НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і

спеціалізованих комп'ютерних систем

«На правах рукопису»
УДК 004.4

До захисту допущено:
Завідувач кафедри
Віталій РОМАНКЕВИЧ
«___»_____2024 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Системне програмування та спеціалізовані комп'ютерні системи»

зі спеціальності 123 «Комп'ютерна інженерія»

на тему: «Децентралізований краудфандинг-протокол для фінансових транзакцій на основі технології блокчейн»

Виконав:
студент ІІ курсу, групи КВ-21 мп
Далечин Владислав Олександрович _____

Науковий керівник:
к.т.н., доц.,
Петрашенко Андрій Васильович _____

Рецензент:
_____
_____ _____

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

Київ – 2024 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність – 123 «Комп'ютерна інженерія»»

Освітньо-професійна програма «**Системне програмування та спеціалізовані комп'ютерні системи»**

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ

«___»_____2024 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Далечину Владиславу Олександровичу

1. Тема дисертації «Децентралізований краудфандинг-протокол для фінансових транзакцій на основі технології блокчейн», науковий керівник дисертації Петрашенко Андрій Васильович, к.т.н, доц., затверджені наказом по університету від «9» листопада 2023 р. №5217-с

2. Термін подання студентом дисертації 10 січня 2024

3. Об'єкт дослідження: методи та засоби розробки децентралізованого краудфандинг-протоколу для фінансових транзакцій

4. Вихідні дані

5. Перелік завдань, які потрібно розробити: проаналізувати існуючі рішення, визначити поточні перешкоди краудфандинг протоколів, визначити інструментарій для роботи над протоколом, визначити блокчейн на якому протокол оперуватиме, розробити протокол децентралізованого краудфандингу

6. Орієнтовний перелік графічного (ілюстративного) матеріалу - презентація

7. Орієнтовний перелік публікацій

7.1. Прикладна математика та комп'ютинг. «XVI науково-практична конференція магістрантів та аспірантів ПМК-2023 факультету прикладної математики».

7.2. CXXXVI Міжнародна науково-практична конференція «Зимові наукові читання – 2023».

9. Дата видачі завдання 1 листопада 2022р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Вивчення літератури за тематикою дисертації.	10.09.2022	
2.	Підготовка матеріалів першого розділу магістерської дисертації.	21.11.2022	
3.	Підготовка матеріалів другого розділу магістерської дисертації	14.02.2023	
4.	Підготовка матеріалів третього розділу магістерської дисертації	19.04.2023	
5.	Підготовка матеріалів четвертого розділу магістерської дисертації	05.05.2023	
6.	Розробка протоколу	19.07.2023	
7.	Тестування протоколу	07.09.2023	
8.	Оформлення документації магістерської дисертації	10.10.2023	
9.	Попередній розгляд магістерської дисертації на кафедрі	21.12.2023	

Студент Владислав ДАЛЕЧИН

Науковий керівник Андрій ПЕТРАШЕНКО

РЕФЕРАТ

Актуальність теми. На сьогодні краудфандинг є популярним способом залучення фінансування для різноманітних проектів. Проте існуючі централізовані платформи краудфандингу мають низку недоліків, таких як високі комісії, ризики шахрайства, відсутність прозорості та ін. Використання технології блокчейн дозволяє створити децентралізований краудфандинг-протокол, який усуває ці недоліки. Тому розробка такого протоколу є актуальною задачею.

Об'єктом дослідження є процес децентралізованого краудфандингу на основі технології блокчейн

Предметом дослідження є методи та засоби розробки децентралізованого краудфандинг-протоколу для фінансових транзакцій.

Метою роботи є розробка та реалізація децентралізованого краудфандинг-протоколу, який на основі технології блокчейн забезпечує безпеку, прозорість, приватність та ефективність фінансових транзакцій. Дослідження спрямоване на налагодження внутрішніх механізмів протоколу, забезпечення масштабованості та швидкості операцій, розробку алгоритмів консенсусу та захисту від шахрайства, а також дослідження можливостей для інтеграції з існуючими блокчейн-платформами.

Наукова новизна полягає в методиці ітеративної витрати зібраних фінансів для забезпечення повернення коштів користувачами, яка відрізняється від методик децентралізованого керування коштами типу DAO завдяки відсутності вектору атак типу Sybil, що дозволяє впровадити ізольований метод керування вкладеннями окремого користувача.

Практична цінність отриманих в роботі результатів полягає в тому, що реалізована в протоколі методика ітеративної витрати зібраних коштів дозволяє забезпечити можливість повернення вкладених користувачами коштів у разі припинення або згортання проекту. Поступова витрата коштів протягом

краудфандингової кампанії підвищує довіру учасників, оскільки їх внески не витрачаються одразу повністю.

Апробація роботи. Запропонований підхід був представлений та обговорений на науковій конференції магістрантів та аспірантів «Прикладна математика та комп‘ютинг» ПМК-2023 (Київ, 28 - 30 листопада 20223 р.) та на «CXXXVI Міжнародній науково-практична конференції: Зимові наукові читання – 2023», яка відбулася 22 грудня 2023 р. у Києві, Україна.

Структура та обсяг роботи. Магістерська дисертація складається з вступу, трьох розділів та висновків.

У *вступі* подано загальну характеристику роботи, обґрунтовано актуальність теми дослідження, сформульовано мету, завдання, об'єкт, предмет та методи дослідження, розкрито наукову новизну та практичне значення отриманих результатів.

У *першому розділі* проведено огляд літератури за темою дослідження, проаналізовано існуючі підходи до побудови децентралізованих систем краудфандингу.

У *другому розділі* сформульовано вимоги до децентралізованого краудфандинг-протоколу, запропоновано його архітектуру та алгоритми функціонування.

У *третьому розділі* описується вибір інструментів реалізації та реалізація прототипу протоколу.

У *четвертому розділі* наведено результати експериментальних досліджень та оцінку ефективності розробленого протоколу.

У *висновках* викладено основні результати дисертаційної роботи.

Робота представлена на 133 аркушах, містить посилання на список використаних літературних джерел.

Ключові слова: смарт-контракти, децентралізований краудфандинг, блокчейн, атаки sybil, DAO.

ABSTRACT

Actuality of theme. Today, crowdfunding is a popular way of obtaining funding for various projects. However, traditional centralized crowdfunding platforms may have some disadvantages, such as high commissions, risk of cheating, lack of visibility, etc. The use of blockchain technology makes it possible to create a decentralized crowdfunding protocol, which is limited. Therefore, the development of such a protocol is an urgent task.

The object of research is the process of decentralized crowdfunding based on blockchain technology.

The subject of research are methods and methods for developing a decentralized crowdfunding protocol for financial transactions.

The goal of the work: development and implementation of a decentralized crowdfunding protocol, which, based on blockchain technology, will ensure security, transparency, privacy and efficiency of financial transactions. The research is aimed at improving the internal mechanisms of the protocol, ensuring the scale and speed of operations, developing consensus algorithms and protecting against fraud, as well as exploring the possibilities for integration with existing blockchain systems.

The scientific novelty consists in the method of iterative spending of the collected finances to ensure the return of funds by users, which differs from the methods of decentralized management of DAO-type funds due to the absence of a Sybil-type attack vector, which allows implementing an isolated method of managing investments of an individual user.

The practical value obtained in the work results lies in the fact that the method of iterative spending of collected funds, implemented in the protocol, makes it possible to ensure the possibility of reversing the contributions of investors' funds once the project is started or started. The incremental contribution of funds during the crowdfunding campaign promotes the trust of participants, and the remainder of their contributions are not immediately spent.

Approbation of work. The proposed approach was presented and discussed at the scientific conference of master's and postgraduate students "Applied mathematics and computing" PMK-2023 (Kyiv, November 28 - 30, 2023) and at the IX International scientific and technical Internet conference "CXXXVI International Scientific and Practical Conference: Winter Scientific Readings – 2023", which was held on December 22, 2023 in Kyiv, Ukraine.

Structure and scope of work. The master's thesis consists of an introduction, three chapters and conclusions.

The introduction presents the fundamental characteristics of the work, the relevance of the research is outlined, the meta, research, object, subject and research methods are formulated, the scientific novelty and practical significance of the results are revealed..

In *the first chapter* the literature on the research topic is carried out, and other approaches to implementing decentralized crowdfunding systems are analyzed.

The second chapter provides a formulation to decentralized crowdfunding protocol, its architecture and functioning algorithms.

In *the third section*, the selection of the implementation tooling and the implementation of the prototype protocol is described,.

In *the fourth section*, the results of experimental research and evaluation of the effectiveness of the developed protocol are presented.

The results of the work are presented in *the conclusions*.

The work is presented on 133 sheets, and contains links to the list of used literary sources.

Keywords: smart contracts, decentralized crowdfunding, blockchain, sybil attacks, DAO.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

- QF (Quadratic Funding) – квадратичне фінансування, метод розподілу громадських коштів, який збільшує вплив невеликих донорів.
- ZK Proof (Zero-Knowledge Proof) – доказ з нульовим розкриттям інформації, криптографічний метод, який дозволяє одній стороні довести іншій, що вони знають певну інформацію, не розкриваючи самої інформації.
- Gas (Газ) – одиниця виміру, яка використовується для вимірювання кількості виконавчого ресурсу, необхідного для проведення транзакцій або виконання смарт-контрактів у мережі Ethereum.
- DAO (Decentralized Autonomous Organization) – децентралізована автономна організація, що керується смарт-контрактами на блокчейні, зазвичай без централізованого контролю.
- Sybil Attack – атака на мережу, де одна особа створює багато підроблених ідентичностей для набуття непропорційно великого впливу.
- ERC20 – стандарт токена для Ethereum, який визначає загальний список правил, якими повинні керуватися токени Ethereum.
- ERC721 – стандарт для незамінних токенів (NFTs) на Ethereum, який дозволяє відстежувати та обмінювати унікальні цифрові активи.
- KYC (Know Your Customer) – процес верифікації ідентичності клієнтів, що є стандартною практикою у фінансовій індустрії для запобігання шахрайства.
- AML (Anti-Money Laundering) – набір процедур, законів та регуляцій, розроблених для запобігання легалізації коштів, отриманих незаконним шляхом.

1 АНАЛІЗ СУЧАСНИХ ДОСЯГНЕНЬ В ДОСЛІДЖЕННІ ДЕЦЕНТРАЛІЗОВАНИХ КРАУДФАНДИНГ-ПРОТОКОЛІВ

1.1 Сутність децентралізованого краудфандингу

Краудфандинг, як феномен сучасного фінансування, є відносно новим, але стрімко розвивається. Визначається він як процес збору коштів від великої кількості людей, зазвичай через Інтернет, для підтримки різноманітних проектів - від творчих ініціатив до стартапів та соціальних кампаній. Історія краудфандингу бере свій початок від традиційних форм колективного фінансування, але отримала значний розвиток із появою Інтернету, що дозволило збирати кошти на глобальному рівні з небаченою до цього масштабністю та швидкістю.

Еволюція краудфандингу тісно пов'язана із розвитком цифрових технологій. У ранніх 2000-х роках, з появою перших платформ краудфандингу, таких як Kickstarter і Indiegogo, ця модель фінансування стала доступною широкому колу людей. Такі платформи надали змогу не тільки збирати кошти на реалізацію ідей, але й створити спільноти підтримки навколо проектів, залучаючи увагу та ресурси від людей, які розділяють спільні інтереси.

Проте, традиційний краудфандинг має свої обмеження. Платформи, що сприяють цьому процесу, часто вимагають значних комісійних виплат, а також можуть ставити під сумнів прозорість та безпеку транзакцій. Це стимулювало пошук нових підходів та технологій, здатних оптимізувати існуючу систему краудфандингу. Однією із відповідей на ці виклики стала концепція децентралізованого краудфандингу, яка базується на використанні блокчейн технології. Ця новаторська модель пропонує альтернативу традиційним методам, видаляючи посередників та забезпечуючи більшу прозорість, нижчі витрати та підвищену безпеку фінансових транзакцій. Вона також розширює можливості краудфандингу, дозволяючи кожному учаснику безпосередньо взаємодіяти з проектом, що підвищує довіру та залученість.

Краудфандинг продовжує розвиватися, адаптуючись до змін у технологічному та фінансовому ландшафті. Від традиційних моделей до сучасних

децентралізованих систем, цей метод фінансування відкриває нові горизонти для інновацій, творчості та соціального впливу. Ключовим аспектом в еволюції краудфандингу є його трансформація з простого збору коштів у потужний інструмент соціального впливу. Сучасні проекти краудфандингу часто виходять за межі одномірного фінансування, включаючи в себе елементи спільноти, співпраці та соціальної відповідальності. Це відображає зміну парадигми у сприйнятті та використанні краудфандингу як інструменту, що впливає не тільки на комерційний успіх, але й на соціальні та культурні ініціативи.

Наразі краудфандинг виступає не лише як механізм фінансування, але й як платформа для тестування ідей, залучення зворотного зв'язку від споживачів та формування лояльних спільнот навколо продуктів та послуг. Це відкриває нові горизонти для підприємців та творців, які тепер можуть спілкуватися зі своїми підтримувачами безпосередньо, отримуючи цінні інсайти та покращуючи свої проекти до їх впровадження на ринок.

Децентралізований краудфандинг використовує технологію блокчейн для збору коштів для проектів або підприємств. Він включає в себе безліч учасників, які вносять вклади в обмін на майбутній продукт, послугу або долю в компанії. Основна перевага децентралізованого краудфандингу полягає в тому, що він видаляє посередників і дозволяє взаємодіяти безпосередньо між учасниками. Також, Технологія блокчейн дозволяє забезпечити прозорість та безпеку транзакцій. Кожна транзакція записується в блокчейн, що забезпечує відслідковуваність та неможливість зміни або видалення запису. Це створює довіру між учасниками та дозволяє забезпечити справедливість процесу.

Децентралізований краудфандинг також демократизує процес фінансування. У традиційних моделях краудфандингу, таких як Kickstarter або Indiegogo, вкладники мають обмежений вплив на процес використання зібраних коштів. Вони можуть вирішити, чи хочуть вони підтримати проект, але після збору коштів вони не мають контролю над тим, як ці кошти використовуються. У випадку з децентралізованим краудфандингом, вкладники мають можливість впливати на

процес використання коштів через механізми голосування, що забезпечуються технологією блокчейн.

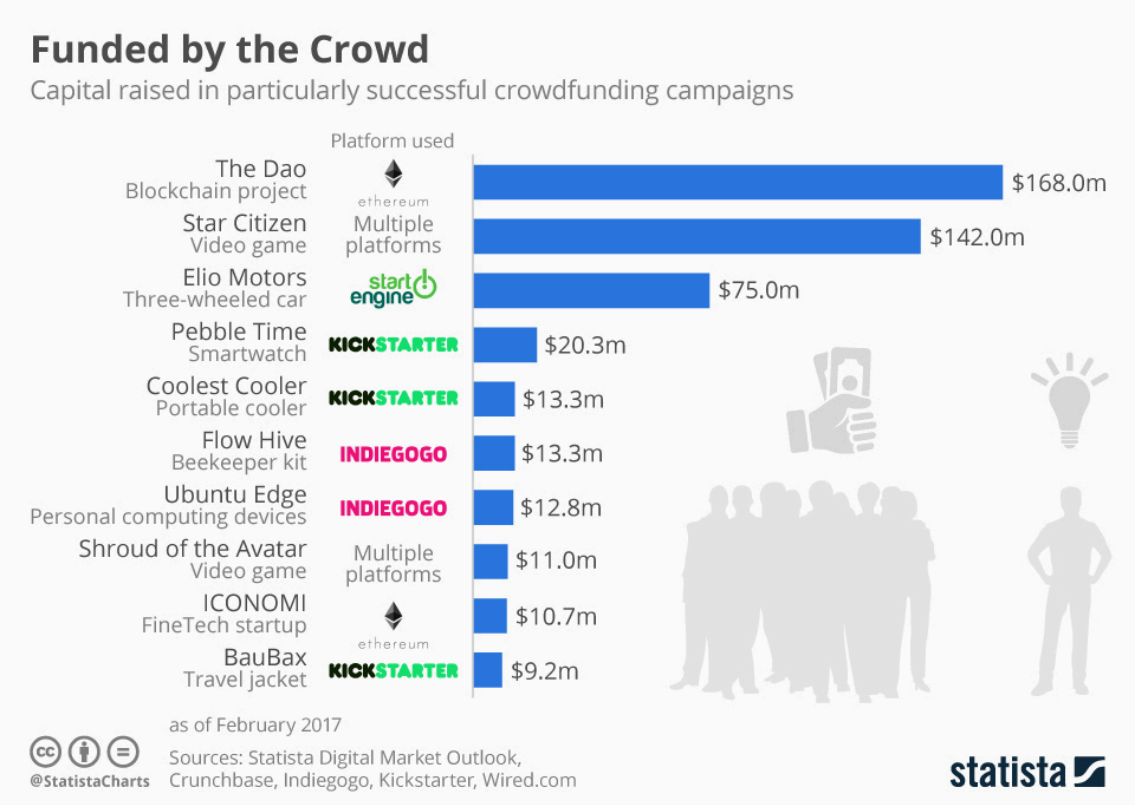


Рисунок 1. Статистика найбільших фінансових залучень до проектів методом краудфандингу[1].

Однак, децентралізований краудфандинг також має свої виклики та обмеження. Зокрема, він вимагає від учасників певного рівня технічних знань та розуміння технології блокчейн. Крім того, через відсутність регулювання існує ризик шахрайства та інших видів зловживань.

Незважаючи на ці виклики, децентралізований краудфандинг продовжує набирати популярність. За останні кілька років з'явилося багато платформ, що пропонують децентралізовані рішення для краудфандингу. Ці платформи використовують різні блокчейн-протоколи, включаючи Ethereum, Binance Smart Chain, Polkadot та інші. Вони використовують смарт-контракти для автоматизації процесу збору коштів та розподілу винагород між учасниками.

[1] https://www.statista.com/chart/8253/crowdfunding-campaigns/

Одним з ключових аспектів децентралізованого краудфандингу є його відкритість та інклюзивність. Будь-хто з доступом до інтернету може взяти участь у децентралізованому краудфандинговому проекті, незалежно від свого місця проживання або соціально-економічного статусу. Це відкриває можливості для людей у регіонах з обмеженими можливостями фінансування, дозволяючи їм залучати кошти на міжнародному рівні. Крім того, децентралізований краудфандинг дає можливість вкладникам більше контролювати використання своїх коштів. Вони можуть голосувати за або проти певних рішень, що стосуються використання коштів, а також відстежувати, як використовуються їх вклади. Це створює більше прозорості та відповідальності, ніж традиційні моделі краудфандингу.

Відмінність децентралізованого краудфандингу полягає також у тому, що він дозволяє зібрати кошти не тільки в традиційних валютах, але й у криптовалютах. Це може бути особливо корисно для проектів, які пов'язані з криптовалютами або блокчейном, а також для учасників, які воліють використовувати криптовалюту замість традиційної валюти. Незважаючи на свою відносну новизну, децентралізований краудфандинг вже має ряд успішних прикладів. Наприклад, проект ConstitutionDAO зібрав більше 47 мільйонів долларів США через децентралізований краудфандинг[2].

