 course\mycourse\build\contracts
> Compiled successfully using:
   - solc: 0.8.17+commit.8df45f5f.Emscripten.clang



Starting migrations...
======================
> Network name:    'goerli'
> Network id:      5
> Block gas limit: 30000000 (0x1c9c380)


1_initial_migration.js
======================

   Replacing 'Migrations'
   ----------------------
   > transaction hash:    0x7d27abfb919e175d6284227fdb0f7be061a662e54386f4959f949b1625eaec
   > Blocks: 1            Seconds: 12
   > contract address:    0x3f7f6b4CD68b8416317200F00545A245BE947aB0
   > block number:        8110909
   > block timestamp:     1670687436
   > account:             0x5d9D2C5783E43486353ed112A49d6067042382b5
   > balance:             1.078177093387015914
   > gas used:            155222 (0x25e56)
   > gas price:           2.500076172 gwei
   > value sent:          0 ETH
   > total cost:          0.000388066823570184 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:     0.000388066823570184 ETH
```

Рис.2.2 – Результат публації смарт-контрактів в мережі, на даному рисунку ми можемо спостерігати як наш контракт відображається в терміналі, які ресурси були задіяні, вартість публікації, а також хеш-код трнзакції(публікації).

```
2_deploy_contracts.js
=====================

   Replacing 'DragonEye'
   ---------------------
   > transaction hash:    0xb3ddc9ae6608d47d832cdf7344dba291c577cced6bdde28db36d90ade1460809
   > Blocks: 2            Seconds: 20
   > contract address:    0xDAB079c4E224489b65D89A2A16e29806D93ae5d3
   > block number:        8110912
   > block timestamp:     1670687484
   > account:             0x5d9D2C5783E43486353ed112A49d6067042382b5
   > balance:             1.072948329902898764
   > gas used:            2045753 (0x1f3739)
   > gas price:           2.50007438 gwei
   > value sent:          0 ETH
   > total cost:          0.00511453466310814 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:     0.00511453466310814 ETH