У цілому, децентралізований краудфандинг представляє собою цікаву та інноваційну альтернативу традиційним моделям фінансування. Він демократизує доступ до фінансування, забезпечує прозорість та дає можливість вкладникам більше контролювати використання своїх коштів. Однак, як і будь-яка нова технологія, він також має власні виклики та обмеження. Одним з найбільших викликів для децентралізованого краудфандингу є його складність та технічна вимогливість. Для участі в децентралізованому краудфандинговому проекті потрібно мати певні технічні знання та розуміння технології блокчейн. Це може

---

[2] https://www.theverge.com/22820563/constitution-meme-47-million-crypto-crowdfunding-blockchain-ethereum-constitution

бути бар'єром для людей, які не мають технічного фону або новачків в криптовалюті.

Крім того, через відсутність регулювання існує ризик шахрайства та інших видів зловживань. Хоча технологія блокчейн може забезпечити прозорість та відслідковуваність транзакцій, вона не може запобігти всім видам шахрайства. Наприклад, організатори проекту можуть просто зібрати кошти та зникнути, не реалізувавши обіцяний проект.

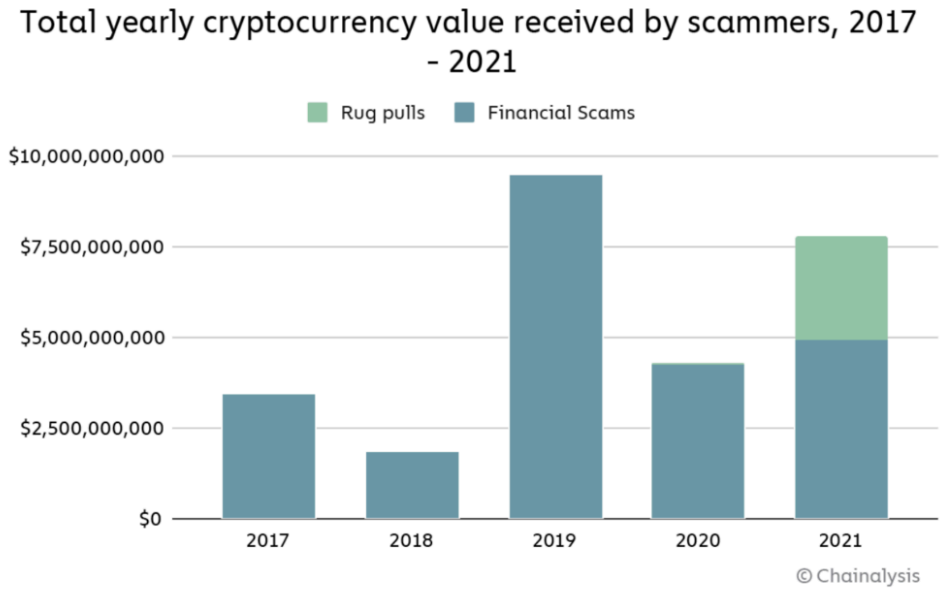


Рисунок 2. Вкрадені кошти в криптовалюті з 2017-2021 роки.

Щоб подолати ці виклики, потрібно подальше дослідження та розробка. Наприклад, можна розробити більш прості та інтуїтивно зрозумілі інтерфейси для децентралізованих краудфандингових платформ, щоб зробити їх більш доступними для широкої аудиторії. Також можна розробити механізми для забезпечення відповідальності організаторів проектів та захисту вкладників від шахрайства. Децентралізація в контексті краудфандингу відкриває нові можливості та виклики. Вона є ключовою характеристикою блокчейн-технологій, що лежать в основі багатьох сучасних платформ краудфандингу. Децентралізація передбачає передачу контролю та влади з централізованих органів або інститутів до децентралізованої мережі. В контексті краудфандингу, це означає, що

управління проектами, збір коштів та розподіл прибутку можуть відбуватися безпосередньо між учасниками - інвесторами та підприємцями - без необхідності в посередниках, як-от традиційні банки або фінансові інститути.

Децентралізація в краудфандингу має ряд переваг. По-перше, вона знижує вартість угод за рахунок виключення посередників, що істотно скорочує комісії. По-друге, вона підвищує прозорість, оскільки всі транзакції реєструються в блокчейні, що забезпечує високий рівень відстежуваності та аудиту. Це, в свою чергу, зменшує ризики шахрайства та збільшує довіру між учасниками.

Крім того, децентралізація сприяє демократизації фінансування. Вона дозволяє залучати кошти від великої кількості невеликих інвесторів, що робить краудфандинг доступнішим для широкого кола підприємців та стартапів, які можуть не мати доступу до традиційних форм фінансування. Також це відкриває можливості для інвесторів із різних частин світу вкладати кошти у проекти, що їх цікавлять, незалежно від географічного розташування. Однак, децентралізація також несе в собі виклики. Одним з основних є потреба в розробці надійних механізмів управління та консенсусу в рамках блокчейну для забезпечення справедливого та ефективного функціонування платформ. Іншим викликом є регуляторна невизначеність, оскільки багато держав ще не розробили чітких правил для регулювання децентралізованих фінансових транзакцій.

Децентралізація відіграє ключову роль у трансформації традиційних моделей краудфандингу, вносячи зміни, які впливають на всі аспекти цього процесу – від залучення фінансування до розподілу прибутку. Основним двигуном цих змін є блокчейн технологія, що дозволяє реалізувати децентралізовані фінансові операції. В результаті цього краудфандинг стає більш гнучким, прозорим та доступним, що сприяє виникненню нових форм фінансування. Однією з найбільших переваг децентралізації є зниження залежності від посередників. Традиційні краудфандингові платформи, як правило, функціонують як посередники між інвесторами та ініціаторами проектів, встановлюючи правила та вимоги, а також збираючи комісійні за свої послуги. Децентралізація дозволяє уникнути цих бар'єрів, надаючи можливість безпосередньої взаємодії між

сторонами, що знижує витрати та підвищує ефективність.

Далі, децентралізація сприяє підвищенню прозорості фінансових операцій. Всі транзакції, які відбуваються в рамках децентралізованого краудфандингу, можуть бути записані в блокчейн, що гарантує їх незмінність та доступність для перевірки. Це забезпечує високий рівень довіри серед інвесторів та проектантів, оскільки кожен може відслідковувати, як використовуються кошти, та перевіряти результати проектів.

Децентралізація також сприяє демократизації краудфандингу. Вона відкриває доступ до фінансування ширшому колу людей, включаючи тих, хто раніше був відсічений від традиційних фінансових ресурсів. Це особливо важливо для малих проектів та стартапів, які часто стикаються з викликами у залученні капіталу через традиційні канали. Крім того, децентралізований краудфандинг дозволяє реалізувати нові моделі фінансування, такі як токенізація. Інвестори можуть отримувати токени, які представляють їх частку в проекті або надають певні права та переваги. Токени можуть бути використані для різних цілей, включаючи голосування по проекту, доступ до продуктів чи послуг та участь у прибутку.

Загалом, роль децентралізації у зміні краудфандингових моделей є багатогранною та перспективною. Вона не тільки сприяє ефективності та прозорості, але й відкриває нові можливості для інноваційного фінансування, зміцнюючи зв'язок між інвесторами та проектантами та розширюючи горизонти для креативних і бізнес-ідей.

1.2 Сучасні здобутки в області децентралізованого краудфандингу

В області децентралізованого краудфандингу за останні роки спостерігається значний прогрес, особливо з приходом інноваційних блокчейн-технологій. Важливу роль в цьому процесі відіграє розробка смарт-контрактів, які дозволяють автоматизувати і оптимізувати збір коштів і розподіл винагород. Ethereum, як платформа для створення децентралізованих

додатків, відкрила нові горизонти у цій сфері, надаючи інструменти для ефективного використання краудфандингу.

Початкові смарт-контракти були відносно простими і зосередженими на базових функціях, таких як збір коштів та автоматичне розподілення винагород. Останніми роками розробники значно розширили їх можливості, інтегруючи складні логічні операції та алгоритми, які дозволяють виконувати більш складні фінансові угоди. Безпека є одним з найважливіших аспектів в смарт-контрактах. Останні розробки зосереджені на підвищенні їх надійності через удосконалення коду та використання передових методів криптографії. Це дозволяє забезпечити, що фонди зберігаються безпечно та використовуються згідно з умовами краудфандингової кампанії.

Інновації у сфері смарт-контрактів відіграють важливу роль у розвитку децентралізованого краудфандингу. Смарт-контракти – це програмні алгоритми, які виконують угоди автоматично, коли визначені умови задовольняються, забезпечуючи ефективність та прозорість угод. Початкові смарт-контракти були відносно простими, але останнім часом вони стали більш складними та функціональними. Це дозволяє їм здійснювати більш комплексні фінансові операції, що є ключовим для залучення та розподілу коштів у краудфандингових проектах. Таке розвиток забезпечує нові можливості для підприємців та інвесторів.

З плином часу смарт-контракти стають більш модульними та адаптивними, дозволяючи користувачам налаштовувати їх під конкретні потреби проектів. Ця гнучкість сприяє індивідуалізації краудфандингових кампаній, забезпечуючи краще врахування унікальних вимог та цілей кожного проекту. Модульність також означає, що смарт-контракти можуть легко інтегруватися з іншими технологіями та платформами, розширюючи їх функціональність. Це дозволяє краудфандинговим платформам використовувати складніші механізми для забезпечення більшої прозорості та ефективності. Такі інновації відкривають двері до нових можливостей у фінансуванні проектів.

Безпека смарт-контрактів є одним з основних пріоритетів розробників,

оскільки це вирішально важливо для забезпечення довіри інвесторів. Останні розробки включають в себе застосування передових методів криптографії та покращення алгоритмів для забезпечення надійності та безпеки смарт-контрактів. Це включає в себе ретельне тестування та верифікацію коду, щоб запобігти помилкам та уразливостям. Забезпечення безпеки смарт-контрактів є ключовим для збереження коштів інвесторів та гарантії виконання угод згідно з встановленими умовами. Це створює міцну основу для довіри та надійності в краудфандингових ініціативах.

Автоматизація, яку пропонують смарт-контракти, істотно змінює спосіб ведення краудфандингових кампаній. Автоматизація не тільки знижує витрати на управління та обслуговування, але й забезпечує швидке виконання угод. Це особливо важливо для залучення та розподілу коштів у реальному часі, зменшуючи затримки та паперову роботу. Крім того, автоматизація допомагає уникнути людських помилок, що можуть виникати при ручній обробці транзакцій. Такий підхід дозволяє забезпечити вищу прозорість та ефективність в краудфандингових проектах, забезпечуючи більш справедливий та демократичний процес.

Інтеграція смарт-контрактів з іншими технологіями, такими як штучний інтелект (ШІ) та машинне навчання (МН), відкриває нові можливості для інновацій у краудфандингу. ШІ та МН можуть допомогти аналізувати тенденції ринку, оцінювати ризики та прогнозувати потенціал успіху різних проектів. Це дозволяє краудфандинговим платформам надавати більш точні рекомендації та покращувати стратегії залучення коштів. Інтеграція з ШІ та МН також може сприяти розвитку більш інтуїтивно зрозумілих інтерфейсів, зробивши краудфандинг доступнішим для ширшого кола користувачів. Ці інновації відіграють важливу роль у створенні більш інтерактивного та залучаючого досвіду краудфандингу.

На сьогоднішній день з'явилася велика кількість платформ, які використовують ці технології для залучення коштів. Вони пропонують користувачам не лише можливість фінансувати проекти, але й активно брати

участь у їх розвитку і управлінні. Наприклад, платформи Gitcoin, DAOstack і Aragon демонструють, як можна використовувати блокчейн для створення спільнот, що самостійно управляють фінансуванням проектів.

Ці платформи надають учасникам значно більше контролю над процесом краудфандингу, дозволяючи їм не тільки вибирати проекти для інвестицій, але й впливати на хід їх реалізації. Прозорість і відкритість, які забезпечуються блокчейн-технологією, змінюють традиційні підходи до фінансування, створюючи більш ефективну та демократичну систему. Незважаючи на існуючі виклики, децентралізований краудфандинг вже демонструє свій потенціал як ефективний інструмент залучення коштів. Він відкриває нові можливості для підприємців та інноваторів, які можуть залучати кошти безпосередньо від спільноти, минаючи традиційні фінансові установи.

Існують певні обмеження та виклики, які необхідно враховувати. Основні з них - це технічні складності та потреба у глибокому розумінні блокчейн-технологій з боку учасників. Це може створити бар'єри для входу, особливо для тих, хто не має достатнього технічного фону. Також, відсутність регулювання у цій сфері може призвести до ризиків шахрайства та зловживань, оскільки традиційні механізми захисту інвесторів тут часто не діють.

Ще одним важливим аспектом є питання масштабування блокчейн-систем, які використовуються для децентралізованого краудфандингу. Оскільки багато блокчейнів, включаючи Ethereum, мають обмеження щодо швидкості транзакцій та пропускної спроможності, це може створити проблеми при залученні великих обсягів коштів або при управлінні великою кількістю мікротранзакцій.

Для подолання цих обмежень, спільнота розробників працює над рішеннями, такими як вдосконалення алгоритмів консенсусу, оптимізація мережевих протоколів та розробка другого рівня рішень для масштабування. Також важливою є розробка більш зрозумілих і доступних інтерфейсів для користувачів, що знизить вхідний поріг і спростить використання децентралізованих краудфандингових платформ.

Ще одним кроком у напрямку забезпечення безпеки та довіри в децентралізованому краудфандингу є створення регулятивних рамок, які зможуть забезпечити захист прав інвесторів та запобігти шахрайству. Це вимагає співпраці регуляторів, розробників та спільноти з метою створення збалансованих та ефективних правил.

Для розуміння сучасних здобутків в краудфандингу, важливо проаналізувати платформи які працюють наразі:

Таблиця 1. Опис найпопулярніших краудфандингових проектів та статистика за долученими інвестиціями.

Проект	Короткий опис	Наукова новизна	Долучили інвестицій на сумму
CLRFund	Протокол квадратичного фінансування і окремі краудфандинг програми	Перший QF протокол на ринку, використання MACI ZK-proofs	$900,000
Giveth	Краудфандинг мультитул	Найбагатша по функціоналу краудфандинг платформа	$1,100,000
Protocol Guild	Коллектив з 128 основних розробників Ethereum Core	Простота	$5,000,000
Mirror	Платформа для опублікування статей і краудфангдингу	Поєднання контенту з краудфандингом	$80,000,000

Juicebox	Краудфандинг платформа	Легке формування капіталу для DAO	$100,000,000
Gitcoin	Протокол квадратичного фінансування і вирішення Sybil проблеми	Перший QF інструмент на ринку.	$72,000,000

- CLRFund є децентралізованою платформою для квадратичного фінансування, яка фокусується на забезпеченні публічних товарів у екосистемі Ethereum. Ця платформа використовує унікальний механізм визначення внесків, який збільшує вплив невеликих донорів та сприяє широкій участі спільноти. Її інноваційний підхід до фінансування робить процес більш демократичним та ефективним, залучаючи більшу кількість учасників до підтримки важливих ініціатив.
- Giveth – це платформа, що впроваджує ідею "Blockchain for Good", надаючи інструменти для безпосередньої благодійності та прозорого краудфандингу. З її допомогою користувачі можуть вільно створювати кампанії для залучення коштів на соціальні проекти та ініціативи, що змінюють світ. Giveth відрізняється високою ступенем залученості та участі спільноти, а також забезпечує прозорий звіт про використання коштів.
- Protocol Guild – це ініціатива, спрямована на об'єднання розробників Ethereum для створення стійкої фінансової системи, яка забезпечує рівні умови та винагороду за внесок у протоколи і проекти. Через документацію та взаємопідтримку розробників, Protocol Guild прагне створити солідарність у спільноті розробників та відданість до спільного успіху проектів на базі Ethereum.
- Mirror є публікаційною платформою та інструментом краудфандингу, що використовує блокчейн для авторів та творців контенту. Вона дозволяє вести блоги, збирати кошти та створювати власні токени. Mirror використовує децентралізовану ідентичність та власні токени для забезпечення авторських прав та винагороди творців, пропонуючи новий підхід до публікацій та монетизації

контенту.

- Juicebox – це протокол краудфандингу, призначений для фінансування проектів та спільнот, який надає гнучкість у створенні та управлінні фінансовими потоками. Цей протокол дозволяє користувачам налаштовувати параметри фінансування, встановлювати правила для залучення коштів та розподілу винагороди. Juicebox забезпечує прозорість та адаптивність для проектів, що шукають підтримки в широкій спільноті.
- Gitcoin продовжує бути однією з провідних платформ у сфері децентралізованого краудфандингу, зосереджуючись на фінансуванні відкритого програмного забезпечення. Завдяки використанню квадратичного фінансування та Ethereum, Gitcoin створює умови для забезпечення справедливого та ефективного розподілу коштів серед проектів, які отримують найширшу підтримку від спільноти.

Кожна з цих платформ внесла значний вклад у розвиток децентралізованого краудфандингу, продемонструвавши практичні приклади використання блокчейн-технологій. Вони демонструють, як інновації можуть змінювати традиційні підходи до залучення коштів та управління проектами, відкриваючи нові можливості для підприємців та інвесторів. Ці платформи також показують важливість спільноти та колективного управління в децентралізованих проектах, наголошуючи на значенні прозорості, довіри та співпраці.

Загалом, кейс-стадії цих платформ відображають еволюцію та різноманітність можливостей у сфері децентралізованого краудфандингу. Вони ілюструють, як децентралізація може впливати на розвиток фінансових технологій, створюючи нові шляхи для інновацій та взаємодії між інвесторами та підприємцями. Ці платформи служать важливим джерелом натхнення та зразком для майбутніх розробок у галузі децентралізованого краудфандингу.

1.3 Розгляд обмежень сучасних методів та потенційних засобів їх подолання

Основним питанням обмеження сучасних методів є масштабування блокчейн платформ. Це стосується здатності системи обробляти велику кількість транзакцій одночасно. На даний момент, найпопулярніші блокчейни, такі як Bitcoin та Ethereum, стикаються з обмеженнями у зв'язку з низькою пропускною спроможністю та довгим часом потвердження транзакцій, що може стати серйозною перешкодою для масового використання краудфандингу.

Вартість транзакцій в блокчейні, також відома як "газ", являє собою іншу перепону. Високі комісії можуть робити невигідними невеликі вклади, які є основою краудфандингу. Для стартапів та ініціатив з низьким бюджетом, високі витрати на проведення транзакцій можуть значно підняти планку входу на ринок та зменшити їх шанси на залучення необхідних коштів.

Суворі обмеження в смарт-контрактах можуть обмежувати функціональність і гнучкість краудфандингових кампаній. Наприклад, вони повинні бути написані таким чином, щоб автоматично обробляти всі можливі стану і варіанти розвитку подій в кампанії. Це вимагає від розробників глибокого розуміння потреб проекту та передбачення всіх потенційних ризиків. Крім того, безпека смарт-контрактів залишається ключовим питанням. Вади в коді можуть бути використані шахраями для крадіжки коштів або завадити правильному виконанню контракту. Історія знає багато випадків, коли вразливості в смарт-контрактах призводили до великих фінансових втрат, що вказує на необхідність ретельного аудиту та тестування коду перед його розгортанням.