Summary
=======
> Total deployments:   2
> Final cost:          0.005502601486678324 ETH
```

Рис.2.3 - Результат публації смарт-контрактів в мережі, на даному рисунку можемо спостерігати переміщення нашого контракту на маркетплейс, вартість операції, а також створення та назву колекції.

3) Запускаєм index.html на локальному сервері:

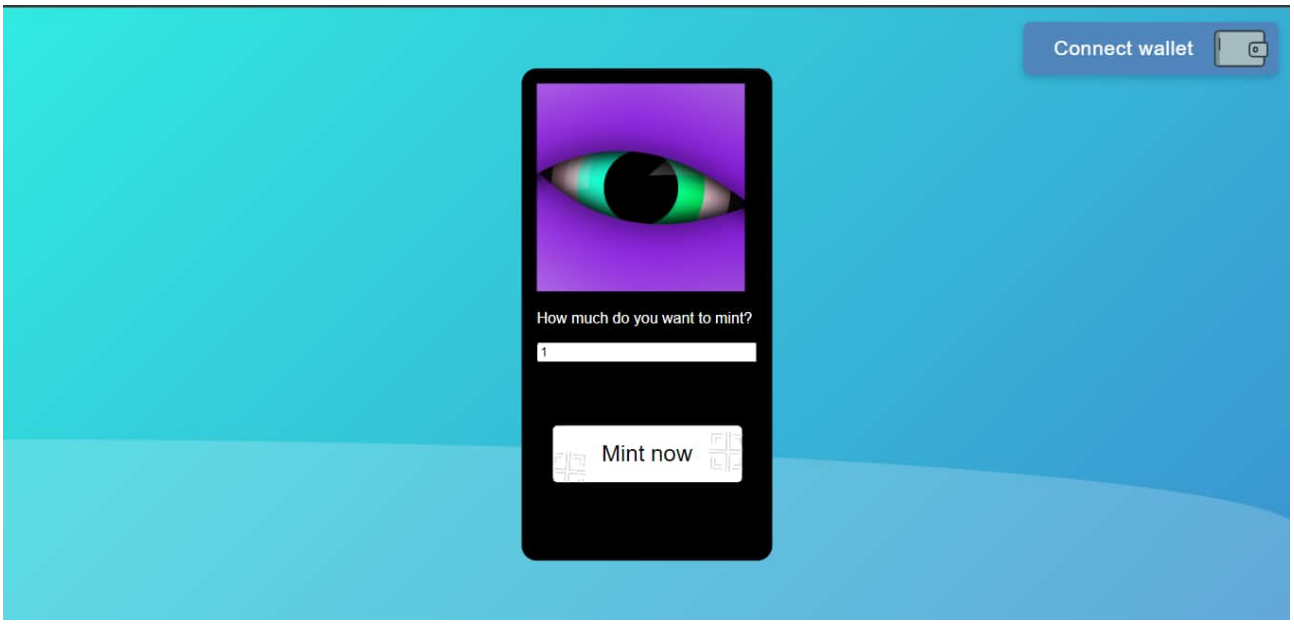

Рис.2.4 – Зовнішній вигляд Minting Page, додаткового ресурсу, за допомогою якого ми маємо можливість запустити контракт через веб сторінку, а не через VS Code.

4) Підключаєм криптогаманець:

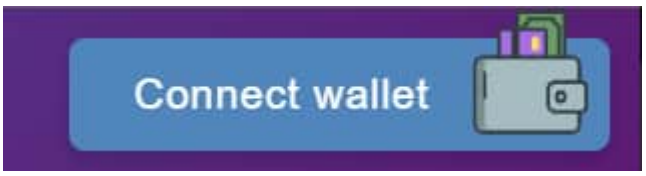

Рис.2.5 – Кнопка підключення гаманця, з якого буде оплата публікації, та який виступає власником смарт-контракту

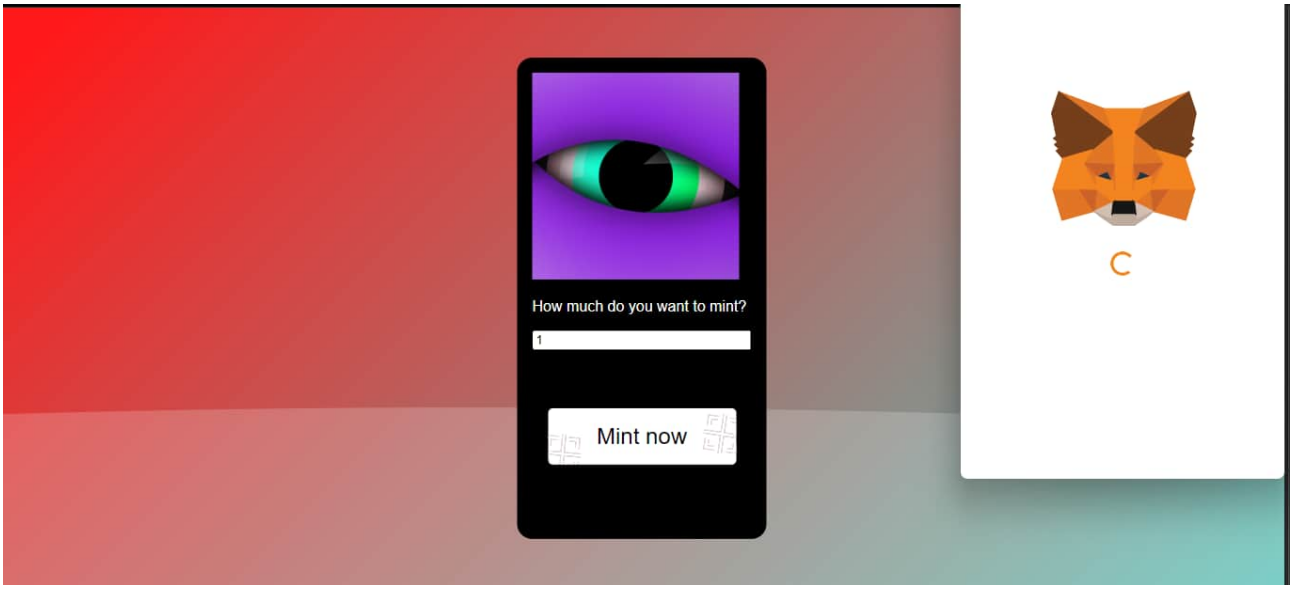

Рис.2.6 – Поп-ап криптогаманця Meta Mask, який ми використовуємо для оплати та публікації смарт-контракту

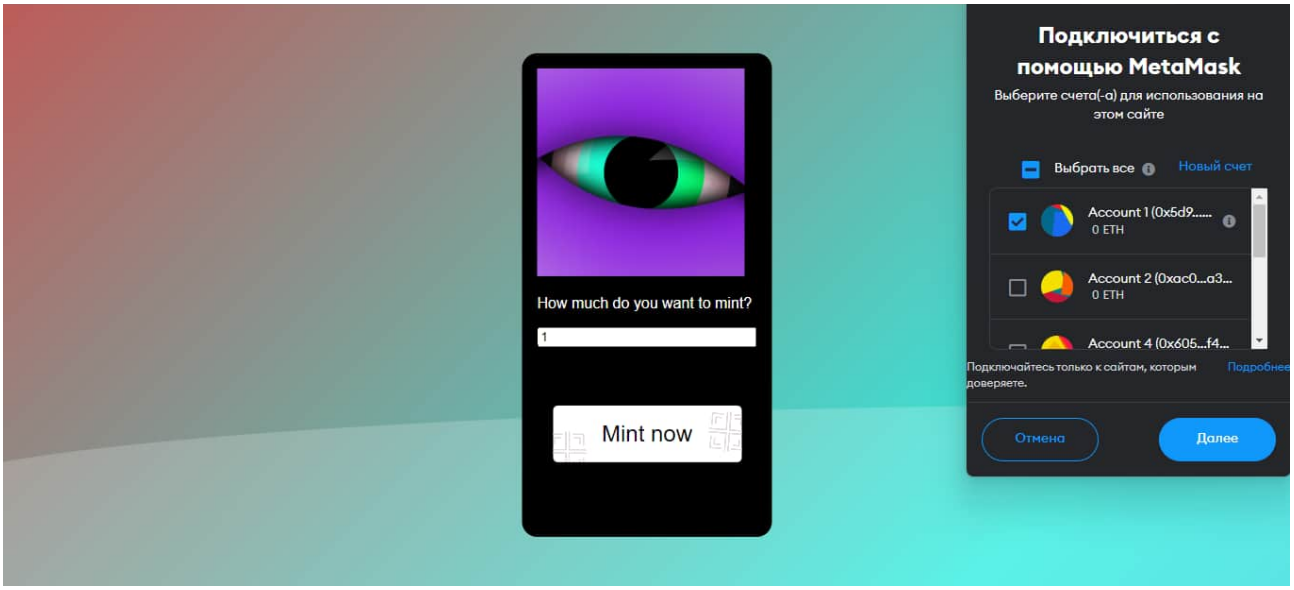

Рис.2.7 – Процес підключення криптогаманця, за допомогою крипто-кошелька, згаданого в описі до попереднього рисунку.

5) Пишем скільки ми хочем “замінтити токенів” і натискаємо на кнопку “Mint now”.

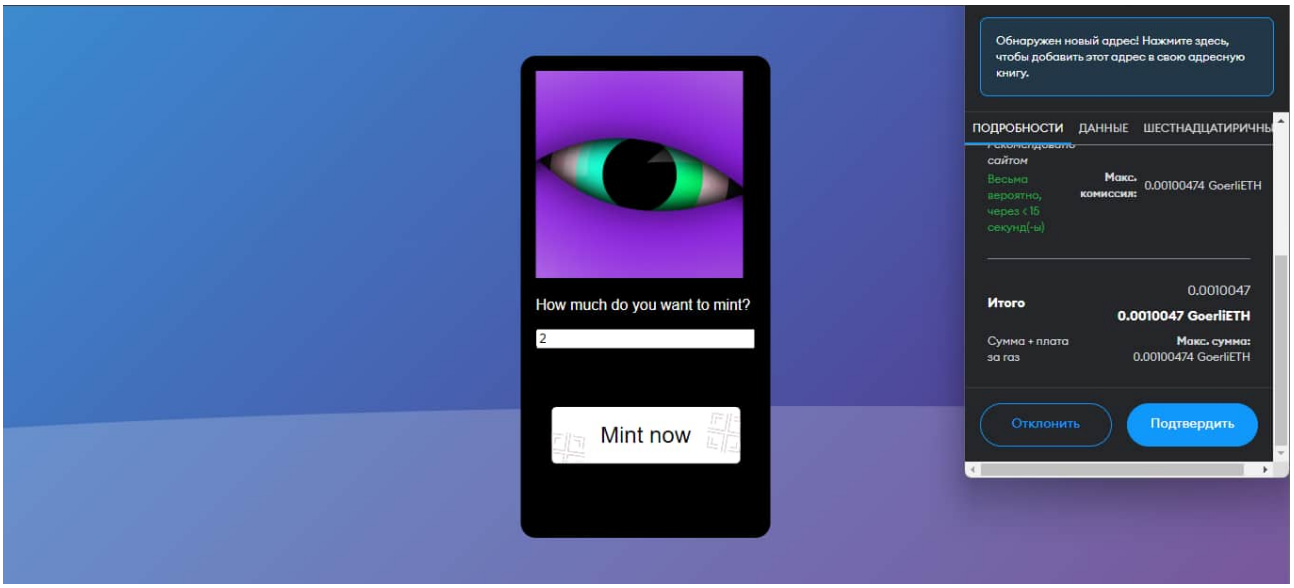

Рис.2.8 – “Мінтинг” токенів, фінальне налаштування гаманця і підписання смарт-контракту для того, щоб запустити його в дію.

Підтверджуєм транзакцію.

Чекаєм поки токен “Замінтиться”.

6) Бачим повідомлення про результат

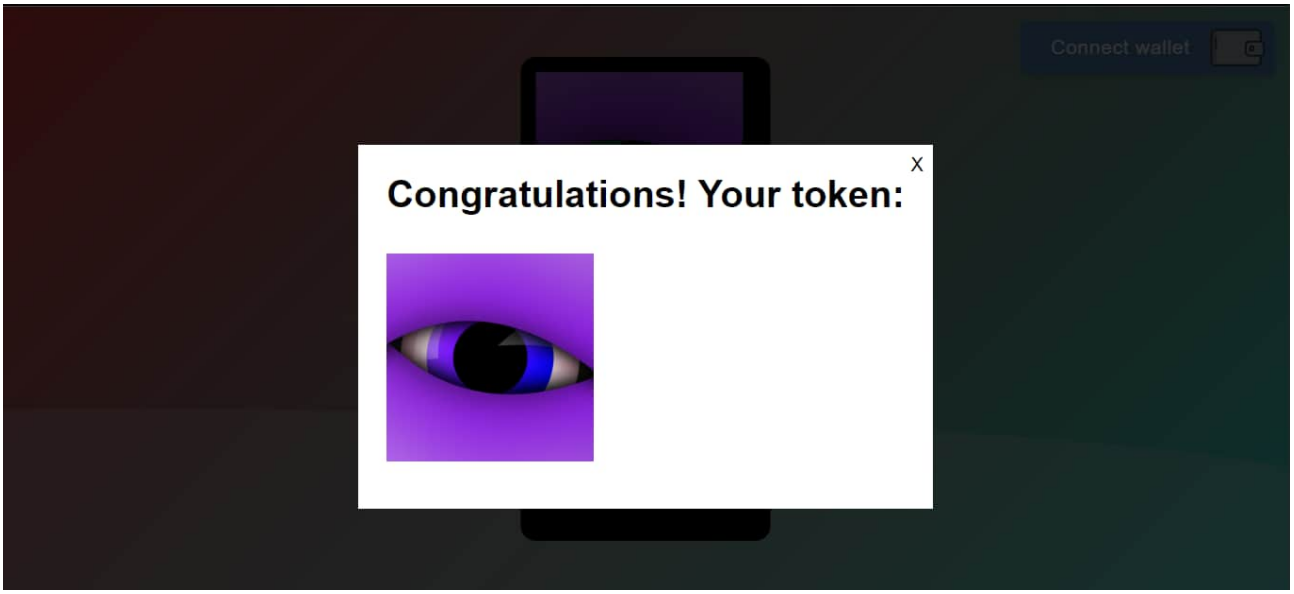

Рис.2.9 – Результат “мінтингу”, додаткове вікно, яке інформує нас про успіх транзакції та публікацію нашого смарт-контракту в мережі

7) Далі заходим на https://testnets.opensea.io/account

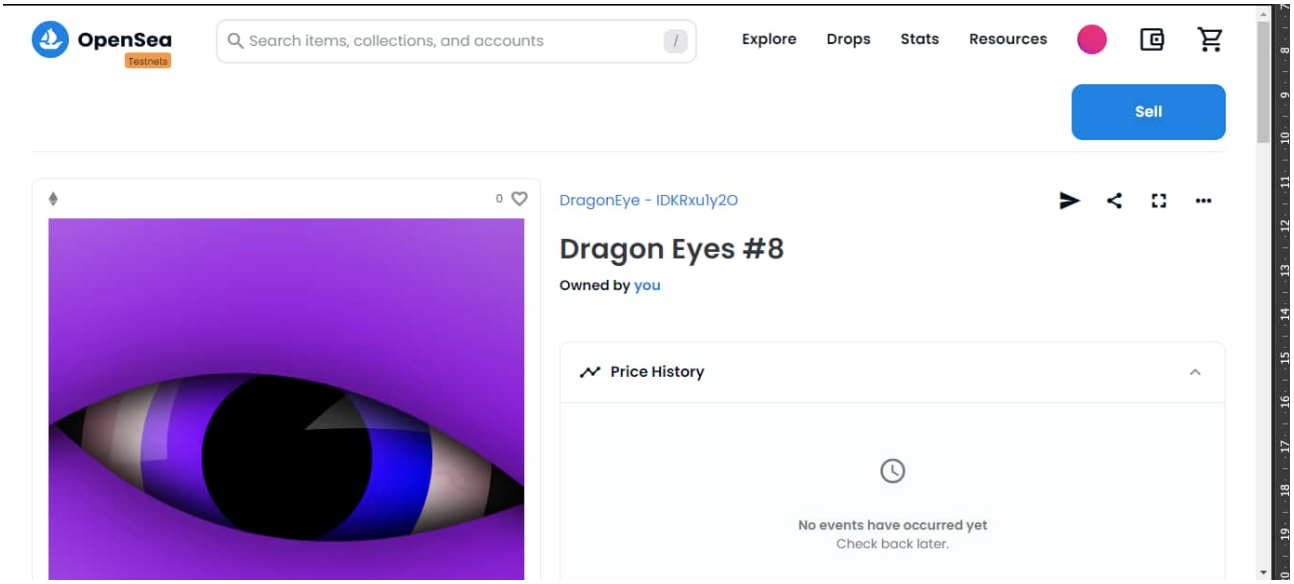

Рис.2.10 – Сторінка токена на торговому майданчику, на якій за допомогою механік смарт-контракту любий користувач може укласти контракт на передачу права власності на наш токен.

І бачим наш токен! Його можна продивитись за посиланням https://testnets.opensea.io/collections

ВИСНОВКИ

У даній дипломній роботі було проведено детальне дослідження та розробка на тему "Використання блокчейн-технології для створення смарт-контракту". Були розглянуті основні аспекти технології блокчейн, її історія, принципи роботи та переваги. Особлива увага була приділена смарт-контрактам, які є центральним елементом блокчейн-технології.

В рамках дослідження було проведено аналіз стану проблемної області, де було виявлено значний потенціал блокчейн-технології та смарт-контрактів у різних галузях, включаючи фінанси, логістику, нерухомість та громадські реєстри. Також були розглянуті сучасні технічні можливості, такі як Visual Studio Code, JavaScript, Solidity, Node.js, Truffle та Web3 API, які були використані для розробки та застосування смарт-контрактів.

Результати дослідження підтвердили, що використання блокчейн-технології для створення смарт-контрактів має значний потенціал у багатьох галузях. Вона дозволяє автоматизувати та забезпечувати безпеку угод, ефективно управляти цифровими активами та створювати децентралізовані додатки. Використання такої технології може призвести до зниження витрат, поліпшення ефективності та підвищення довіри між учасниками.

Загальний висновок полягає в тому, що блокчейн-технологія та смарт-контракти є важливими інструментами для майбутнього розвитку цифрового світу. Їх використання може відкрити нові можлив

ості для ефективної та безпечної автоматизації угод та процесів. Дана робота є важливим кроком у дослідженні та розвитку блокчейн-технологій та сприяє впровадженню їх у реальному суспільстві.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. William Entriken, Dieter Shirley, Jacob Evans, Nastassia Sachs, EIP-721: Non-Fungible Token Standard, 2018-01-24,
2. HashLips Art Engine, MIT license, https://github.com/HashLips/hashlips_art_engine
3. Смарт-промисловість в епоху цифрової економіки: перспективи, напрями і механізми розвитку : монографія / В.П. Вишневський та ін. ; за ред. В.П. Вишневського. Київ : Інститут економіки промисловості, 2018. 192 с