Проблема стабілізації валюти в децентралізованих системах також не може бути недооцінена. Волатильність криптовалют може суттєво впливати на зібрані в рамках краудфандингу кошти, оскільки вартість криптовалюти може коливатися в широкому діапазоні. Це ставить під загрозу як фінансову стабільність проекту, так і довіру інвесторів. Стейблкойни та інші механізми стабілізації, такі як хеджування, можуть бути використані для зниження цього ризику, але їх застосування може бути обмеженим через юридичні та регуляторні фактори.

Розвиток інфраструктури блокчейну з новими ланцюгами та протоколами шкаливання пропонує потенційні рішення для обмежень поточних систем. Другорядні ланцюги, децентралізовані обмінні протоколи та технології шардингу можуть забезпечити шляхи для підвищення пропускної спроможності та скорочення витрат, однак, інтеграція цих інновацій з існуючими децентралізованими краудфандинговими платформами вимагає складної роботи та оптимізації.

Залучення і ретенція талановитих розробників є вирішальним фактором у впровадженні технічних вдосконалень децентралізованих краудфандингових платформ. Створення більш надійних та безпечних смарт-контрактів, так само як і підвищення усвідомленості про кращі практики розробки, можуть вигодовувати виправлення системних недоліків та захисту від атак. Мінімізація ризиків для інвесторів є ще одним аспектом, що вимагає уваги. Передбачення механізмів для управління коштами і контролю над фінансовим потоком в процесі краудфандингу, наприклад, моделі "ітеративного фінансування", де гроші випускаються по етапах залежно від досягнення вказаних цілей, може сприяти підвищенню впевненості інвесторів. Інновації в блокчейні розвиваються стрімко, тому в цьому секторі особливо актуальна потреба у гнучкості та швидкості пристосування до нових реалій. Сучасні краудфандингові платформи повинні бути здатні інтегрувати останні досягнення в області консенсус-алгоритмів, децентралізованих фінансових механізмів (DeFi) та засобів кібербезпеки.

Розвиток інтерфейсів користувача та покращення використання децентралізованих краудфандингових платформ є численним викликом. Інтуїтивно зрозумілі і прозорі інтерфейси сприяють залученню більш широкого кола інвесторів, які можуть не мати глибоких технічних знань про блокчейн, але мають бажання вкладати кошти у перспективні проекти.

Комунікаційна прозорість і ясність управлінської моделі краудфандингових проектів є ключовим фактором для підтримання довіри учасників. Це також стосується того, як дані про хід кампанії та використання коштів будуть фіксуватися і відображатися в блокчейні, щоб забезпечити прозорість та єдність

інформації для всіх зацікавлених сторін.

Централізовані системи краудфандингу традиційно мають рівень посередницьких послуг, що може бути трудно замінити в повністю децентралізованих системах без втрати важливої функціональності, наприклад, надання експертної оцінки проектів та рішення конфліктних ситуацій. Відкритий доступ до блокчейн-даних дає можливість для створення аналітичних та прогностичних інструментів для краудфандингу, які можуть використовувати історичні дані для оцінки ризиків та потенціалу інвестицій. Такі інструменти можуть ефективно аналізувати успішність попередніх кампаній та надавати інвесторам цінну інформацію для прийняття рішень. Однак, з цими можливостями приходить і необхідність захисту приватних даних та гарантій, що аналітична інформація не буде використана шкідливим чином або для маніпуляцій ринком.

Впровадження механізмів штучного інтелекту та машинного навчання може сприяти автоматизації та оптимізації багатьох процесів у сфері децентралізованого краудфандингу. Це може включати розвиток самовчитися смарт-контрактів, які зможуть адаптувати свою поведінку до змінилися умов ринку чи поведінки інвесторів. Однак, це також посилює потребу в ретельному аудиті та перевірці таких технологій з точки зору безпеки. Проблеми приватності та анонімності у децентралізованому краудфандингу суттєво відрізняються від традиційних систем, де індивідуальні дані можуть бути краще захищені. В умовах блокчейну, транзакції зазвичай є публічними, що може підвищити ризик несанкціонованого збору даних та використання їх для неетичних цілей.

Реалізація ефективного децентралізованого правового врегулювання конфліктів у сфері краудфандингу є особливо важливою. В даний час немає універсальних рішень для вирішення суперечок, які необхідні для забезпечення довіри до платформи і захисту інтересів як інвесторів, так і організаторів проектів. В цьому контексті можуть використовуватись такі інструменти, як арбітражні смарт-контракти.

Захист інтелектуальної власності у межах децентралізованого краудфандингу є значним викликом. Посилання на інтелектуальну власність

можуть бути вмонтовані в смарт-контракти, але їх реальний захист вимагає співпраці з існуючими юридичними системами та інтеграції з авторським правом. Це може створювати непередбачені правові та технічні проблеми, що потребують виважених рішень для забезпечення інтересів творців та інвесторів.

Розвиток технологій збереження конфіденційності транзакцій у блокчейні, такі як Zero-Knowledge Proofs (докази з нульовим розголошенням) чи приватні ланцюги, можуть забезпечити користувачам більший рівень анонімності. Такі технології можуть поліпшити прийнятність децентралізованого краудфандингу для ширшого кола учасників, які переймаються питаннями особистої приватності. Важливою проблемою є також ідентифікація та верифікація користувачів на децентралізованих платформах краудфандингу. З одного боку, анонімність є основною перевагою блокчейну, але з іншого – для підтримання законності та попередження шахрайства необхідні ефективні механізми KYC (Know Your Customer) та AML (Anti-Money Laundering).

Проблема Sybil в DAO (децентралізованих автономних організаціях) виникає, коли одна особа або група створює багато фальшивих ідентичностей в мережі з метою впливати на прийняття рішень або отримання невиправдано великої винагороди. Ця проблема є суттєвою для DAO, оскільки вони покладаються на концепцію децентралізованого управління та часто використовують голосування для прийняття рішень. Sybil-атаки можуть спотворювати демократичний процес у DAO, оскільки дозволяють маніпулювати голосуваннями та іншими механізмами управління. Вони підривають довіру учасників до системи, що може призвести до зниження участі та інвестицій. Щоб протистояти Sybil-атакам, DAO розробляють механізми верифікації ідентичності, такі як Proof of Humanity або використання соціальних сітей та репутаційних систем. Однак, ці методи можуть бути складними у реалізації та часто вимагають додаткових витрат на їх підтримку та управління. Іноді вони можуть створювати бар'єри для входу нових учасників, обмежуючи децентралізацію та інклюзивність DAO. Розвиток та імплементація ефективних рішень для проблеми Sybil є одним з ключових викликів для сталого розвитку децентралізованих організацій, з яким

зіштовхуються майже всі поточні краудфандинг протоколи. Gitcoin, наприклад, має 14% Sybil адрес[3], що означає що 14% користувачів обманом отримують нагороди, створюючи багато гаманців.

Наявність релевантних та надійних смарт-контрактних бібліотек та інструментів розробки може істотно вплинути на якість та безпеку краудфандингових проектів. Вводити стандартизацію та практики повторного використання коду може допомогти зменшити кількість помилок та вразливостей, пов'язаних з розробкою смарт-контрактів.

Нагальна потреба в широкому запровадженні методів формальної верифікації смарт-контрактів та інших блокчейн-компонентів сприяє створенню більш надійних і безпечних систем краудфандингу. Використання формальних методів дозволяє математично довести відповідність реалізованого коду смарт-контрактів до їх специфікацій, знижуючи потенційні ризики. Юридичні та регуляторні обмеження є однією з ключових проблем у сфері децентралізованого краудфандингу. На міжнародному рівні не існує єдиної правової доктрини, що регулює використання блокчейн технологій та криптовалют. Різні країни мають свої власні визначення та правила, що ускладнює для стартапів процес планування глобального краудфандингового проекту та приводить до юридичної невизначеності.

Іншою важливою проблемою є вважання токенів, залучених через краудфандинг, цінними паперами. Деякі установи, як-от Комісія з цінних паперів і бірж (SEC) у США, можуть розглядати токени як цінні папери. Це підпадає під регулювання, яке вимагає реєстрації та виконання вимог щодо дозволів, що значно збільшує витрати та ускладнює процедуру залучення капіталу.

Захист прав інвесторів є важливим аспектом, який законодавці намагаються гарантувати у традиційних фінансових системах. В контексті децентралізованого краудфандингу, надання інвесторам точної інформації та захист від шахрайства стають складнішими з огляду на анонімність і відсутність централізованого контролюючого органу.

---

[3] https://dune.com/queries/1939489/3199754

Введення механізмів KYC (знання свого клієнта) та AML (протидія відмиванню грошей) є обов'язковими для забезпечення транспарентності фінансових операцій в багатьох юрисдикціях. Децентралізовані платформи вимушені шукати способи інтеграції цих процесів, що часто суперечить філософії повної децентралізації та анонімності, закладеної в блокчейн. Відповідальність організаторів краудфандингових кампаній також є предметом юридичної складності. У випадку невдалих проектів або шахрайства, виявляється складним визначення, хто несе відповідальність та які кроки можуть бути вжиті проти винних. В децентралізованій системі, де смарт-контракти автоматично виконують транзакції без центрального регулюючого органу, встановлення відповідальності стає ще більш заплутаним.

Ускладнення виникають із необхідністю інтеграції юридично значимих документів, таких як угоди інвесторів, терміни та умови краудфандингових платформ і традиційні контрактні відносини в рамках блокчейн схем. Юридична валідність смарт-контрактів часто залишається невирішеною проблемою в контексті національного законодавства. Анонімність учасників блокчейну є подвійною проблемою: з одного боку, вона захищає приватність користувачів, з іншого – створює труднощі щодо виявлення особи у випадку юридичних конфліктів. Це особливо складно під час розгляду судових справ та призначення штрафних санкцій або компенсацій.

Створення юридичних рамок, які дозволяють гнучко підходити до змін в технологіях, так і до різноманітності проектів у сфері краудфандингу, залишається однією з головних задач законотворців. Перепоною тут є ризик перевантаження регулятивними вимогами, які можуть стримувати інновації та підприємницьку активність.

Важливим аспектом є розробка міжнародних стандартів та консенсусу між різними юрисдикціями для спрощення процедур крос-кордонного краудфандингу. Це могло б зменшити юридичну невизначеність для проектів, що залежать від інвестицій з різних частин світу, а також сприяти всебічному розвитку цифрової економіки. Поширення блокчейн технологій вимагає оновлення та модернізації

існуючих регуляторних режимів, що стосуються не тільки технологічних, але й економічних та соціальних аспектів. Юристи і законодавці повинні працювати над створенням гнучких, прозорих і зрозумілих правил, які б враховували непостійний характер цих технологій та їхній потенціал для інновацій у сфері фінансування.

Комплексність міжнародного права і відсутність єдиних стандартів розслідування і притягнення до відповідальності у сфері кіберзлочинів ставить під загрозу безпеку інвесторів у децентралізованому краудфандингу. Наприклад, труднощі виникають, коли необхідно розслідувати шахрайські дії, які трапляються через мережі, які охоплюють кілька правових систем. Вплив національної монетарної політики та контролю за фінансовими потоками може бути послаблений у результаті використання децентралізованих криптовалют. Урядам важливо виміряти, наскільки вони готові дозволити цифрові валюти впливати на національну економіку, збалансувавши між необхідністю стимулювання інновацій та забезпеченням фінансової стабільності.

Необхідність створення умов для взаємодії блокчейн платформ з традиційними фінансовими інститутами також є важливою регуляторною задачею. Для криптовалют і смарт-контрактів потрібно знайти місце у вже існуючих банківських та фінансових системах, враховуючи при цьому вимоги і обмеження, які вони накладають.

Висновки до розділу

У розділі, присвяченому аналізу сучасних досягнень у дослідженні децентралізованих краудфандинг-протоколів, розглядаються ключові аспекти та інновації в цій сфері. Встановлено, що блокчейн-технологія значно трансформувала підходи до краудфандингу, зокрема через прозорість і децентралізацію, які вона пропонує. В той же час, проблема довіри та імплементація правових рамок для регулювання характеру і форми токенів все ще потребують законодавчого врегулювання, зокрема, щодо оподаткування і вимог до звітності. Інтенсивне зростання загального інтересу до криптовалют і блокчейну стимулює розробку технічних рішень, що полегшують ведення бізнесу в рамках

цифрової економіки, а також покращують умови взаємодії між інвесторами і стартапами.

Під час дослідження було встановлено, що наявні технічні обмеження сучасних децентралізованих краудфандингових платформ потребують ретельного технічного аналізу та оптимізації. Витрати на виконання смарт-контрактів, а також складність розробки і супроводження їх коду, можуть суттєво впливати на ефективність краудфандингових кампаній. Було визначено, що поліпшення доступності та зниження складності смарт-контрактів через використання інструментів візуального програмування та створення єдиних шаблонів може стати потенційним рішенням цих проблем. Це, в свою чергу, може сприяти збільшенню кількості проектів, здатних залучити фінансування через децентралізовані краудфандингові платформи, забезпечуючи тим самим розвиток інновацій, підприємництва та економічного росту.

Виходячи із зібраної інформації, стає зрозуміло, що для вдалого подолання юридичних та регуляторних обмежень необхідно ефективне співпрацювання між законодавчими органами, блокчейн спільнотою та індустрією краудфандингу. Критично необхідно розробити такі законодавчі акти, які з одного боку захищатимуть інвесторів та споживачів, а з другого – не обмежуватимуть інноваційний потенціал децентралізованих краудфандинг-платформ. Визначення чітких правил приведе до зменшення правової невизначеності, створить сприятливі умови для розвитку та може привести до підвищення загальної ефективності краудфандингу за рахунок прозорості та довіри.

Окрім того, в розділі було наголошено на важливості подальших досліджень у сфері забезпечення приватності та безпеки даних у децентралізованих мережах. Обробка і зберігання персональних даних вимагає глибокого розуміння та дотримання міжнародних стандартів і норм, таких як GDPR. Розробка та інтеграція спеціалізованих рішень, що дозволяють виконувати умови конфіденційності залишається важливою умовою для досягнення більшої прийнятності краудфандинг-платформ на глобальному ринку.

2 ДОСЛІДЖЕННЯ, ВИБІР ТА ОГЛЯД ОБРАНИХ МЕТОДІВ ДЛЯ РЕАЛІЗАЦІЇ ДЕЦЕНТРАЛІЗОВАНОГО КРАУДФАНДИНГ-ПРОТОКОЛУ

2.1 Ітеративна витрата зібраних фінансів як актуальний метод

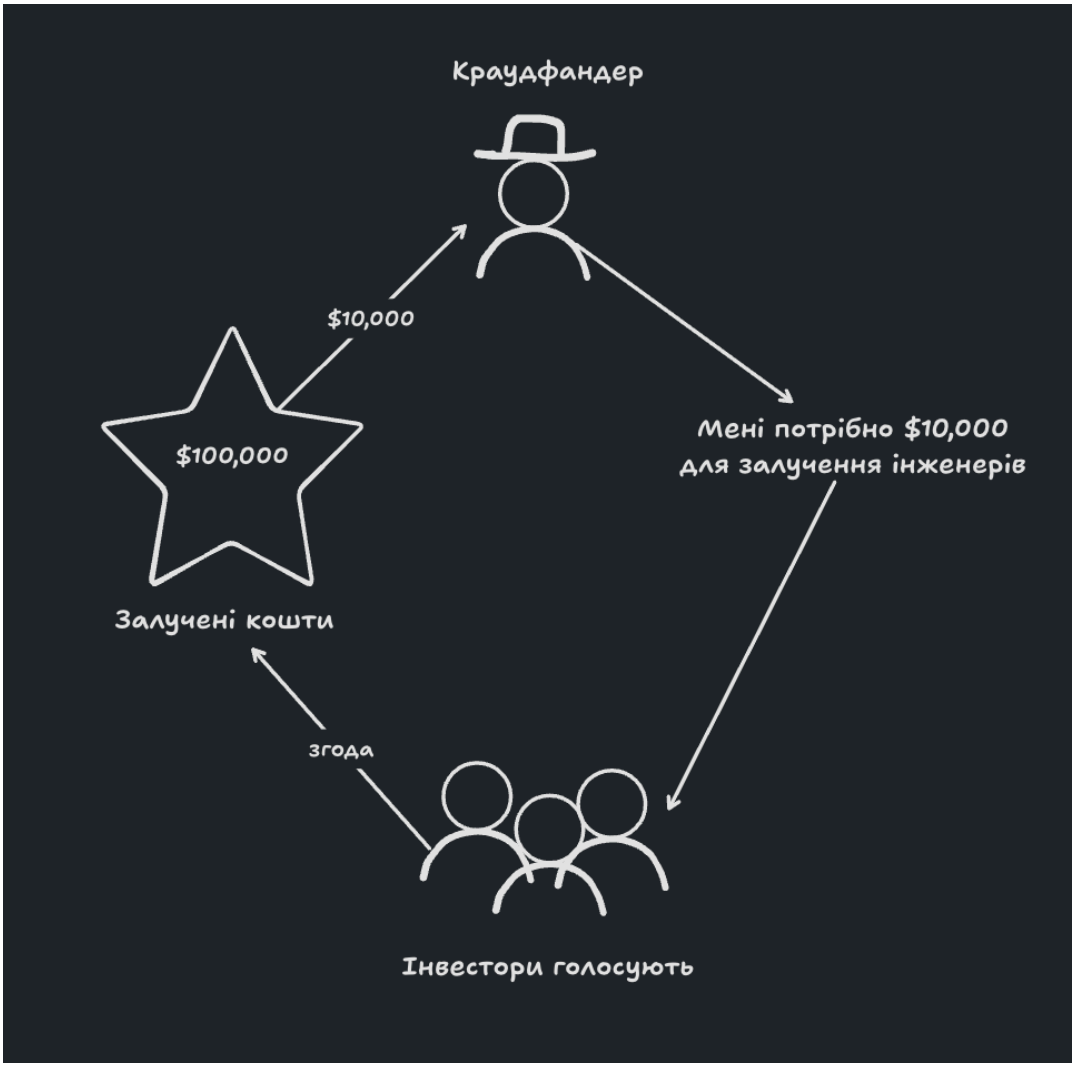


Рисунок 3. Спрощена схема ітеративної витрати коштів.

Ітеративна витрата коштів є фундаментальним принципом обережного управління капіталом у краудфандингових кампаніях. Цей підхід дозволяє організаторам проектів отримувати доступ до зібраних коштів частинами за досягненням певних цілей розвитку чи етапів, замість одноразової витрати всієї суми. Таким чином, інвестори можуть бачити прогрес в реалізації проекту, а ризик зловживань коштами мінімізується. Ретельне розподілення бюджету на етапи призводить до більш ефективного використання ресурсів, підсилює зв'язок між фінансуванням та результатами роботи, та дає можливість інвесторам відігравати

вирішальну роль в прийнятті рішень про подальше фінансування.

Захист інвестицій є однією з провідних характеристик, які інвестори шукають при виборі платформ для вкладення коштів. У децентралізованому середовищі, смарт-контракти надають умову "якщо-то", забезпечуючи автоматизоване виконання угод на основі попередньо заданих критеріїв. Це підвищує довіру, оскільки інвестори знають, що їхні кошти будуть зарезервовані для конкретних цілей і не можуть бути використані на незаплановані витрати або виведені без виконання обіцяних умов. Водночас це стимулює розробників проекту до своєчасного виконання поставлених завдань та забезпечення очікуваних результатів.