4. Paul Rosenzweig (August 19, 2017). Blockchain Standards. The Lawfare website. August 19, 2017. URL: https://www.lawfareblog.com/blockchain-standards .
5. Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System.
6. Buterin, V. (2014). Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform.
7. Swan, M. (2015). Blockchain: Blueprint for a New Economy.
8. Antonopoulos, A. M. (2017). Mastering Ethereum: Building Smart Contracts and DApps.
9. Wood, G. (2018). Ethereum: A Secure Decentralized Generalized Transaction Ledger.
10. Solidity Documentation. Available at: https://docs.soliditylang.org/
11. Node.js Documentation. Available at: https://nodejs.org/en/docs/
12. Truffle Documentation. Available at: https://www.trufflesuite.com/docs/truffle/overview
13. Web3.js Documentation. Available at: https://web3js.readthedocs.io/
14. Microsoft Visual Studio Code Documentation. Available at: https://code.visualstudio.com/docs
15. Ethereum Whitepaper. Available at: https://ethereum.org/whitepaper/
16. ConsenSys. (2021). Beginner's Guide to Smart Contracts.

Ці джерела надали важливу теоретичну та практичну інформацію про технологію блокчейн, смарт-контракти та використані програмні засоби. Вони були використані для отримання поглибленого розуміння теми та підтримки аргументації у дипломній роботі.

5 ДОДАТОК 1

DragonEye.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import
"@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol";
import
"@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
import "@openzeppelin/contracts/security/Pausable.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/Counters.sol";

contract DragonEye is ERC721, ERC721Enumerable, ERC721URIStorage, Pausable,
Ownable {
    using Counters for Counters.Counter;
     using Strings for uint256;

    Counters.Counter private _tokenIdCounter;

    string baseURI;
    string public baseExtension = ".json";
    uint256 public maxSupply = 100;
    uint256 public maxMintAmount = 3;

    constructor(string memory _initBaseURI, string memory _name, string memory
_symbol) ERC721(_name,_symbol) {
        setBaseURI(_initBaseURI);
    }

    function _baseURI() internal view virtual override returns (string memory) {
        return baseURI;
    }

    function pause() public onlyOwner {
        _pause();
    }

    function unpause() public onlyOwner {
```

```
        _unpause();
    }

    function mint(uint256 _mintAmount) public {
        uint256 supply = totalSupply();
        require(_mintAmount > 0);
        require(_mintAmount <= maxMintAmount);
        require(supply + _mintAmount <= maxSupply);

        for (uint256 i = 1; i <= _mintAmount; i++) {
        _safeMint(msg.sender, supply + i);
        }
    }

    function _beforeTokenTransfer(address from, address to, uint256 tokenId, uint256
batchSize)
        internal
        whenNotPaused
        override(ERC721, ERC721Enumerable)
    {
        super._beforeTokenTransfer(from, to, tokenId, batchSize);
    }

    function _burn(uint256 tokenId) internal override(ERC721, ERC721URIStorage) {
        super._burn(tokenId);
    }

    function tokenURI(uint256 tokenId)
        public
        view
        override(ERC721, ERC721URIStorage)
        returns (string memory)
    {
        string memory currentBaseURI = _baseURI();
        return bytes(currentBaseURI).length > 0
        ? string(abi.encodePacked(currentBaseURI, tokenId.toString(), baseExtension))
        : "";
    }

    function setmaxMintAmount(uint256 _newmaxMintAmount) public onlyOwner {
        maxMintAmount = _newmaxMintAmount;
    }
```

```
    function setBaseURI(string memory _newBaseURI) public onlyOwner {
        baseURI = _newBaseURI;
    }

    function setBaseExtension(string memory _newBaseExtension) public onlyOwner
{
        baseExtension = _newBaseExtension;
    }

    function supportsInterface(bytes4 interfaceId)
        public
        view
        override(ERC721, ERC721Enumerable)
        returns (bool)
    {
        return super.supportsInterface(interfaceId);
    }
}
```