Відстеження витрати фінансів є ключовим компонентом транспарентності та ефективності в краудфандингових платформах. Завдяки блокчейну, кожна транзакція може бути зареєстрована та перевірена всіма учасниками мережі, що забезпечує відкритість фінансових потоків та запобігає несанкціонованому переміщенню коштів. Інвестори можуть використовувати спеціалізовані інструменти або децентралізовані додатки (DApps) для моніторингу стану і прогресу проекту, переконуючись, що їхні внески йдуть на заплановане використання. Ця прозорість може також допомогти у виявленні та запобіганні фінансових зловживань з боку організаторів. Впровадження ітеративної моделі в децентралізованому краудфандингу вже демонструє позитивні результати на багатьох проектах. Наприклад, платформи, які практикують випуск коштів поетапно, часто відзначаються більшою відкритістю та ефективністю проектного менеджменту. Інвестори отримують можливість оцінювати доцільність подальших інвестицій на основі детальних звітів про досягнуті результати. Водночас, використання ітеративної моделі допомагає проектам формувати стабільне співтовариство підтримки, що є особливо важливим для тривалості та сталості ініціатив.

У рамках ітеративного підходу, проекти можуть залучати не лише капітал, але й активно включати громаду в процеси прийняття рішень. Така модель стала важливим засобом децентралізованої демократії, даючи можливість місцевим

спільнотам безпосередньо впливати на розвиток проектів, що мають локальне значення. Це не тільки підвищує залученість та відповідальність громади за успіх проекту, але й стимулює економічне самоврядування та рівень суспільної участі у важливих рішеннях. Ітеративний підхід підкорюється принципам соціального підприємництва, допомагаючи проектам концентрувати зусилля на створенні соціальної вартості та спонукає до використання отриманих коштів відповідально та ефективно. Потенціал ітеративної моделі невичерпний, особливо з урахуванням швидкого розвитку цифрових технологій, що можуть забезпечити нові можливості для автоматизації і вдосконалення процесів взаємодії з інвесторами. З розширенням функціоналу смарт-контрактів можливі й новітні методи контролю за виконанням проектів, такі як автоматизоване стимулювання успішного завершення етапів. В майбутньому можна очікувати більшу інтеграцію з різними платформами та сервісами, що забезпечуватимуть більш плавний та гнучкий процес краудфандингу, а також допомагати проектам зберігати і зростати свою базу підтримки, включаючи як фінансові, так і нефінансові ресурси.

Ітеративний підхід у децентралізованому краудфандингу виходить з принципів прозорості, залучення спільноти та ретельної перевірки досягнення зазначених цілей проектом перед тим, як надати додаткове фінансування. Це відрізняється від традиційної моделі, де кошти передаються одразу в повному обсязі. Ітеративна модель дозволяє інвесторам поступово відпускати капітал, тим самим розбиваючи процес на менші етапи, що знижує загальний ризик втрати інвестованих коштів і стимулює відповідальність на стороні отримувачів фінансування.

Ітеративна модель ідеально поєднується з концепцією децентралізованих автономних організацій (DAO). У DAO, кожен етап краудфандингу може вимагати згоди спільноти через механізми голосування. Це забезпечує, що проект, який не відповідає очікуванням інвесторів, не буде отримувати подальше фінансування без чіткого виправдання й належного перегляду плану розвитку.

Приклад існуючих ітеративних підходів можна знайти у краудфандингових платформах, що використовують блокчейн, таких як Gitcoin. Тут проекти

фінансуються через серії "Грантових раундів"[4], де зібрані кошти поступово розподіляються серед успішних ініціатив на основі голосування спільноти і оцінки результатів роботи проектів.

Така ітеративна структура краудфандингу стимулює проекти до забезпечення регулярних оновлень про свої досягнення та підтримки активного діалогу з інвесторами. Це допомагає уникнути ситуацій, коли кошти витрачаються нецільово, оскільки періодичний перегляд досягнень стає умовою для отримання подальшого фінансування. З іншого боку, ітеративний підхід вносить певні виклики для організаторів проектів, які повинні не лише вести постійну роботу над розвитком своїх ініціатив, але й активно комунікувати з їхніми підтримувачами. Кожен етап проекту має бути добре задокументований і представлений спільноті, щоб забезпечити прозорість і підтвердити доцільність додаткового фінансування. Це вимагає додаткових часових вкладень та ресурсів на ринковий аналіз і створення звітів, що може бути викликом для команд із обмеженими ресурсами.

Ітеративна модель фінансування також сприяє більш гнучкому управлінню капіталом. Організатори мають можливість адаптуватися до змінних умов ринку, змінюючи пріоритетність завдань або перерозподіляючи ресурси відповідно до нових викликів та можливостей. Це, у свою чергу, може підвищувати якість виконання проектів та задоволення інтересів інвесторів. Завдяки використанню ітеративної моделі, проекти можуть знаходити ширшу підтримку. Інвестори, які спостерігають за успішним виконанням ініційованих етапів, більш схильні поновлювати свої вклади або рекомендувати проект іншим потенційним інвесторам. Це створює відчуття поступального прогресу та динамічного розвитку.

Цільовий підхід до використання коштів ітеративним методом може сприяти вищій оцінці проекту з боку інвесторів і, як наслідок, привабливості для нового капіталу. Крім того, чітке визначення фінансових цілей та етапів розвитку позитивно впливає на планування роботи в середині проекту та на зовнішню

[4] https://grants.gitcoin.co/

комунікацію.

В довгостроковій перспективі, ітеративна модель відкриває додаткові можливості для впровадження аналітичних інструментів, які можуть автоматично оцінювати досягнення проектів, сприяти розумному прийняттю рішень про фінансування та забезпечувати більш високий рівень управління ризиками. Спільноти DAO можуть вважати ітеративний краудфандинг особливо привабливим, оскільки ця модель фінансування відповідає їхньому бажанню до самоуправління та колективного прийняття рішень. Ітеративне виділення коштів вимагає від учасників DAO регулярно переоцінювати та обговорювати прогрес проектів, за які вони вклали свої кошти, що сприяє підвищенню залучення та активного участі в розвитку ком'юніті.

Створення та підтримка ітеративної витрати зібраних коштів подає чудовий приклад реалізації принципів агайл (agile) в краудфандингу. Це підхід орієнтований на гнучкість, адаптивність до змін та неперервне вдосконалення, що повністю корелює з динамікою та вимогами сучасного стартап-середовища. Проте, ітеративна модель також вступає в деякі протиріччя з ідеєю швидкого запуску та масштабування, яка є притаманною для багатьох стартапів. Проекти з великими амбіціями та великою швидкістю розробки можуть виявити, що ітеративні обмеження коштів ускладнюють швидке досягнення їх мети, що вимагає певної гнучкості та креативності у фінансовому плануванні.

Ітеративне фінансування має потенціал повністю змінити практику венчурного інвестування, надаючи інвесторам більший контроль над розподілом своїх коштів, у той же час надаючи стартапам додаткові можливості для залучення фінансування на підставі їх поточних досягнень і майбутнього потенціалу. Однак, слід визнати, що ітеративна модель може бути більш складною для розуміння середньостатистичним інвестором, ніж традиційні форми краудфандингу. Для успішного впровадження ітеративних методів необхідно забезпечити простоту та зрозумілість інтерфейсів і процесів, а також вести активну освітню роботу серед потенційних учасників платформи.

Розробники ініціатив у сфері децентралізованого краудфандингу мають

надавати достатньо інформації про стратегію і плани розвитку, щоб інвестори могли адекватно оцінити ризики і потенціал проекту. Використання дашбордів з детальною аналітикою, живими графіками прогресу та форумів для обговорення може поліпшити інформування та зробити процес більш залучаючим.

Застосування ітеративної моделі в краудфандингу з трьома або більше фазами фінансування може допомогти розподілити ризик між більшою кількістю невеликих інвестицій, ніж це робиться в традиційних моделях. Це може допомогти проекту отримати доступ до більш стабільного і передбачуваного джерела фінансування замість необхідності залучення одного великого інвестора. Хоч ітеративний підхід зменшує ризики, він також може уповільнити залучення загального обсягу необхідного фінансування, оскільки кошти випускаються тільки після досягнення попередніх цілей. Це ставить підвищені вимоги до терміну реалізації проектів і потребує детального планування кожного етапу. З іншого боку, ітеративний підхід може заохочувати до більшої креативності та інновацій у процесах розробки, оскільки команди можуть переосмислювати свої стратегії залучення коштів та розвитку продукту на кожному етапі, відповідно до зворотного зв'язку від спільноти та змін на ринку.

Використовуючи ітеративний підхід, можливо також ефективно маневрувати ресурсами, спрямовуючи їх у найбільш перспективні напрямки розвитку проекту. Це додатково забезпечує прозорість використання капіталу та дозволяє інвесторам бачити, як їхні кошти працюють на реалізацію конкретних завдань та цілей.

Суть ітеративного фінансування полягає не лише у способі збору коштів, але й у формуванні комунікації між розробниками проекту та його підтримувачами. Реалізація прозорої системи звітності є ключовим аспектом для підтримки високого рівня довіри, а також стимулює розробників на досягнення важливих рубежів у роботі над проектом. Ітеративний процес фінансування сприяє встановленню глибших зв'язків між розробниками проекту та інвесторами. Останні стають активними учасниками життя проекту, замикаючи коло від внеску коштів до участі в оцінці досягнутих результатів та прийнятті рішення щодо подальшого фінансування.

Ітеративне фінансування може бути застосоване не тільки для стартапів, але й для некомерційних та соціально орієнтованих проектів. Такі проекти можуть використовувати ітеративний підхід для залучення коштів на створення та підтримку громадських послуг, розробку освітніх ресурсів або ініціативи із впровадження екологічних технологій. Подальше розвиток та інтеграція ітеративного фінансування в сучасний фінансовий ландшафт може сприяти створенню нових видів фінансових інструментів. Це можуть бути нові форми інвестиційних цінних паперів або похідних фінансових інструментів, що базуються на ітеративному фінансуванні та успіху проектів.

Блокчейн технології прискорюють прийняття ітеративного фінансування завдяки своїй унікальній спроможності до шифрування даних і створення неізмінних записів. Ці технології спрощують управління коштами й одночасно забезпечують повну аудиторську прозорість, що є необхідним для залучення інвестицій та довіри з боку інвесторів.

Викликом для ітеративного краудфандингу є необхідність у високому рівні грамотності інвесторів в області блокчейну. Для забезпечення широкого прийняття цієї моделі потрібно впевнитись, що кожен потенційний учасник розуміє принципи та ризики, пов'язані з децентралізованим інвестуванням. Освітні кампанії та спрощення інтерфейсу користувача можуть сприяти досягненню цієї мети. Додатковою перевагою ітеративного підходу є можливість поступового накопичення статистичних даних про ефективність інвестиційних стратегій, які можуть надавати цінний інсайт для майбутніх рішень як на рівні окремих проектів, так і на рівні всієї краудфандингової платформи. Однак, ітеративний краудфандинг теж стикається з технічними викликами. Інтеграція з різноманітними блокчейнами, сумісність смарт-контрактів і захист від потенційних загроз безпеці вимагає безупинної роботи над підвищенням якості програмного коду та його аудиту.

Оскільки ітеративний підхід передбачає більш розгалужену інтеракцію зі спільнотою, це може створити стимулів для активнішої участі користувачів у процесі краудфандингу. Учасники можуть не тільки фінансово підтримувати

проекти, а й впливати на їх розвиток через фідбек та голосування, що робить кожний проект більш співтворчим. Для забезпечення напрацювань і довіри з боку інвесторів важливою стає участь в різноманітних галузевих подіях, круглих столах та конференціях. Представлення успіхів проекту та подальших планів на публічних заходах може збільшити прозорість роботи та поліпшити загальне сприйняття проекту на ринку.

2.2 Можливість використання Solidity для реалізації протоколу

Solidity є об'єктно-орієнтованою мовою програмування, призначеною для написання смарт-контрактів, які створюють правила розумних угод та автоматично виконують операції в мережах блокчейну, таких як Ethereum. Мова надає повний контроль над функціональністю смарт-контрактів, що дозволяє розробникам впроваджувати складні логічні умови й управління станами. Основне призначення Solidity полягає у створенні програм, що можуть взаємодіяти з блокчейном, з спеціалізованими функціями, які виконуються за умовою настання певних подій.

Solidity має синтаксис, що нагадує JavaScript або C++, що робить її порівняно зрозумілою для розробників, знайомих з цими мовами. Однак, важливо розуміти, що Solidity строго типізована, тобто кожна змінна або параметр функції вимагає вказівки їх типу.

Solidity використовує широкий спектр типів даних, серед яких:

- Примітивні типи, такі як bool, string, int (цілі числа зі знаком) та uint (цілі числа без знака), з різними розмірами (наприклад, uint8, uint256).
- Довільні типи, такі як address для зберігання Ethereum-адрес або bytes для довільних послідовностей байт.
- Складні типи, включно з масивами (які можуть бути фіксованими або динамічними), а також структурами (struct) та переліками (enum).
- Спеціалізовані контрактні типи для створення нових контрактів або взаємодії з існуючими.

Смарт-контракти в Solidity містять функції, які виконують певні дії. Функції

можуть бути:

- Зовнішні (external), тобто доступні для виклику з інших контрактів та транзакцій.
- Публічні (public), доступні з будь-якого місця, в тому числі в межах самого контракту.
- Внутрішні (internal), доступні лише в межах контракту та його наслідуваних контрактів.
- Приватні (private), доступні лише в контракті, де вони були визначені.

В порівнянні з іншими мовами, Solidity має ряд унікальних характеристик:

- Стан контракту у блокчейні зберігається між транзакціями, що робить можливими складні взаємодії та збереження даних.
- Смарт-контракти мають власну систему виключних режимів (revert, require, assert) для обробки помилок і відкату транзакцій.
- Gas-оптимізація залишається важливою складовою при написанні контрактів, оскільки всі операції в блокчейні мають витрати, тому код має бути ефективним для збереження ресурсів та зниження витрат.
- Solidity дозволяє використовувати спадкування та імпорт інших смарт-контрактів, дозволяючи створювати багаторівневу структуру та сприяючи повторному використанню коду.
- Мова включає особливості, направлені на роботу з блокчейном, такі як операції з криптовалютою та доступ до блокчейн-специфічної інформації (наприклад, msg.sender для ідентифікації відправника транзакції та block.timestamp для часу блоку).

Незважаючи на унікальні аспекти Solidity, існує східність з більш традиційними мовами програмування:

- Синтаксис та базові програмні парадигми Solidity нагадують JavaScript і C++, що робить її вивчення та перехід для розробників, знайомих з цими мовами, менш складним.
- Концепції об'єктно-орієнтованого програмування в Solidity, такі як класи (які тут представлені у вигляді контрактів) та опис спадкування, а

також механізми керування пам'яттю, є спільними для багатьох мов програмування.

Окрім Solidity, існують інші мови програмування смарт контрактів, але вони є менш популярними та широковживанними.

Таблиця 2. Детальне порівняння мов програмування смарт-контрактів.

Переваги	Недоліки
Solidity	
● Сумісність з Ethereum: Solidity розроблено для Ethereum, найпопулярнішого у світі блокчейна для децентралізованих програм (dApps) і смарт-контрактів. Якщо ваш проект включає Ethereum, Solidity є природним вибором. ● Велика спільнота розробників: Solidity може похвалитися великою та активною спільнотою розробників. Це означає, що для тих, хто вирішив працювати з нею, доступні великі ресурси, документація та підтримка. ● Шаблони смарт-контрактів: Ethereum має багату екосистему	● Виклики безпеці. Незважаючи на те, що Solidity досягла значних успіхів у покращенні безпеки, написання безпечного коду Solidity вимагає глибокого розуміння потенційних уразливостей, таких як атаки повторного входу та переповнення цілих чисел. ● Складність: Написання складних смарт-контрактів у Solidity може бути складним через його відносно низькорівневу природу та відсутність певних абстракцій високого рівня. ● Витрати на газ. Витрати на газ Solidity можуть бути непередбачуваними, що призведе до неочікуваних витрат

<table>
<tr><td>шаблонів смарт-контрактів, створених у Solidity, що полегшує розробку та розгортання поширених типів контрактів, таких як токени ERC-20, ERC-721 NFT і децентралізовані біржі (DEX).
● Інтеграція з інструментами Ethereum: Solidity без проблем працює з такими інструментами розробки Ethereum, як Truffle, Remix і Hardhat, спрощуючи процес розробки.</td><td>під час розгортання контрактів або виконання транзакцій у мережі Ethereum.</td></tr>
<tr><td colspan="2">Vyper</td></tr>
<tr><td>● Зручність читання: синтаксис Vyper дуже нагадує Python, що робить його легкочитабельним і доступним, особливо для розробників із досвідом роботи з Python.
● Безпека на першому місці: вибір дизайну Vyper надає пріоритет безпеці, що ускладнює написання вразливого коду.
● Ефективність газу: код Vyper, як правило, має передбачувані та</td><td>● Обмежені функції: Vyper навмисно пропускає деякі складні функції, що може бути обмеженням для розробників, яким потрібна розширена функціональність.
● Менша спільнота: незважаючи на зростання, спільнота Vyper менша за спільноту Solidity, що означає менше ресурсів і бібліотек.</td></tr>
</table>

нижчі витрати на газ порівняно з Solidity.	
Rust	
• Безпека пам’яті: система власності Rust і суворі перевірки компілятора знижують ризик уразливості, пов’язаної з пам’яттю. • Продуктивність: продуктивність Rust є винятковою, що робить його придатним для ресурсомістких блокчейн-додатків. • Зростаюча екосистема: екосистема блокчейнів Rust швидко розширюється завдяки таким проектам, як Framework Substrate від Parity. •	• Крива навчання: Rust має круту криву навчання, особливо для розробників без досвіду системного програмування. • Обмежені бібліотеки смарт-контрактів: екосистема смарт-контрактів Rust не така зріла, як Solidity.
LLL	
• Екстремальний контроль: LLL дозволяє розробникам писати високооптимізовані та ефективні розумні контракти з детальним	• Складність: LLL, як відомо, складно вивчати та використовувати навіть для досвідчених розробників.

контролем. ● Ефективність використання газу: завдяки своїй природі низького рівня LLL може призвести до контрактів з високою ефективністю використання газу.	● Відсутність абстракцій: бракує абстракцій високого рівня, що робить код більш багатослівним і складним для розуміння.
Scilla	
● Формальна перевірка. Головною перевагою Scilla є підтримка формальної перевірки, яка може призвести до високозахищених контрактів. ● Незмінні дані: Scilla забезпечує незмінність певних структур даних, зменшуючи ризик ненавмисних змін стану.	● Крива навчання: функції формальної перевірки Scilla можуть бути складними та вимагати досвіду формальних методів. ● Обмежене застосування: Scilla має обмежене застосування порівняно з такими мовами, як Solidity.

Смарт-контракти в Ethereum реалізують призначений для користувача код, який запускається в глобальному децентралізованому Blockchain середовищі, пропонуючи безпечне та надійне виконання угод без потреби в посередниках. Розвиток смарт-контрактів у Ethereum відкрив надзвичайні можливості для автоматизації цифрових операцій, зменшивши ризик людських помилок та забезпечивши іммунітет до втручань ззовні. Ці контракти створюють основу для довіри й гнучкості в цифровій економіці.

Безпека смарт-контрактів є критичною: дефекти чи уразливості в коді можуть мати серйозні фінансові наслідки, не кажучи вже про втішення довіри до технології блокчейну. Стратегії написання безпечного коду в Solidity включають

обережне використання зовнішніх викликів, управління помилками та чітке розуміння виконуваної логіки. Важливо враховувати всі можливі сценарії взаємодії контрактів, планувати варіанти відновлення після непередбачених подій і активно використовувати патерни безпеки, рекомендовані досвідченими розробниками. Аудит коду та його перевірка фахівцями з кібербезпеки є обов'язковим кроком перед впровадженням смарт-контрактів у життя. Solidity також має систему типів та мовні конструкти для захисту даних, які є важливими для запобігання помилок та зловмисних дій.

Таблиця 3. Опис найчастіших помилок в безпеці при імплементації смарт-контарктів.

Reentrancy	Атака, що відбувається, коли зовнішній контракт викликає функцію у поточному контракті, ще до завершення її виконання.
Integer Overflow/Underflow	Несанкціоноване переповнення або недоповнення змінних, коли вони досягають максимальних чи мінімальних значень.
Unchecked External Calls	Виклики зовнішніх контрактів без перевірки повернення значення, які можуть призвести до непередбачуваних поведінок.
Denial of Service (DoS)	Відмова у обслуговуванні може статися, коли контракт навмисно заблокований або зламаний, що робить його недоступним.

Gas Limit and Loops	Несподіване вичерпання газу через неконтрольовані цикли або операції, які вимагають великої кількості газу.
Timestamp Dependence	Залежність від блокчейн-таймстемпів може призвести до маніпуляції і неправильного виконання контракту.
Blockhash Manipulation	Використання blockhash як джерела випадковості може бути небезпечним, оскільки майнери можуть впливати на результат.
Visibility/Access Modifiers	Неправильне використання модифікаторів доступу (public/private/internal/external) може призвести до витоку даних або атак.
Delegatecall	Неналежне використання delegatecall може призвести до неконтрольованого поведінки, оскільки викликаний контракт може змінювати стан поточного контракту.
Phishing with tx.origin	Використання tx.origin для аутентифікації може бути піддане атакам фішингу, де атакувач може використовувати підроблені транзакції.
Hardcoded Secrets	Залишення таємниць або приватних ключів у коді контракту може призвести до їх викрадення і

	подальших зловживань.

Серед найбільш відомих безпекових вразливостей у Solidity є реентрантні атаки, проблеми під час виконання викликів на інші контракти та неправильне використання газу, що може призвести до неприємностей, таких як блокування коштів або небажані перекази. Розгляд цих вразливостей і побудова архітектури контрактів з урахуванням потенційних загроз є не лише бажаним, але й критично необхідним для забезпечення надійності та довіри.

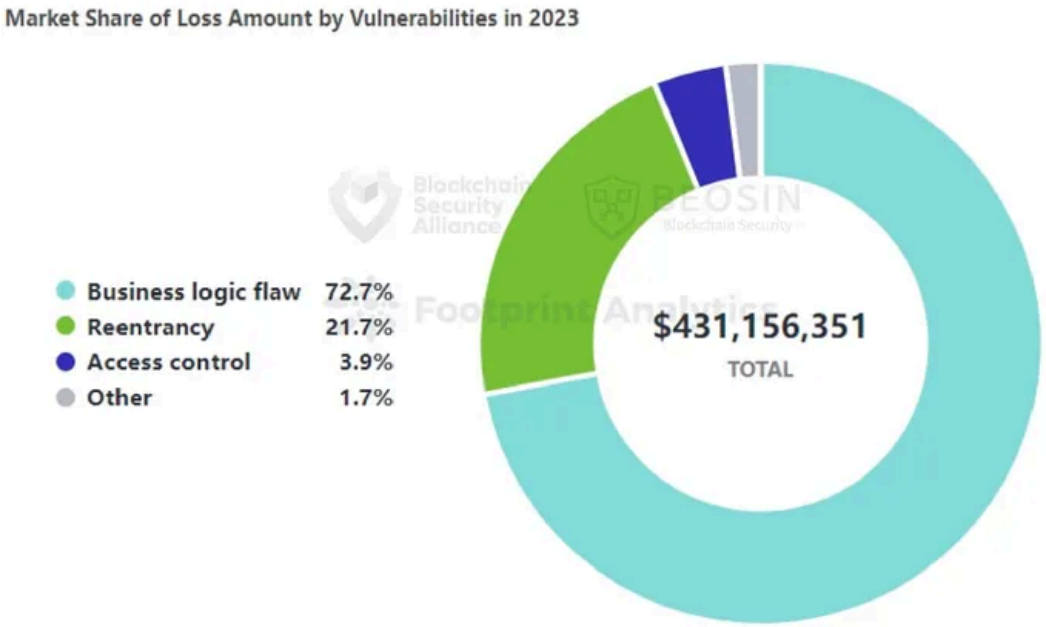


Рисунок 3. Втрачені гроші на крипто-ринку класифіковані за категорією вразливостей.

Оптимізація газу у контрактах Solidity є важливим аспектом, оскільки вартість виконання операцій на Ethereum безпосередньо пов'язана з витратами газу. Складність оптимізації полягає у досягненні балансу між скороченням витрат і збереженням необхідної функціональності та безпеки коду. Використання шаблонів проектування та стандартних бібліотек може допомогти в оптимізації та підвищенні ефективності смарт-контрактів.

Оптимізація смарт-контрактів для ефективної роботи на Ethereum може значно знизити вартість транзакцій та підвищити швидкість виконання операцій.

Використання циклів, умовних виразів та вкладених викликів функцій має бути максимально ефективним, оскільки ці конструкції потребують більше газу для обчислень. Розробники повинні ретельно підходити до вибору алгоритмів та структур даних, що використовуються у контрактах.

Переходи між версіями компілятора Solidity потребують уважного перевірення компатибільності коду та можливих впливів на витрати газу. Включення оптимізаційних прапорців компілятора та аналіз байт-коду може виявити потенційні місця для зниження витрат газу, а також допоможе забезпечити, що контракт не містить непотрібних операцій або надмірного коду.

Важливим елементом оптимізації смарт-контрактів є тестування в першу чергу тих частин коду, які найчастіше використовуються або найбільше коштують. Автоматизовані інструменти тестування можуть допомогти розробникам виявити люкси в оптимізації та знайдуть найкращий підхід до досягнення балансу між функціональністю і ефективністю витрат газу.

Ще одним ключовим аспектом у написанні продуктивних смарт-контрактів є правильне управління пам'яттю та вибір типів змінних. Типи даних з фіксованою розмірністю, як uint256 або bytes32, традиційно використовуються для економії газу порівняно із їхніми менш розмірними рівнями, такими як uint8 або bytes.

Таблиця 4. Типи даних в Solidity.

Цілі числа	int / uint	Використовуються для представлення цілих чисел. int - зі знаком, uint - без знака. Можуть мати суфікси, що вказують розмір (наприклад, uint256, int8).
Булеві	bool	Представляє логічні істиннісні значення true або false.
Адреси	address	Використовуються для зберігання Ethereum-адрес. Існує також address payable, що дозволяє відправляти ефіри.

Фіксовані числа	fixed / ufixed	Числа з фіксованою кількістю десяткових знаків. Цей тип не рекомендується до використання оскільки ще не повністю реалізований.
Числа з плаваючою комою	Не підтримується	Solidity не підтримує числа з плаваючою комою.
Байти	bytes1 ... bytes32	Незмінні послідовності байтів фіксованої довжини. bytes використовується для динамічної послідовності байтів.
Рядки	string	Використовується для представлення послідовностей символів UTF-8.
Масиви	type[]	Динамічні або фіксовані масиви, де type може бути будь-яким іншим типом даних.
Структури	struct	Використовуються для представлення складних типів даних, які містять кілька змінних.
Вибори	enum	Використовується для створення користувацьких типів, які можуть приймати одне значення з визначеного набору.
Відображення	mapping	Колекції пар ключ-значення, де ключами можуть бути лише скалярні типи (наприклад, address або uint).

Оптимальне використання шаблонів fallback та receive також сприяє

ефективності смарт-контрактів у Solidity, забезпечуючи, що кошти відправляються і отримуються контрактом найбільш оптимізованим чином. Регулярний рефакторинг і оновлення контрактів для підтримки останніх рекомендацій з безпеки і ефективності допоможе уникнути витіків коштів та зайвого витрачання газу.

Solidity, як базова мова програмування для Ethereum, пропонує потужні можливості для реалізації ітеративного краудфандингу. Смарт-контракти можуть бути ретельно налаштовані на випуск коштів у відповідності з досягненням проектом конкретних цілей, що забезпечують розумне розподілення капіталу і захист інвестицій.

Завдяки мові програмування Solidity та її інтеграції з Ethereum Virtual Machine, смарт-контракти краудфандингу можуть включати детальні правила та умови, які автономно регулюють випуск фінансування на основі виконання визначених цілей. Це дає високий рівень контролю над процесом ітеративної витрати і може істотно знизити ризики для інвесторів, оскільки кошти розблоковуються лише після підтвердження реалізації етапів розвитку проекту. Однією з ключових переваг Solidity є здатність смарт-контрактів відображати складні угоди в коді, які можуть імітувати різноманітні фінансові конструкції. Наприклад, можна створювати механізми застави, ескроу-рахунки та інші структури, які підсилюють довіру до процесу ітеративного краудфандингу та дозволяють інвесторам відчувати себе більш захищеними. Інтеграція смарт-контрактів на Solidity із зовнішніми джерелами даних через оракулів надає додаткові можливості для автоматизації ітеративного фінансування. Залучення інформації зі світу реальних подій, таких як досягнення проектом певних KPIs або певних ринкових параметрів, може бути використане як умова для випуску коштів.

Програмування індивідуальних умов договорів за допомогою Solidity відкриває перспективу для створення більш динамічних і кастомізованих моделей краудфандингу. Розробники можуть створювати унікальні варіанти угод, які точно відповідають потребам їх проектів та специфіці їх інвесторів.

Можливість інтеграції смарт-контрактів, написаних на Solidity, з іншими блоками та сервісами екосистеми Ethereum відкриває шлях для створення комплексних фінансових інструментів. Зокрема, ERC-стандарти токенів, такі як ERC-20 для фангібл токенів та ERC-721 для невзаємозамінних токенів (NFTs), можуть бути легко інтегровані в краудфандингові проекти.

Смарт-контракти на Solidity мають змогу безпосередньо комунікувати з децентралізованими обмінниками (DEXs), платформами для залучення ліквідностей (DeFi) і іншими фінансовими сервісами, застосовуючи синтетичні активи або запозичуючи кошти під заставу токенів проекту. Це дозволяє проектам з ітеративним краудфандингом створювати розширені стратегії фінансування, які забезпечують додаткову ліквідність та гнучкість у використанні капіталу.

Використання самовиконуючихся контрактів, спроектованих у Solidity, спрощує процес автоматичного розподілу доходів між учасниками проекту. Це особливо корисно для краудфандингових платформ, які виконують виплати дивідендів або інші подібні виплати, засновані на токенізованих активах.

Солідність екосистеми Solidity в Ethereum не обмежується лише смарт-контрактами, що обслуговують транзакції. Інтеграція з такими системами, як ENS (Ethereum Name Service), дозволяє забезпечити більше зручностей для користувачів, таких як людино-читабельні адреси для контрактів і транзакцій, кодифікація доменних імен в матеріали контрактів та його інтерфейси.

Для тих, хто створює краудфандингові протоколи, інтеграція з інфраструктурними інструментами, які базуються на Ethereum, такими як Truffle, Hardhat або Remix, дозволяє розробникам ефективно тестувати, розгортати і оновлювати свої смарт-контракти, спираючись на багатий набір бібліотек і шаблонів керування проектами.

Як правило, користувачі краудфандингових платформ мають обмежений технічний досвід, тому розробка інтерфейсів користувача, які взаємодіють із смарт-контрактами розробленими на Solidity, повинна бути інтуїтивно зрозумілою. Важливим є створення зрозумілих візуальних елементів та інструкцій, які дозволяють користувачам легко здійснювати інвестиції, слідкувати

за станом проектів та управління своїми активами.

Використання таких JavaScript фреймворків, як React чи Angular, разом з бібліотеками, наприклад web3.js чи ethers.js, дозволяє створювати багатофункціональні веб-інтерфейси, які спрощують процес взаємодії з блокчейном та смарт-контрактами. Ці інструменти заповнюють розрив між складними технологічними процесами та потребами кінцевих користувачів.

Успішний інтерфейс користувача для смарт-контракту повинен також підтримувати різні криптовалютні гаманці та платіжні опції, надаючи інвесторам свободу вибору зручного способу участі в краудфандинговому проекті. Він також повинен здійснювати автоматичне повернення вкладень у випадку невдачі проекту або скасування кампанії.

У розробці інтерфейсів користувача для смарт-контрактів повинна бути врахована необхідність взаємодії з різними мережами тестування і виробничим блокчейном. Це забезпечить користувачам спокій та впевненість щодо можливості експериментувати і тестувати функціональність перед внесенням реальних інвестицій. Створення інтерфейсів користувача смарт-контрактів, які є доступними і зрозумілими для широкого кола користувачів, має велике значення для успішного залучення інвестицій та широкого прийняття краудфандингових платформ. Такий інтерфейс забезпечує зв'язок між комплексною логікою смарт-контрактів і кінцевими користувачами, що дозволяє з невеликими зусиллями керувати своїми інвестиціями, відслідковувати стадії проекту та брати участь у голосуваннях.

Мова Solidity постійно розвивається, що вимагає від розробників смарт-контрактів оновлювати свої знання та уміння. Однією з головних трансформацій є прагнення до покращення безпеки та ефективності виконання контрактів, зокрема через впровадження нових версій компілятора Solidity. Розробка Solidity зустрічає виклики, пов'язані з масштабуванням та пропускною спроможністю Ethereum, адже кожна оптимізація в мові може мати значний вплив на витрати газу та швидкість транзакцій у мережі. В той же час, за останні роки було зроблено значні прогреси у вирішенні цих питань, створенні нових

інструментів дебагінгу та тестування.

Перспективи розвитку Solidity яскраво вказують на прагнення до ширшої інтеграції з іншими блокчейн платформами та шукання нових рішень для cross-chain взаємодії. Це дозволить створювати більш універсальні та адаптивні смарт-контракти, здатні функціонувати в різноманітних блокчейн екосистемах. Водночас, швидкий розвиток Solidity також створює ризик відставання в навчанні та адаптації для розробників, що вже працюють з існуючими версіями мови. Зусилля спільноти та навчальні ресурси грають ключову роль у передачі важливих знань та підтримці професійного зростання розробників. Очікується, що у майбутньому Solidity стане ще більш оптимізованою, гнучкою та потужною мовою, здатною підтримувати новітні тенденції в децентралізованому фінансуванні та контрактних механізмах, що лише розширить горизонти для інновацій та розвитку в області блокчейн технологій.

Ще один важливий аспект розвитку Solidity полягає в його здатності інтегруватися із новими протоколами консенсусу та шкаливання, такими як Proof-of-Stake та шардинг. Такі інновації в окремих блокчейн платформах збільшують швидкість транзакцій та операцій зі смарт-контрактами, що потребуватиме адаптації та возможності писати більш ефективні смарт-контракти. З огляду на постійну появу нових патернів та методик програмування, Solidity має утримувати свою позицію як мова, що забезпечує безпеку, масштабованість та гнучку логіку для краудфандингових платформ. Це передбачає на те, що спільнота розробників буде продовжувати працювати над удосконаленням інструментів та бібліотек для цілей смарт-контракт розробки.

Впровадження функцій та інструментів для забезпечення відкритості коду, таких як контракти з відкритим джерельним кодом та системи автоматичного тестування контрактів, стимулює транспарентність та довіру до мережі. Це допоможе зменшити відчуття невпевненності серед нових користувачів та забезпечити легшість вхождення в блокчейн-простір. Залишається проблемою зручність вивчення та розуміння Solidity для новачків у програмуванні смарт-контрактів. Це спонукає спільноту до створення все більше навчальних

матеріалів та курсів, які охоплюють базові принципи та розширені теми, надаючи користувачам широкий спектр знань для повноцінної роботи із смарт-контрактами. Основою успіху Solidity є активна та ресурсна спільнота розробників, яка неперервно ділиться досвідом, кодовими бібліотеками та інструментами. Онлайн форуми, освітні платформи та хакатони є цінними ресурсами для навчання та обміну знаннями, забезпечуючи розвиток Solidity та його застосування в діапазоні проектів, від простих до високо складних.

Розробка інтерфейсів користувача може включати розробку панелей адміністрування, які дозволяють організаторам краудфандингових проектів легко налаштовувати та керувати параметрами фінансування. Також важливий елемент - забезпечення гідного користувацького досвіду без прямого звернення до коду, що робить процес інвестування простішим та доступнішим для широкого кола учасників.

Solidity продовжує розвиватися разом із новими версіями Ethereum, такими як Ethereum 2.0, який прагне вирішити деякі з існуючих викликів у масштабуванні та пропускній спроможності. Це створює можливості для удосконалення існуючих контрактів та стимулювання впровадження нововведень у мові програмування. Аналіз вартості газу та ефективності контрактів постійно відіграє важливу роль у розробці контрактів Solidity. Дослідження й розуміння оптимальних шляхів впровадження логіки дозволять розробникам створювати більш економічні та швидші конфігурації смарт-контрактів для краудфандингових платформ. Розвиток децентралізованих фінансів (DeFi) бачить в Solidity ключовий елемент для побудови їх систем. Через це, розробники, що володіють глибоким розумінням Solidity, стають цінним активом в індустрії, оскільки вони можуть розробляти смарт-контракти, що впроваджують нові економічні моделі та фінансові інновації.

У майбутньому, можливо бачимо подальше поширення Solidity та смарт-контрактів в традиційні галузі, такі як нерухомість, юридичні послуги та управління цифровими ідентичностями. Роль Solidity як мови для реалізації протоколів на кшталт ітеративного краудфандингу лише зростатиме, оскільки більше секторів визнають вигоди блокчейну для безпечних та ефективних

транзакцій.

Інтеграція смарт-контрактів, написаних на Solidity, з різними компонентами Ethereum, такими як програми для залучення ліквідності та децентралізовані автономні організації (DAOs), стає зручнішою завдяки стандартизованим інтерфейсам. Таке з'єднання відкриває можливості для нових сценаріїв використання краудфандингу, зокрема гнучких моделей залучення спільнот. Впровадження інтерфейсів ERC, таких як ERC-20 для фангібл токенів або ERC-721 для невзаємозамінних токенів (NFTs), може суттєво розширити можливості краудфандингових платформ. Користувачі можуть отримувати токени як винагороду за участь у проекті, а також використовувати їх для голосування або інших форм участі в процесах управління.