Migrations.sol

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

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

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

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

1_initial_migration.js

```
const Migrations = artifacts.require("Migrations");

module.exports = function (deployer) {
  deployer.deploy(Migrations);
};
```

2_deploy_contracts.js

```
const DE = artifacts.require("DragonEye");
const fs = require('fs');
const link = fs.readFileSync(".link").toString().trim();

module.exports = function (deployer) {
  deployer.deploy(DE, link, "DragonEye", "DE");
};
```

Deploy.js

```
const { spawn } = require("child_process");

const run = async () => {
  console.log("📄 Deploying and updating contracts...");
  try {
    spawn(
      "truffle migrate --network goerli --reset",
      {
        shell: true,
        stdio: "inherit",
      }
    );
  } catch (e) {
    console.log(e);
  }
};
run();
```

index.html

```
<!DOCTYPE html>
<html>
<head>
	<meta charset="utf-8">
	<title>Minting Page</title>
	<link rel="stylesheet" href="style.css">
	<script
src="https://cdnjs.cloudflare.com/ajax/libs/web3/1.7.4-rc.1/web3.min.js"></script>
</head>
<body bgcolor="black">
	<div class="wave"></div>
	<script type="module" src="web3.js"></script>
	<div class="image-wrap">
		<img src="14.png" width="220px">
		<p>How much do you want to mint?</p>
		<input type="text" id="num" size="28" value="1">
		<div class="container">
		  <div class="btn" id="mint">
		    <div class="shop-now">Mint now</div>
		    <div class="snowflake-grid to-left">
```

```
                    <div class="snowflake-item border-bottom border-right">
                      <div class="sub-items border-right border-bottom pull-down">
                        <div class="m-w-15 m-h-15 border-right border-bottom
m-3"></div>
                      </div>
                    </div>
                    <div class="snowflake-item border-bottom border-left">
                      <div
                        class="sub-items border-right border-bottom r-90
pull-down-right"
                        >
                        <div class="m-w-15 m-h-15 border-right border-bottom
m-3"></div>
                      </div>
                    </div>
                    <div class="snowflake-item border-top border-right">
                      <div class="sub-items border-right border-bottom r-270
pull-right">
                        <div class="m-w-15 m-h-15 border-right border-bottom
m-3"></div>
                      </div>
                    </div>
                    <div class="snowflake-item border-top border-left">
                      <div class="sub-items border-right border-bottom r-180
pull-left">
                        <div class="m-w-15 m-h-15 border-right border-bottom
m-3"></div>
                      </div>
                    </div>
                  </div>
                  <div class="snowflake-grid to-right">
                    <div class="snowflake-item border-bottom border-right">
                      <div class="sub-items border-right border-bottom pull-down">
                        <div class="m-w-15 m-h-15 border-right border-bottom
m-3"></div>
                      </div>
                    </div>
                    <div class="snowflake-item border-bottom border-left">
                      <div
                        class="sub-items border-right border-bottom r-90
pull-down-right"
                        >
```

```
                        <div class="m-w-15 m-h-15 border-right border-bottom
m-3"></div>
                      </div>
                    </div>
                    <div class="snowflake-item border-top border-right">
                      <div class="sub-items border-right border-bottom r-270
pull-right">
                        <div class="m-w-15 m-h-15 border-right border-bottom
m-3"></div>
                      </div>
                    </div>
                    <div class="snowflake-item border-top border-left">
                      <div class="sub-items border-right border-bottom r-180
pull-left">
                        <div class="m-w-15 m-h-15 border-right border-bottom
m-3"></div>
                      </div>
                    </div>
                  </div>
                </div>
              </div>
        </div>

        <button class="button" id="wallet">
  <span class="button__text">

<span>C</span><span>o</span>n</span><span>n</span><span>e</span><span>c
</span><span>t</span><span>
</span><span>w</span><span>a</span><span>l</span><span>l</span><span>e</s
pan><span>t</span>
  </span>
  <svg class="button__svg" role="presentational" viewBox="0 0 600 600">
    <defs>
      <clipPath id="myClip">
        <rect x="0" y="0" width="100%" height="50%" />
      </clipPath>
    </defs>
    <g clip-path="url(#myClip)">
      <g id="money">
        <path
d="M441.9,116.54h-162c-4.66,0-8.49,4.34-8.62,9.83l.85,278.17,178.37,2V126.37C4
50.38,120.89,446.56,116.52,441.9,116.54Z" fill="#699e64" stroke="#323c44"
stroke-miterlimit="10" stroke-width="14" />
```

```
      <path
d="M424.73,165.49c-10-2.53-17.38-12-17.68-24H316.44c-.09,11.58-7,21.53-16.62,2
3.94-3.24.92-5.54,4.29-5.62,8.21V376.54H430.1V173.71C430.15,169.83,427.93,166
.43,424.73,165.49Z" fill="#699e64" stroke="#323c44" stroke-miterlimit="10"
stroke-width="14" />
    </g>
    <g id="creditcard">
      <path
d="M372.12,181.59H210.9c-4.64,0-8.45,4.34-8.58,9.83l.85,278.17,177.49,2V191.42
C380.55,185.94,376.75,181.57,372.12,181.59Z" fill="#a76fe2" stroke="#323c44"
stroke-miterlimit="10" stroke-width="14" />
      <path
d="M347.55,261.85H332.22c-3.73,0-6.76-3.58-6.76-8v-35.2c0-4.42,3-8,6.76-8h15.3
3c3.73,0,6.76,3.58,6.76,8v35.2C354.31,258.27,351.28,261.85,347.55,261.85Z"
fill="#ffdc67" />
      <path d="M249.73,183.76h28.85v274.8H249.73Z" fill="#323c44" />
    </g>
  </g>
  <g id="wallet">
    <path
d="M478,288.23h-337A28.93,28.93,0,0,0,112,317.14V546.2a29,29,0,0,0,28.94,28.9
5H478a29,29,0,0,0,28.95-28.94h0v-229A29,29,0,0,0,478,288.23Z" fill="#a4bdc1"
stroke="#323c44" stroke-miterlimit="10" stroke-width="14" />
    <path
d="M512.83,382.71H416.71a28.93,28.93,0,0,0-28.95,28.94h0V467.8a29,29,0,0,0,28
.95,28.95h96.12a19.31,19.31,0,0,0,19.3-19.3V402a19.3,19.3,0,0,0-19.3-19.3Z"
fill="#a4bdc1" stroke="#323c44" stroke-miterlimit="10" stroke-width="14" />
    <path
d="M451.46,435.79v7.88a14.48,14.48,0,1,1-29,0v-7.9a14.48,14.48,0,0,1,29,0Z"
fill="#a4bdc1" stroke="#323c44" stroke-miterlimit="10" stroke-width="14" />
    <path
d="M147.87,541.93V320.84c-.05-13.2,8.25-21.51,21.62-24.27a42.71,42.71,0,0,1,7.1
4-1.32l-29.36-.63a67.77,67.77,0,0,0-9.13.45c-13.37,2.75-20.32,12.57-20.27,25.77l.3
8,221.24c-1.57,15.44,8.15,27.08,25.34,26.1l33-.19c-15.9,0-28.78-10.58-28.76-25.93
Z" fill="#7b8f91" />
    <path
d="M148.16,343.22a6,6,0,0,0-6,6v92a6,6,0,0,0,12,0v-92A6,6,0,0,0,148.16,343.22Z"
fill="#323c44" />
  </g>

 </svg>
</button>
```

```
<div id="popup" class="popup">
        <div class="popup__body">
                <div class="popup__content">
                        <a href="" class="popup__close">X</a>
                        <div class="popup__title"><b id='title'>Ошибка</b></div>
                        <div class="popup__text"><p id='text'></p></div>
                </div>
        </div>
</div>
</body>
</html>
```