Важливою перевагою Solidity є його сумісність з іншими блокчейн-протоколами та платформами через використання містків (bridges) та cross-chain технологій. Це дозволяє смарт-контрактам Solidity взаємодіяти з іншими мережами і валютами, пропонуючи розширені можливості для реалізації краудфандингових проектів. Єдність екосистеми Ethereum забезпечує легкий доступ до таких сервісів і інструментів як Infura, що надає розробникам API для взаємодії з Ethereum без необхідності власного повноцінного вузла. Це ефективно спрощує процес розробки і розгортання смарт-контрактів, знижуючи вхідний поріг для нових розробників.

Мова програмування Solidity вже інтегрована з такими ринковими механізмами, як предиктівні ринки та платформи кредитування, демонструючи багатогранність та гнучкість в організації краудфандингових кампаній. Ця інтеграція створює можливості для нових фінансових моделей та сприяє диверсифікації фінансування в екосистемі Ethereum.

2.3 Потенційне застосування протоколу на блокчейні Ethereum.

Ethereum є передовою блокчейн платформою, яка відкрила двері для широкого спектра децентралізованих застосунків, включаючи смарт-контракти та децентралізовані фінансові сервіси (DeFi). Він значно вийшов за рамки

використання як проста криптовалюта, надаючи можливість створення комплексних програмованих контрактів.

Специфіка блокчейну Ethereum полягає у його повній тюринг-вмісті, яка дозволяє смарт-контрактам виконувати майже будь-яку продуману логіку. Це створює незліченну кількість можливостей для задоволення потреб користувачів та реалізації складних систем автоматизації без втручання зовнішніх сторін.

Будучи основою для багатьох інноваційних проектів, Ethereum відіграє ключову роль у розширенні горизонтів децентралізованих застосунків. З особливим акцентом на децентралізацію й безпеку, Ethereum є природним вибором для розробників, які прагнуть створити прозорі та надійні краудфандингові протоколи.

Етеріум вводить газ як механізм виміру та обмеження ресурсів, використовуваних для виконання операцій, що забезпечує справедливість та запобігає зловживанням обчислювальними ресурсами. Вартість газу флуктує на основі завантаженості мережі, що вимагає оптимізації контрактів для мінімізації витрат та забезпечення економічної доцільності децентралізованих проектів. Смарт-контракти виконують вирішальну роль у функціонуванні екосистеми Ethereum, дозволяючи автоматизувати, стандартизувати та безпечно виконувати комплексні угоди та процеси. Вони не лише виключають потребу у посередництві та знижують можливість зловживань, але й забезпечують значну гнучкість у створенні різноманітних децентралізованих додатків (dApps), які можуть використовувати як окремі особи, так і компанії.

Екосистема розробки на Ethereum включає великий вибір інструментів та фреймворків, таких як Truffle, Hardhat та Remix, які істотно спрощують процес створення, дебагінгу та розгортання смарт-контрактів. Вони забезпечують інтегровані середовища розробки та тестувальні мережі, які дають розробникам можливість оцінювати свої проекти в умовах, максимально наближених до реального блокчейну.

Інтероперабельність є одним з найактуальніших тем блокчейну. Завдяки розвитку технологій, таких як міжланцюгові комунікації (cross-chain

interoperability) та бічні ланцюги (sidechains), смарт-контракти в Ethereum тепер можуть взаємодіяти з іншими блокчейнами, розширюючи свою функціональність та відкриваючи нові можливості для краудфандингових проектів, дозволяючи їм користуватись активами й сервісами з різних мереж.

Практичне впровадження протоколу на Ethereum вимагає ретельного тестування й аудиту смарт-контрактів з метою забезпечення їх надійності та безпеки перед запуском у основну мережу. Перед впровадженням протоколу девелопери повинні провести глибокий аудит коду, включаючи як внутрішній аналіз, так і незалежну перевірку третьою стороною. Аудиторські звіти та рекомендації стають важливою частиною документації, яка допомагає іншим розробникам та користувачам зрозуміти безпековий статус смарт-контракту.

Впровадження децентралізованого краудфандингового протоколу в мережу Ethereum починається з ретельної розробки смарт-контракту на Solidity, який буде виконувати його ключові функції. Після розробки важливим етапом є широко охоплене тестування, що допомагає ідентифікувати та усунути можливі помилки та уразливості.

Транспарентність та відкритість коду для суспільного аудиту забезпечує довіру до протоколу серед користувачів та інвесторів. Це передбачає публікацію коду в репозиторіях, таких як GitHub, та організацію періодів для відгуків та пропозицій від спільноти перед остаточним впровадженням в основну мережу.

Запуск смарт-контрактів вирізняється високою ступенем відповідальності, оскільки один раз розгорнуті контракти стають безпосередньою частиною децентралізованої екосистеми. Це вимагає не тільки глибоких технічних знань з боку розробників, але й розуміння економічних, юридичних та етичних нормативів.

Аудит смарт-контрактів повинен забезпечувати не тільки код, що відповідає стандартам безпеки, але й ефективне споживання газу, щоб мінімізувати вартість транзакцій для користувачів. Для цього, розробники часто звертаються до спеціалізованих аудиторських фірм, які мають досвід перевірки контрактів на Ethereum.

Після аудиту та успішного тестування, етап впровадження включає планомірне введення протоколу на Ethereum через використання тестових мереж (testnets), таких як Ropsten або Rinkeby. Це дає додаткові можливості для виявлення несподіваних помилок в умовах, схожих на реальну експлуатацію, без ризику для реальних коштів користувачів.

Управління версіями смарт-контрактів в Ethereum має критичне значення для забезпечення стійкості та безперервної роботи децентралізованих додатків. Розробники повинні використовувати системи управління версіями, як-от Git, для контролю змін, а також для підтримки переходів між версіями без втрати даних або функціональності. Процес розгортання обновлень до смарт-контрактів повинен бути бережністю синхронізований з фронтендом децентралізованих додатків. Це вимагає чіткого механізму відстеження заявлених функцій, а також повідомлення юзерів про нові версії контрактів та їхні можливості. Використання систем контролю версій для смарт-контрактів також спрощує аудит та ревізію коду. Це дає можливість третім сторонам, як-от іншим розробникам або аудиторам, швидко виявляти зміни в коді та оцінювати їх ймовірний вплив на систему.

Важливо також мати план відкату на випадок неуспішного оновлення чи виявлення кризових уразливостей у нових версіях контрактів. Такі плани допомагають швидко повертати систему до стабільного стану, мінімізуючи потенційні збитки для користувачів і підтримуючи довіру до платформи.

Події (events) у смарт-контрактах Ethereum є потужним механізмом для реєстрації та відстеження змін у стані контракту. Вони використовуються для логування важливих дій та змін, таких як транзакції та зміни у балансах токенів, забезпечуючи тим самим прозорість та аудітованість операций на блокчейні. Розробники можуть визначити події у своїх смарт-контрактах, що дозволяє зовнішнім програмам, таким як веб-інтерфейси та інші децентралізовані застосунки (dApps), реагувати на такі події в реальному часі. Це робить події корисним інструментом для повідомлення користувачів про статус транзакцій та оновленнях у проекті.

Одним із прикладів використання подій у краудфандингових контрактах може бути оголошення про успішне залучення коштів, досягнення цілей збору вкладень, або виплату дивідендів інвесторам. Події також можуть бути використані для сповіщення про надходження вкладень від учасників.

Події використовуються також для інтеграції з аналітичними інструментами, які можуть трекати активність користувачів на платформі та забезпечувати цінні дані для розвитку проекту. Зібрану через події інформацію можна використовувати для створення детальних звітів і дашбордів, які допомагають зрозуміти поведінку інвесторів і налагодити процеси взаємодії. Щоб використовувати події в краудфандингових протоколах ефективно, потрібно ретельно планувати, які події слід запровадити та як вони будуть сповіщати сторони угоди. Оскільки події є необоротними та додають записи в блокчейн, їх організація має бути продуманою та вичерпною, аби відповідати потребам як розробників, так і користувачів.

Оптимальне використання подій вимагає також оптимізації з точки зору витрат на газ, оскільки кожна подія, що запускається, потребує обчислювальних ресурсів. Продуктивне використання подій означає, що вони повинні бути дизайновані таким чином, щоб надавати максимум інформації за мінімальної кількості транзакцій.

Розуміння та використання подій може допомогти в реалізації складних логічних операцій, таких як автоматизовані відповіді на зміни в стані контракту або умовах ринку. До прикладу, зміна в ціні токена або досягнення певного обсягу зборів можуть бути вбудовані як тригери для виклику певних подій. З точки зору аудиту та звітності, події є неоціненним інструментом оскільки вся інформація, зареєстрована подіями, є легко доступною та відстежуваною в блокчейні. Це дозволяє забезпечити повність звітування та спрощує процес відстеження історії проекту, його прогресу та руху коштів.

У зв'язку з краудфандингом, події можуть бути використані для створення комплексного зображення вкладів користувачів і використаних коштів за весь час існування проекту. Така прозорість стимулює здорову ситуацію, сприяє довірі

інвесторів та мотивує проектні команди на забезпечення належного розподілу та використання зібраних ресурсів.

Використання подій у блокчейні дозволяє краудфандинговим платформам створювати інформаційні канали. Це означає, що розробники можуть налаштувати свої додатки таким чином, що кожна важлива дія на платформі може бути легко відстежена, і автоматично надіслати оповіщення користувачам.

Аналітика, що базується на даних подій, може повідомляти організаторів проекту про звукові профілі їхньої аудиторії і показувати, які інвестиції здійснюються найчастіше. Це може допомогти проектам збільшити свою ефективність шляхом оптимізації стратегій маркетингу і налагодження взаємодії з інвесторами. Події також відіграють важливу роль у процесах голосування на децентралізованих платформах. Вони можуть реєструвати голоси та автоматично виконувати колективні рішення, які були ухвалені спільнотою, забезпечуючи таким чином чесний і демократичний процес прийняття рішень.

З огляду на збільшення обсягу даних і складність смарт-контрактів, управління подіями може стати складнішим. Проте, розвиток інструментів для моніторингу та аналізу подій, таких як графічні інструменти "dashboard", може полегшити цей процес і допомогти у відстеженні активностей контрактів без потреби у вдаванні до складних запитів чи прямих взаємодій з блокчейном.

Нарешті, передові практики використання подій можуть відіграти життєво важливу роль у розвитку нових моделей краудфандингу, які будуть ще більш прозорими, ефективними та зорієнтованими на спільноту. З приходом нових технологічних можливостей, краудфандингові платформи, що використовують програмовані події, можуть набути значної переваги у вирішенні викликів сучасного фінансування.

Висновки до розділу

У ході цього розділу було здійснено всебічний аналіз та оцінку різних методів та підходів, які застосовуються при розробці децентралізованих краудфандингових протоколів з використанням мови програмування Solidity на

блокчейні Ethereum. Виявлено ключеві аспекти, які необхідно враховувати при цьому:

1. Solidity залишається надійною та перевіреною технологією для реалізації складних логічних операцій у смарт-контрактах, що є основою для краудфандингових протоколів. Оптимізація програмного коду для зменшення витрат газу і підвищення продуктивності транзакцій є істотною для привабливості протоколу перед користувачами та інвесторами.
2. Роль подій (events) у смарт-контрактах була виділена як істотна для трекінгу активностей та забезпечення аудиторської прозорості, а також інтеграції з аналітичними інструментами для моніторингу проекту.
3. Інтероперабельність та можливість взаємодії з іншими блокчейн-мережами та криптовалютними платформами були розглянуті як суттєві переваги, відкриті завдяки можливостям Solidity та Ethereum, з подальшим розширенням можливостей краудфандингу.
4. Впровадження і управління версіями смарт-контрактів було оцінено як критичний процес для підтримання стабільності та безперервності функціонування краудфандингових платформ, особливо наявляючи стратегії управління ризиками та непередбаченими ситуаціями.
5. Розглянуті практичні аспекти розробки та виклики, пов'язані із розгортанням та тестуванням смарт-контрактів, підкреслили потребу в глибокій спеціалізації та знанні, які необхідні для створення надійних і ефективних рішень для краудфандингу.

Загалом, дослідження підкреслило значущість належного аналізу планування і впровадження передових технологічних практик у контексті розробки децентралізованих краудфандингових протоколів. Інноваційні аспекти та майбутній потенціал Solidity на Ethereum можуть забезпечити нові можливості для вдосконалення та збільшення привабливості краудфандингових платформ.

3. ОПИС РЕАЛІЗОВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір та огляд оптимальних технологій для розробки

Огляд інструментів розробки для Ethereum можна побудувати навколо трьох ключових інструментів: Foundry, Hardhat та Truffle. Кожен з них має свої переваги та недоліки, що робить їх більш або менш підходящими до конкретних завдань і проектів. Давайте розглянемо кожен з цих інструментів докладніше.

Foundry - це порівняно новий інструмент у екосистемі Ethereum, створений з використанням мови програмування Rust, який набирає популярності завдяки своїй швидкості та модулярності. Його компонент "Forge" дозволяє розробникам швидко та ефективно компілювати та тестувати смарт-контракти, забезпечуючи розробникам миттєвий зворотний зв'язок.

Модулярність Foundry є значною перевагою, оскільки розробники можуть вибрати специфічні інструменти для розробки, такі як "Cast" для взаємодії з блокчейном Ethereum і "Anvil" як локальне тестове середовище для смарт-контрактів, що робить підхід до розвитку більш гнучким.

Однією з переваг Foundry є висока продуктивність: тести смарт-контрактів виконуються дуже швидко, що дозволяє розробникам прискорити ітерації розробки і збільшити продуктивність. Це особливо важливо при розвитку складних проектів, де часті тести є необхідними.

З огляду на гнучкість, Foundry дозволяє швидко адаптуватися до змінних вимог проекту. Його здатність інтегруватися з іншими сервісами та інструментами підвищує ефективність розробки і дає спільноті більше можливостей для конкретизації робочих процесів.

Незважаючи на сильні сторони, Foundry все ще може бути складним для новачків через необхідність вивчення нових концептів і інструментів. Крім того, оскільки він досить новий, документація і ресурси для навчання можуть бути менш обширними в порівнянні з більш встановленими інструментами.

Hardhat - це широко використовуваний інструмент для розробки смарт-контрактів Ethereum, який славиться своїм "Hardhat Network", локальним

Ethereum емулятором, який дозволяє розробникам запускати і маніпулювати блоками для вчасного аналізу роботи контракту в реальних умовах. Це імітування дозволяє розробникам дбайливо перевіряти кожен сценарій взаємодії перед тим, як смарт-контракт буде розгорнуто в основну мережу Ethereum.

Hardhat також має зростаючу екосистему плагінів, які розширюють його базовий функціонал і дозволяють інтегрувати додаткові інструменти, які можуть покращити процес розробки. Це включає інструменти для бенчмаркінгу газу, перевірки безпеки контрактів, автоматизованого розгортання та більше.

Основний недолік Hardhat полягає в тому, що його основний функціонал може бути перевантажений для новачків у світі блокчейну. Крім того, хоча Hardhat Network є потужним інструментом для розробки, він може не повністю відтворювати поведінку основної мережі, що може призвести до неточностей у тестуванні. Тим не менш, широта функціоналу Hardhat робить його привабливим вибором для багатьох великих проектів і тих розробників, які шукають всеохоплюючий інструмент для роботи з протоколами Ethereum. Він виступає як міцна середа розробки для продуктивного і професійного створення децентралізованих додатків.

Обираючи Hardhat як основний інструмент розробки, важливо оцінити ступінь готовності своєї команди адаптуватися до його швидкого розвитку та потреб ком'юніті в додаткових можливостях. Такий вибір також залежить від конкретних технічних та бізнес-потреб проекту. Truffle Suite є одним з найдавніших і найбільш впізнаваних інструментів в екосистемі Ethereum. Він пропонує зручні клієнтські інтерфейси і вбудовані рішення для розробки, тестування та розгортання смарт-контрактів, роблячи його добре підходящим для розробників, котрі цінують стабільність та зрозумілість інструментарію.

Ganache, частина Truffle Suite, забезпечує розробникам локальне тестове середовище, де вони можуть емулювати взаємодію смарт-контракту з Ethereum блокчейном, дозволяючи виконувати швидке тестування та відлагодження без витрат на реальний газ. Це робить процес розробки водночас ефективним та економічно вигідним.

Truffle також надає систематичний підхід до розгортання контрактів за допомогою певних міграційних сценаріїв, що є корисним для розробників, які хочуть мати чітку та усталену процедуру оновлення та адміністрування своїх контрактів. Основним недоліком Truffle може бути його відносна складність та великий об'єм інструментарію, що може виявитися важкою для розуміння на початку роботи. Крім того, попри значна сумісність із різними Ethereum вузлами, конфігурація та налаштування Truffle можуть вимагати поглибленого знання і розуміння як смарт-контрактів, так і Ethereum блокчейну в цілому.

У підсумку, Truffle Suite залишається популярним і перевіреним вибором серед розробників, особливо для складних проектів, які вимагають комплексного підходу до розробки, тестування та розгортання. Однак, воно може визначитися як менш привабливий варіант для стартапів або дрібних команд, які шукають більшу швидкість та гнучкість, яку можуть надати інші інструменти, такі як Foundry або Hardhat.

Враховуючи те, що Foundry базується на Rust, це надає йому переваги з точки зору продуктивності та захисту від пам'яті, забезпечуючи новий рівень оптимізації та безпеки для розробників смарт-контрактів. Його швидкість компіляції та тестування також робить його ідеальним вибором для ітеративної розробки та неперервної інтеграції сучасних розробничих практик. З огляду на все це, вибір Foundry може стати виграшним для проектів, які прагнуть максимізувати ефективність розробки смарт-контрактів на Ethereum.

3.2 Імплементація протоколу та його елементів

Науковий внесок запропонованого протоколу полягає в новаторському методі управління та витрачання зібраних коштів. Відхід від традиційної парадигми використання Escrow-смарт контрактів для заморожування коштів до моменту виконання певних умов, протокол вводить механізм, що дозволяє учасникам індивідуально керувати та розпоряджатися власними внесками. Цей підхід не тільки зменшує вразливість до Sybil-атак за рахунок розподіленого

контролю інвестицій, але й сприяє підвищенню прозорості та ефективності процесу витрачання коштів.

Створення краудфандингу починається з визначення токену який краудфандер надаватиме в обмін на інвестовані кошти, за принципами ICO. Таким токеном може виступати токен стандарту ERC-20, або ж токен стандарту ERC-721. Після чого, краудфандер має визначити токен який він збирає, це токен стандарту ERC-20.

Наприклад, краудфандер створює проект “Pepe Frog”, і створює власний токен $PEPE, який надаватиме інвесторам в обмін на їх USDT, що є аналогом доллара в світі криптовалют.

Також, краудфандер має визначити вищу та нижчу планку свого краудфандингу (Hard Cap, Low Cap). При недоборі за визначений протоколом час коштів до нижчої планки визначеної краудфандером, краудфандинг вважається проваленим і самовидаляється з платформи, повертаючи кошти всім початковим власникам. При наближенні зібраних коштів до вищої планки, імплементацією гарантується, що зібрани кошти не вийдуть за рамки вищої планки. Вищеописана стадія краудфандингу називається “посівною”, і на цій стадії проект залучує кошти. Після її завершення, починається стадія “будування”, де і відбуватимуться ітеративні витрати.

Введення коштів у краудфандинг-протоколі ініціюється шляхом взаємодії інвесторів зі смарт-контрактом, який приймає внески і фіксує їх у блокчейні. В основі моделі лежить принцип, що сума зібраних коштів відповідає загальній сумі внесків усіх інвесторів. Математично це виражається як $amount_{raised} = \Sigma_{i=1}^{n} amount_{invested}[i]$, де i - індекс інвестора в масиві інвесторів, а n - загальна кількість інвесторів.Ця модель відображає прозору та аудитовану структуру фінансування, де кожен внесок є верифікованим і зафіксованим на блокчейні.

Коли краудфандер має намір використати частину зібраних коштів, він оголошує про це через блокчейн, ініціюючи тим самим процес, який дозволяє інвесторам відкликати свої внески протягом визначеного періоду часу (

$t_{refundStartDeadlineDelay}$). Цей механізм додає гнучкості в управління краудфандингом, надаючи інвесторам можливість забезпечити свої інвестиції в разі зміни умов або появи непередбачених ризиків. Цей період також є захистом від потенційного FUD, що може спотворювати ринкові умови та інвестиційні рішення.

Якщо інвестори вирішують не відкликати свої кошти до кінця встановленого часу, ці кошти автоматично стають доступними для краудфандера. Така структура впроваджує секвенційний принцип запитів на виведення коштів, забезпечуючи ритмічний потік фінансування та уможливлює постійне оновлення фінансової стратегії проекту. Це сприяє створенню довіри між краудфандером та інвесторами, оскільки кожна сторона має чітке розуміння своїх прав та можливостей у рамках протоколу.

Процес повернення коштів в рамках децентралізованого краудфандинг-протоколу включає можливість для інвесторів запитувати виведення своїх вкладень протягом встановленого часового вікна, визначеного параметром $t_{refundStartDeadlineDelay}$. У разі такого запиту, частина коштів інвестора переводиться у статус "заморожених" ($amount_{freezed}$), що запобігає їх використанню краудфандером до закінчення визначеного періоду ($t_{refundDelay}$). Цей механізм враховує можливість зміни рішення інвестором, що дозволяє повернути внесок у випадку непередбачених обставин або змін у проекті.

Краудфандер, який ініціює зняття коштів, має розуміти, що загальна доступна для виведення сума ($amount_{availableToWithdraw}$) буде зменшуватися з кожним раундом повернення коштів інвесторами. Формально, для кожного раунду k, сума доступних для краудфандера коштів розраховується як $amount_{availableToWithdraw} =$

$\Sigma_{i=1}^{n}(amount_{invested}[i] - amount_{refunded}[i] - amount_{freezeded}[i])$, де n - кількість інвесторів. Це враховує інвестиції ($amount_{invested}[i]$), повернені кошти (

$amount_{refunded}[i]$) та заморожені суми в очікуючі на можливе повернення ($amount_{freezed}$).

Водночас, інвестори мають можливість збільшити свої внески (($amount_{invested}[i]$), якщо бажають збільшити свою участь у фінансуванні проекту. Таке збільшення внесків також відображається в математичній моделі, адаптуючи загальну суму доступних коштів відповідно до нових інвестицій. Це вносить динаміку в краудфандинговий процес і дозволяє краудфандеру мати більш точне уявлення про поточний фінансовий ресурс, доступний для використання.

Далі буде наведений опис імплементованих модулів протоколу.

Бібліотека AddressArrayUtils

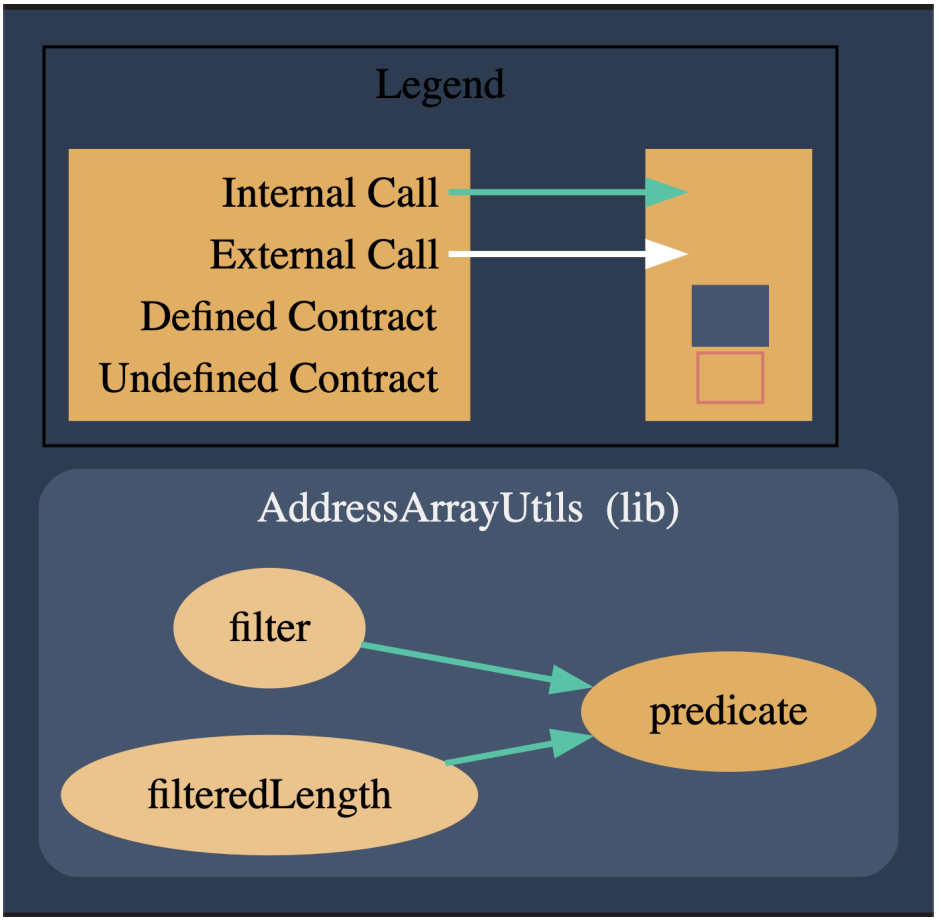


Рисунок 5. Діаграма викликів контракту AddressArrayUtils

Бібліотека AddressArrayUtils надає утилітні функції для роботи з массивами адрес Ethereum. Вона дозволяє фільтрувати массиви з використанням переданої предикат-функції та рахувати кількість елементів, які задовільняють дану умову.

- filter - функція приймає массив адрес та предикат (умова). Вона повертає новий массив, який містить лише ті адреси, що виконують задану умову.
- filteredLength - ця функція повертає кількість елементів в массиві, які виконують задану умову, без створення нового массиву.

Контракт SaleRounds

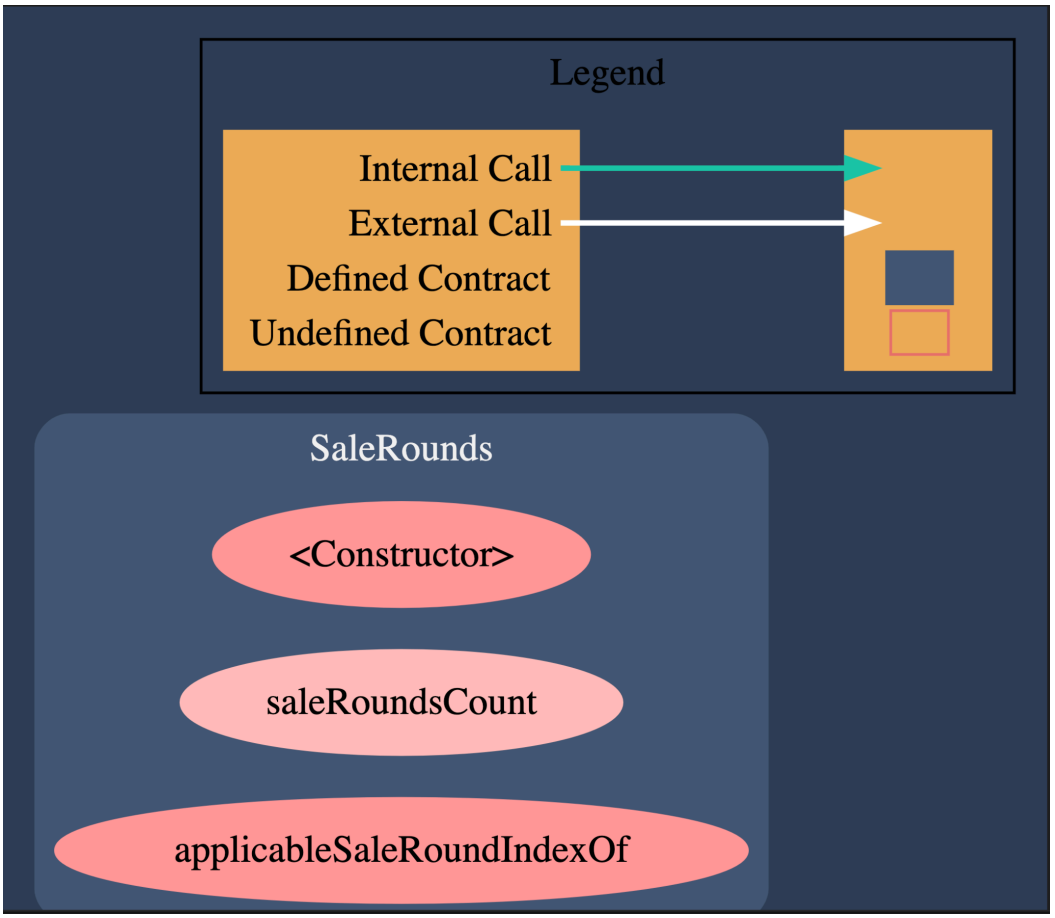


Рисунок 6. Діаграма викликів контракту SaleRounds

Контракт SaleRounds описує логіку створення та управління раундами продажів (розподілу токенів) у краудфандинговій кампанії.

- Конструктор ініціалізує массив saleRounds, куди можна додати певну кількість раундів продажів.
- saleRoundsCount - функція повертає загальну кількість раундів продажів.
- applicableSaleRoundIndexOf - ця функція приймає дані транзакції та

повертає індекс активного раунду продажів, якщо такий існує.

Контракт BuildRounds

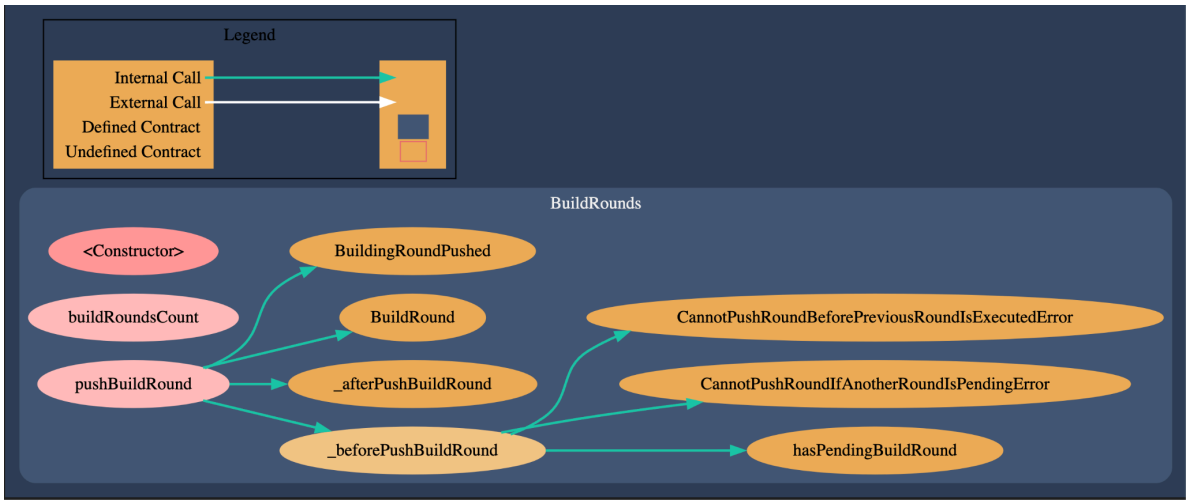


Рисунок 7. Діаграма викликів контракту BuildRounds

Контракт BuildRounds управляє раундами розвитку проекту, дозволяючи команді проекту запитувати фінансування для окремих раундів реалізації проекту.

- Використовує executionDelay - часову затримку для запуску наступного раунду після створення попереднього.
- pushBuildRound - дозволяє додавати нові раунди розвитку з певною сумою фінансування та описом.
- hasPendingBuildRound - перевіряє чи існує невиконаний раунд розвитку.

Контракт Refunds

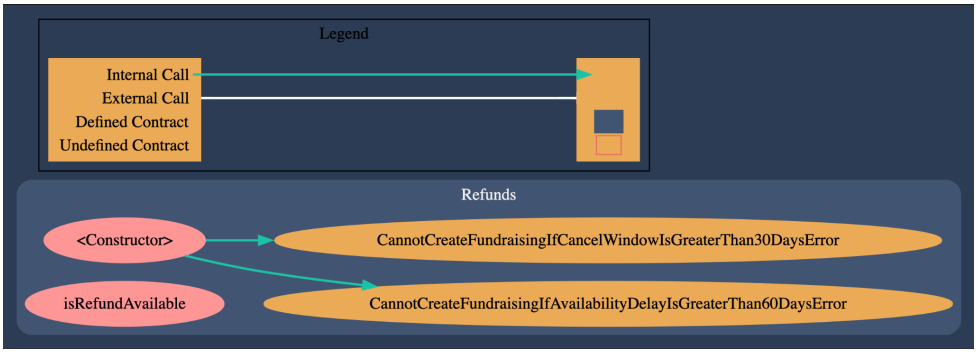


Рисунок 8. Діаграма викликів контракту Refunds

Контракт Refunds забезпечує механізм повернення коштів інвесторам, якщо

кампанія не досягає своїх цілей або виникають інші визначені умови.

- Встановлює умови для доступності повернення фінансів, включаючи availabilityDelay (затримку доступності) та cancelWindow (період, протягом якого можливо скасувати запит на повернення коштів).
- isRefundAvailable - функція перевіряє, чи настала умова, за якою можливе повернення коштів, заснована на загальному часовому плані кампанії та певних тривалих метах розвитку проекту.

Контракт Fundraising01Base

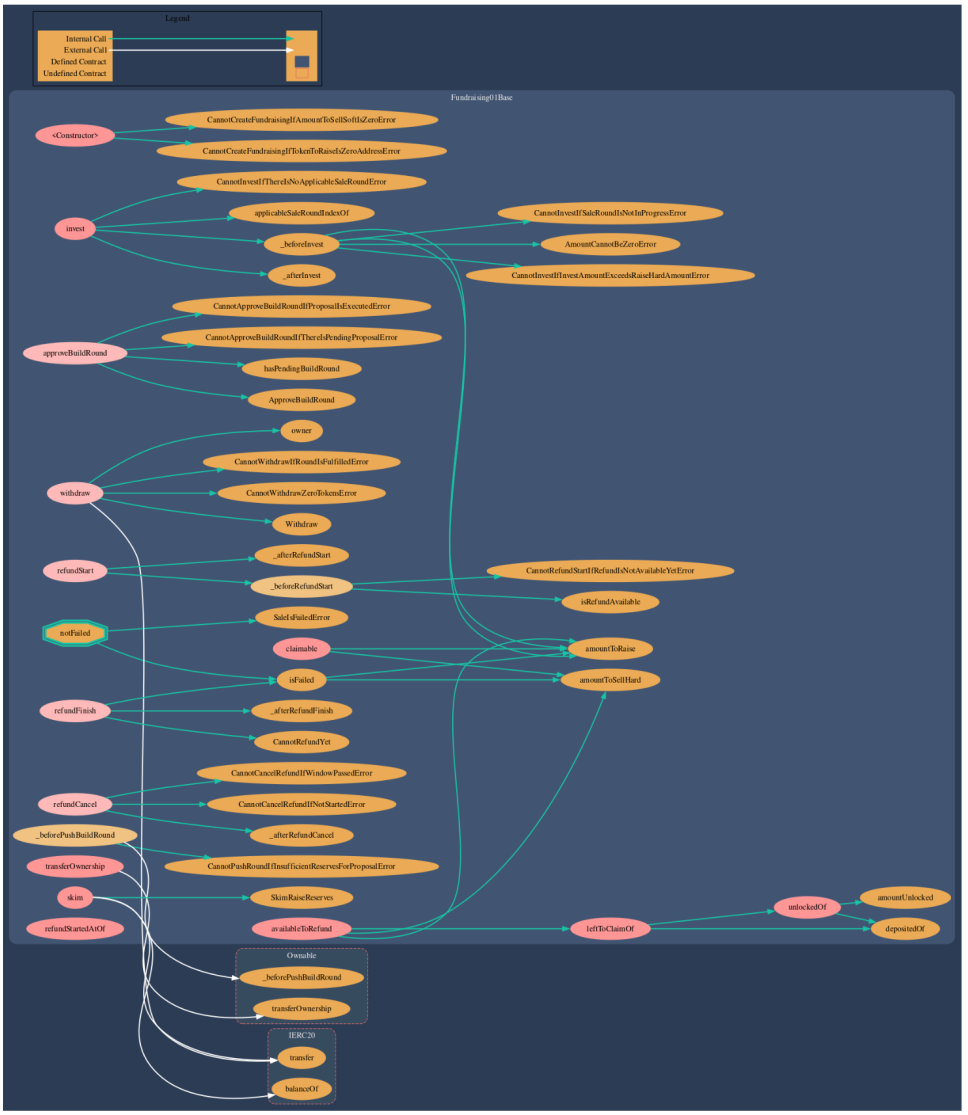


Рисунок 9. Діаграма викликів контракту Fundraising01Base

Контракт Fundraising01Base є основою для фандрайзингової платформи, яка об'єднує в собі логіку повернення коштів (Refunds), раундів продажів

(SaleRounds), та розвитку проекту (BuildRounds).

- Змінні createdAt, amountToSellSoft, tokenToRaise визначають базову конфігурацію кампанії, включно з часом створення, м'якою капіталізацією для успішної кампанії, та адресою токена, який використовується для залучення інвестицій.
- Структура InvestorData служить для зберігання інформації по кожному інвестору, включаючи суму інвестиції та зарезервовані на повернення кошти.
- Модифікатор notFailed перевіряє статус кампанії, аби гарантувати, що операції, такі як інвестиції та виводи, здійснюються лише якщо кампанія не вважається провальною на даному етапі.