Style.css

```
.image-wrap{
        background: black;
        color: white;
        border-radius: 1em;
        padding: 1em;
        position: absolute;
        top: 50%;
        left: 50%;
        margin-right: -50%;
        transform: translate(-50%, -50%)
}
/* only animate if the device supports hover */
@media (hover: hover) {
  #creditcard {
    /*  set start position */
    transform: translateY(110px);
    transition: transform 0.1s ease-in-out;
    /*  set transition for mouse enter & exit */
  }

  #money {
    /*  set start position */
    transform: translateY(180px);
    transition: transform 0.1s ease-in-out;
    /*  set transition for mouse enter & exit */
  }

  button:hover #creditcard {
    transform: translateY(0px);
```

```
    transition: transform 0.2s ease-in-out;
    /*  overide transition for mouse enter */
  }

  button:hover #money {
    transform: translateY(0px);
    transition: transform 0.3s ease-in-out;
    /*  overide transition for mouse enter */
  }
}

@keyframes bounce {
  0% {
    transform: translateY(0);
  }
  50% {
    transform: translateY(-0.25rem);
  }
  100% {
    transform: translateY(0);
  }
}

.button:hover .button__text span {
  transform: translateY(-0.25rem);
  transition: transform .2s ease-in-out;
}

/* styling */

@import
url("https://fonts.googleapis.com/css2?family=Lato:wght@300;400&display=swap");

body {
  height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
}

.button {
  border: none;
  outline: none;
```

```
  background-color: purple;
  padding: 1rem 90px 1rem 2rem;
  position: absolute;
  top: 15px;
  right:15px;
  border-radius: 8px;
  letter-spacing: 0.7px;
  background-color: #5086bd;
  color: #fff;
  font-size: 21px;
  font-family: "Lato", sans-serif;
  cursor: pointer;
  box-shadow: rgba(0, 9, 61, 0.2) 0px 4px 8px 0px;
}

.button:active {
  transform: translateY(1px);
}

.button__svg {
  position: absolute;
  overflow: visible;
  bottom: 6px;
  right: 0.2rem;
  height: 140%;
}

:root {
  --background-color: #000000;
  --color: #000000;
  --light-theme: #ffffff;
}


body {
  font-family: "Mulish", sans-serif;
  background-color: var(--background-color);
}

.container {
  display: flex;
  justify-content: center;
  align-items: center;
```

```
  height: 30vh;
}

.btn {
  display: flex;
  justify-content: center;
  align-items: center;

  width: 200px;
  height: 60px;

  background-color: var(--light-theme);
  cursor: pointer;
  font-size: 24px;
  color: var(--color);
  transition: all 0.3s;
  position: relative;

  text-align: center;
  overflow: hidden;

  border-radius: 5px;
  box-shadow: 0 6px 30px -10px rgba(#CCCCCC, 1);

  &:hover {
    transform: translateX(5px) translateY(-7px);
  }
}

.shop-now {
  position: relative;
  .santa-icon {
    position: absolute;
    width: 30px;
    height: 30px;
    top: -17px;
    right: -20px;
    transform: rotate(14deg);
    img {
      width: 100%;
      height: 100%;
    }
```

```
  }
}

.snowflake-grid {
  display: inline-grid;
  grid-template-columns: 1fr 1fr;
  grid-auto-rows: 50px;
  gap: 5px;

  width: 100px;
  transform: scale(0.4);
  div {
    border-radius: 5px;
  }

  .snowflake-item {
    position: relative;
  }
}

.to-left {
  position: absolute;
  top: -4px;
  left: -35px;
  animation: swing 3s infinite;
  div {
    animation: flash 3s infinite;
  }
}

.to-right {
  position: absolute;
  top: -25px;
  right: -35px;
  animation: swing 2.5s infinite;
  div {
    animation: flash 2.5s infinite;
  }
}

.border-left {
  border-left: 4px solid #e6dada;
```

```
}

.border-right {
  border-right: 4px solid #e6dada;
}

.border-bottom {
  border-bottom: 4px solid #e6dada;
}

.border-top {
  border-top: 4px solid #e6dada;
}

.sub-items {
  height: 28px;
  width: 28px;
}

.m-w-15 {
  width: 15px;
}

.m-h-15 {
  height: 15px;
}

.r-270 {
  transform: rotate(270deg);
}

.r-180 {
  transform: rotate(180deg);
}

.r-90 {
  transform: rotate(90deg);
}

.pull-down {
  position: absolute;
  bottom: 5px;
  right: 5px;
```

```
}

.pull-down-right {
  position: absolute;
  bottom: 5px;
  left: 5px;
}

.pull-right {
  position: absolute;
  right: 5px;
  top: 5px;
}

.pull-left {
  position: absolute;
  left: 5px;
  top: 5px;
}

.m-3 {
  margin: 3px;
}

@keyframes swing {
  50% {
    transform: rotateZ(10deg) scale(0.4);
  }
}

@keyframes flash {
  50% {
    border-color: #485563;
  }
}

body {
    margin: auto;
    font-family: -apple-system, BlinkMacSystemFont, sans-serif;
    overflow: auto;
    background: linear-gradient(315deg, rgba(101,0,94,1) 3%, rgba(60,132,206,1)
38%, rgba(48,238,226,1) 68%, rgba(255,25,25,1) 98%);
    animation: gradient 15s ease infinite;
```

```
    background-size: 400% 400%;
    background-attachment: fixed;
}

@keyframes gradient {
    0% {
        background-position: 0% 0%;
    }
    50% {
        background-position: 100% 100%;
    }
    100% {
        background-position: 0% 0%;
    }
}

.wave {
    background: rgb(255 255 255 / 25%);
    border-radius: 1000% 1000% 0 0;
    position: fixed;
    width: 200%;
    height: 12em;
    animation: wave 10s -3s linear infinite;
    transform: translate3d(0, 0, 0);
    opacity: 0.8;
    bottom: 0;
    left: 0;
    z-index: -1;
}

.wave:nth-of-type(2) {
    bottom: -1.25em;
    animation: wave 18s linear reverse infinite;
    opacity: 0.8;
}

.wave:nth-of-type(3) {
    bottom: -2.5em;
    animation: wave 20s -1s reverse infinite;
    opacity: 0.9;
}

@keyframes wave {
```

```
    2% {
        transform: translateX(1);
    }

    25% {
        transform: translateX(-25%);
    }

    50% {
        transform: translateX(-50%);
    }

    75% {
        transform: translateX(-25%);
    }

    100% {
        transform: translateX(1);
    }
}

.popup{
  position: fixed;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.8);
  top: 0;
  left: 0;
  opacity: 0;
  visibility: hidden;
}

.popup__body{
  min-height: 100%;
  display: flex;
  align-items: center;
  justify-content: center;
  padding: 30px 10px;
}

.popup__content{
  background-color: #fff;
  color: #000;
```

```
  max-width: 800px;
  padding: 30px;
  position: relative;
}

.popup__close{
  position: absolute;
  right: 10px;
  top: 10px;
  font-size: 20px;
  color: #000;
  text-decoration: none;

}

.popup__title{
  font-size: 40px;
  margin: 0px 0px 1em 0px;

}

.popup__text{

}
```

Web3.js

```
import myContract from "./json/DragonEye.json" assert { type: "json" };


window.onload=function(){
  var btn = document.getElementById("wallet");
  var mintBtn = document.getElementById("mint");
  btn.addEventListener("click", connect);
  mintBtn.addEventListener("click", mint);
  init();
}
```

```
let walletAddress;
let contract;

async function connect() {
  if (window.ethereum) {
     await window.ethereum.request({ method: "eth_requestAccounts" });
     window.web3 = new Web3(window.ethereum);
     const account = web3.eth.accounts;
     //Get the current MetaMask selected/active wallet
     walletAddress = account.givenProvider.selectedAddress;
     console.log(`Wallet: ${walletAddress}`);

  } else {
    document.getElementById('title').innerHTML = "Something went wrong with your wallet";
       document.getElementById('text').innerHTML = "It seems like you dont have cryptowallet";
    console.log("No wallet");
    document.getElementById('popup').style.visibility = "visible";
    document.getElementById('popup').style.opacity = "1";
  }
}

const init = async () => {
  const web3 = new Web3(window.ethereum);
  const id = await web3.eth.net.getId();
                contract = new web3.eth.Contract(myContract.abi, myContract.networks[id].address);
```

```
}

async function mint(){
   if(walletAddress == undefined){
     await connect();
   }
   const number = document.getElementById('num').value;
   try{
      await contract.methods.mint(number).send({from: walletAddress})
          .then(result=>{
           console.log(result);
                document.getElementById('title').innerHTML = "Congratulations! Your token:";
           document.getElementById('text').innerHTML = "";
           for (var i = 0; i < number; i++) {
              let id = result.events.Transfer[i].returnValues.tokenId;
              console.log(id);
                                  document.getElementById('text').innerHTML += "<img src='https://bafybeigyqifpczmfidzy27lwmnako6bv3nz2eo4znkv52tr53ochrbguzu.ipfs.nftstorage.link/"+id+".png' width='220px align='center'>";
           }
           document.getElementById('popup').style.visibility = "visible";
           document.getElementById('popup').style.opacity = "1";
          })
   }catch(error){
     document.getElementById('title').innerHTML = "Something went wrong!";
        document.getElementById('text').innerHTML = "It seems like something was corrupted";
```

```
      document.getElementById('popup').style.visibility = "visible";
      document.getElementById('popup').style.opacity = "1";
      console.log(error);
    }
}

const totalSupply = async () =>{
  if(walletAddress == undefined){
      await connect();
    }
    return await contract.methods.totalSupply().call({from: walletAddress});
}
```