Контракт Fundraising01ERC20

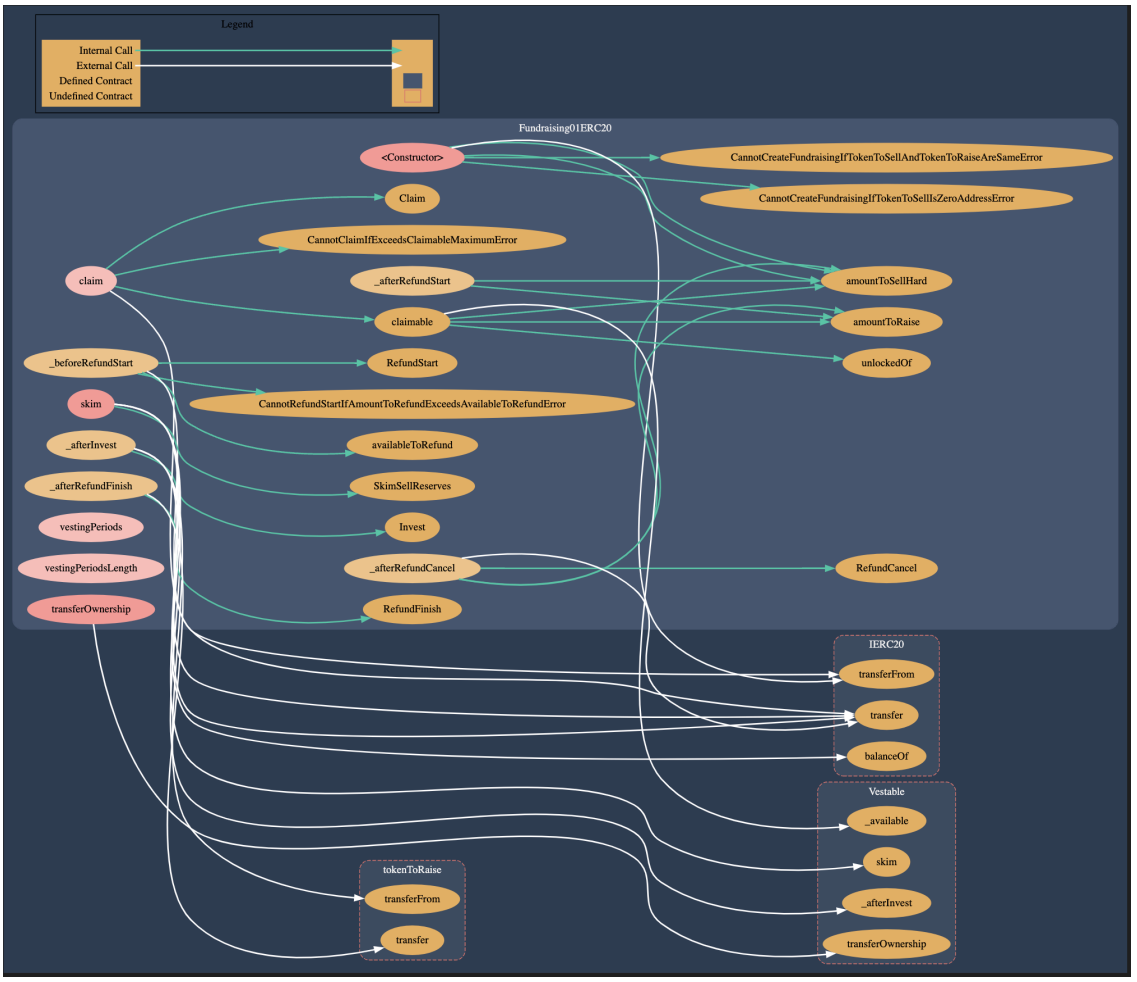


Рисунок 10. Діаграма викликів контракту Fundraising01ERC20

Fundraising01ERC20 реалізує модель фандрайзингу, дозволяючи користувачам вкладати кошти у вигляді криптовалюти ERC20, та керувати циклами інвестицій, виплати, та проектного розвитку на основі стандартів, встановлених в Fundraising01Base.

- Інвестиційна логіка в Fundraising01ERC20 включає функції, які дозволяють користувачам здійснювати інвестиції, автоматично розміщуючи їхні кошти в контракт і одночасно враховуючи ці кошти як "залучені".
- За допомогою функції claim, інвестори можуть вимагати виплату токенів пропорційно до їх вкладу за результатами успішних етапів кампанії, що забезпечує справедливе та прозоре розподіл прибутків.
- Механізм повернення коштів в контракті включає в себе процедури для ініціювання, відміни та завершення процесу повернення, дозволяючи інвесторам відкликати свої внески в певних обставинах, вказаних в RefundConfiguration.

Через динаміку і високий рівень ризику, пов'язаних з краудфандинговими кампаніями, контракт ретельно побудований з урахуванням потенційних перепон та вимог до управління ресурсами. Оскільки токени можуть зберігатись або обігатися на платформі в межах контракту, створена структура InvestorData та набір функцій для управління станом вкладень кожного інвестора є особливо важливими для запобігання шахрайства та надання гарантій.

Використання контракту Fundraising01ERC20, що базується на солідних принципах краудфандингу, дозволяє розробникам використовувати стандартизований підхід до залучення інвестицій та управління капіталом в рамках децентралізованого фінансування. Водночас, наявність Ownable модифікатора направлена на те, щоб надавати адміністратору контракту (який здебільшого є розробником проекту), здатність передавати право власності чи керувати ключовими особливостями контракту, забезпечуючи таким чином гнучкість управління та можливість швидкої реакції на нестандартні ситуації, що можуть виникати під час життєвого циклу краудфандингової кампанії.

Контракт Fundraising01ERC721

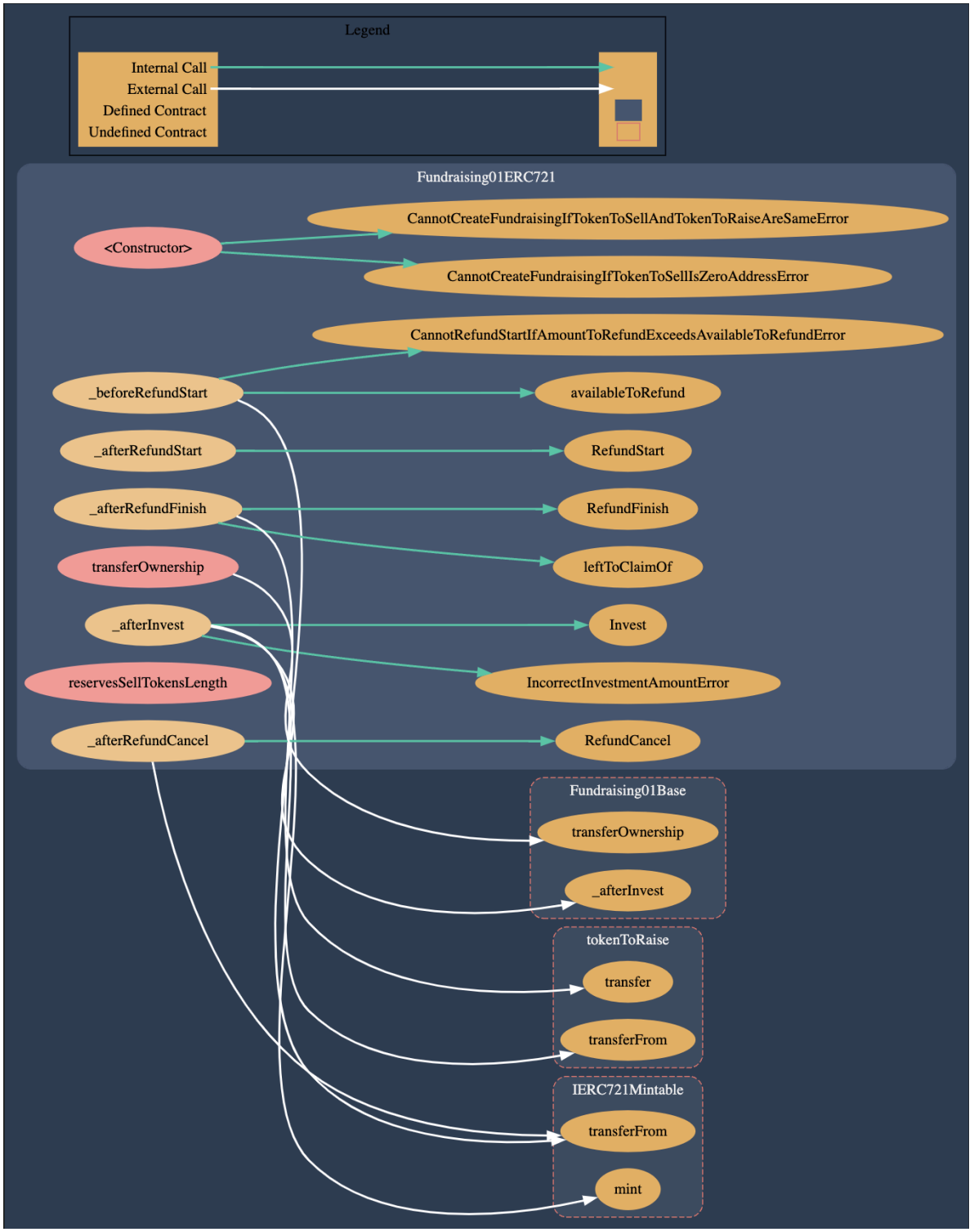


Рисунок 11. Діаграма викликів контракту Fundraising01ERC721

Контракт Fundraising01ERC721 розроблений як модель для впровадження краудфандингових кампаній, орієнтованих на невзаємозамінні токени (NFTs). Він будується на базі Fundraising01Base, інтегруючи в себе механізми з попередніх раундів, такі як SaleRounds на продаж токенів та BuildRounds для розвитку

проекту, а також включає систему повернення коштів Refunds. Важливою особливістю контракту є те, що NFTs не передаються автоматично під час створення контракту, а поповнюються за допомогою функції replenish(...).

Ключові компоненти контракту:

1. tokenToSell: Інтерфейс IERC721Mintable, який використовується для взаємодії з токенами, що продаються в рамках краудфандингової кампанії. Він визначає контракт токена, з яким буде працювати Fundraising01ERC721.
2. sellTokens: Масив uint256, який містить ідентифікатори токенів, що доступні для продажу. Цей список токенів може бути поповнений власником контракту за допомогою виклику зазначеної функції replenish(...).
3. tokensOf: Мапінг, який відображає адресу інвестора в масив ідентифікаторів токенів, які вони купили чи якими вони володіють.
4. frozenTokensOf: Аналогічний до tokensOf мапінг, який використовується для відстеження заморожених або зарезервованих для повернення токенів. Ці токени перебувають у "замороженому" стані під час процесу повернення коштів.
5. reservesSellTokens: Масив, який використовується для відстеження токенів, які були повернуті або покладені в резерв для майбутніх продажів або повернень.

Функції контракту:

- replenish(...): Функція, яку викликає власник для поповнення списку доступних для продажу NFTs.
- _afterInvest(...): Змінена версія функції з базового контракту Fundraising01Base, адаптована для роботи з NFT. Вона обробляє логіку після внесення інвестиції, включаючи присвоєння NFT та виклик функції mint для створення нового токену.

Функції повернення коштів:

- _beforeRefundStart(...): Передує процесу повернення коштів, перевіряючи, чи інвестор дійсно може розпочати процедуру повернення NFT, відповідно до кількості токенів, що має бути повернута.

- _afterRefundStart(...): Реалізує логіку яка починає процес повернення коштів, зокрема, заморожуючи токени інвестора в контракті, вказуючи, що ці токени беруть участь у процедурі повернення.
- _afterRefundFinish(): Завершує процес повернення коштів, передаючи ERC20 токени назад інвестору, знімаючи заморожені NFT з їх списку та додаючи їх до резервів контракту.
- _afterRefundCancel(): Відміняє процес повернення коштів, якщо інвестор протягом встановленого часу скасовує своє рішення про повернення та передає NFT назад у їхню власність.

Ці функції повернення коштів сприяють захисту інвесторів, оскільки надають їм можливість отримання вкладів назад у разі зміни обставин або якщо проект не відповідає їх очікуванням.

Функції допоміжного характеру:

- reservesSellTokensLength(): Функція для отримання довжини списку резервних токенів, що допомагає оцінити поточний резерв та кількість токенів доступних для майбутнього продажу чи повернення.

Впровадження такої системи означає, що краудфандинг базується на прозорих і чесних фундаментах, де кожен учасник має повний контроль та інформацію про стан своїх внесків і володіння. У разі збоїв або невдач, механізми повернення коштів гарантують інвесторам право на повернення їхніх активів, знижуючи ризики і підвищуючи довіру до краудфандингової платформи.

Використання ERC721 (NFT) токенів замість ERC20 може привносити додаткову вартість у вигляді унікальності, можливості збирання та інших аспектів, які не доступні в стандартних токенах, тим самим збагачуючи внутрішню економіку краудфандингової платформи та пропонуючи інвесторам більш гнучкі та цікаві варіанти інвестування в краудфандинг.

Контракт Fundraising01SBT

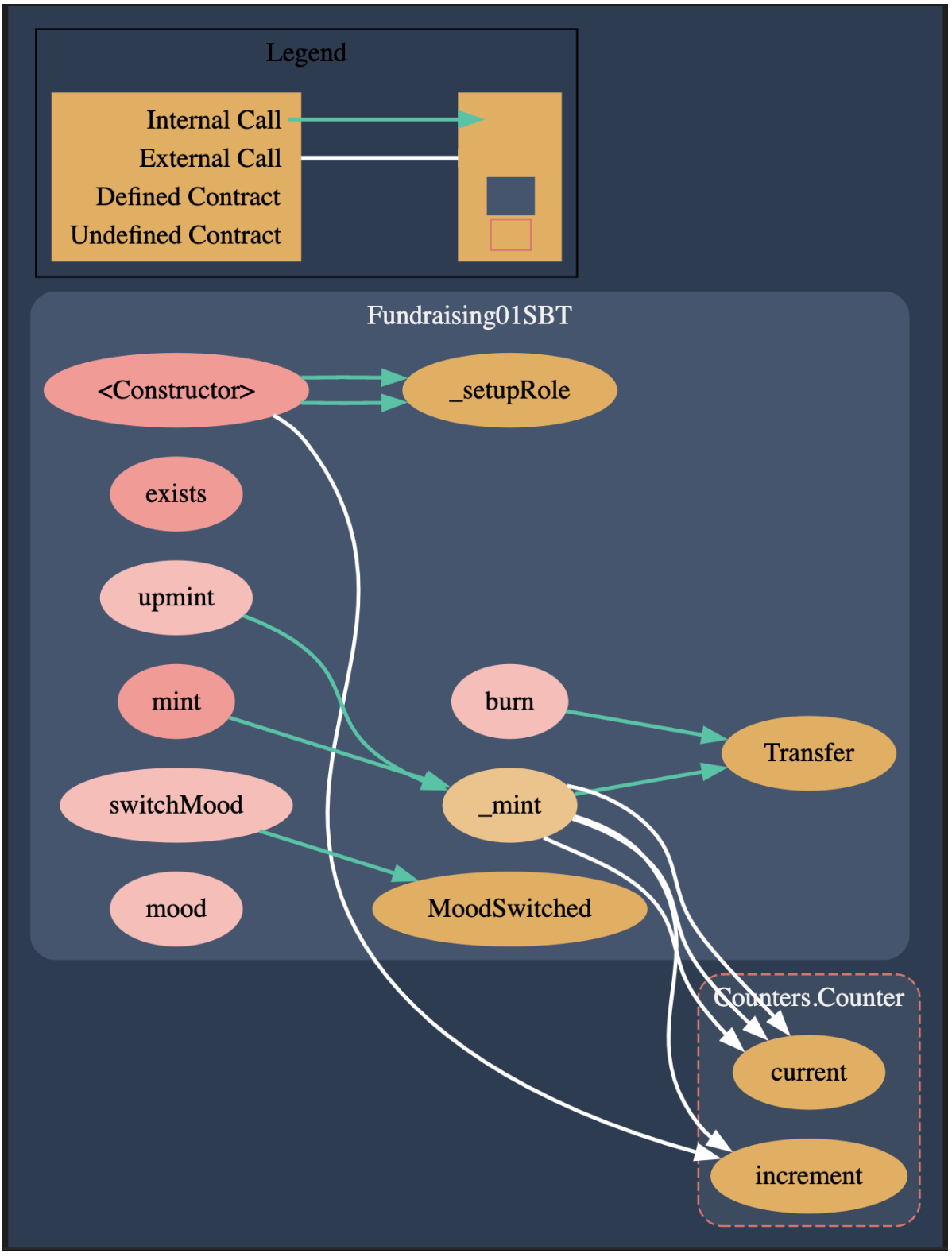


Рисунок 12. Діаграма викликів контракту Fundraising01SBT

Контракт Fundraising01SBT представляє собою реалізацію інтерфейсу для токенів на основі стандарту ERC-721 (non-fungible tokens, NFTs), які використовуються в краудфандингових цілях. Він розширює можливості базового фандрайзингового контракту Fundraising01Base і діє як апгрейдований

проксі-контракт, оскільки стандарт SBT (Soulbound Token) зараз не є стандартизованим.

Особливість Fundraising01SBT полягає в тому, що він дозволяє репрезентувати інвестиції та голоси членів спільноти у формі NFT, забезпечуючи більшу гнучкість та нові можливості у сфері краудфандингу та участі у рішеннях.

Ключові елементи контракту включають:

- ISSUER_ROLE: Статична константа, яка представляє хеш ролі емітента (Issuer), використовуваної для надання дозволів на створення та спалення токенів.
- Transfer: Подія, яка фіксує передачу токена від однієї адреси до іншої та включає в себе ідентифікатор переданого токена.
- MoodSwitched: Подія, котра фіксує зміну "настрою" (mood) токену, що може означати голосування або іншу бінарну зміну стану, пов'язану з конкретним токеном.

Функції контракту:

- exists: Перевіряє, чи існує токен з вказаним ідентифікатором (tokenId), повертаючи true, якщо токен існує, та false в протилежному випадку.
- upmint: Функція, котра дозволяє емітенту створювати нові токени для певної адреси, якщо на вказаній адресі ще не існує SBT.
- mint: Функція, що дозволяє емітенту створювати нові токени для певної адреси, але з перевіркою на те, що ця адреса ще не володіє SBT.
- burn: Дозволяє емітенту "спалювати" токени (які еквівалентно видаляються з обігу), видаляючи їх з власності конкретного користувача.
- switchMood: Функція, яка дозволяє інвесторам перемикати свої преференції або "настрої" в межах певної краудфандингової кампанії.
- mood: Цей метод надає інформацію про "настрій" конкретного інвестора стосовно конкретного раунду продажу, повертаючи булеве значення, що відображає їхню позицію (за або проти).
- _mint: Внутрішня функція, яка відповідає за власне створення SBT, закріпленого за певною адресою учасника. Виділений внутрішній лічильник

_tokenIdCounter використовується для гарантування унікальності кожного токену.

- _tokenIdCounter: Змінна-лічильник, яка слугує для присвоєння унікальних ідентифікаторів кожному випущеному токену, гарантуючи, що кожний SBT унікальний.
- ownerOf: Відображення, що зберігає відносину "токен-власник",, яке дозволяє швидко визначити власника конкретного токену по його ID.
- holds: Відображення, що містить інформацію про токени, які утримує конкретний користувач, зазначаючи ID токена, асоційованого з їхньою адресою.
- metadata: Внутрішня структура даних для зберігання метаданих токенів, помічених ідентифікатором раунду продажу та ID токену. Ці метадани можуть включати додаткову інформацію, таку як лояльнісні очки та інші характеристики.

Конструктор контракту Fundraising01SBT ініціалізує ролі доступу та підготовляє фундамент для створення та управління SBT. Використання EIP712 стандарту дозволяє реалізувати типізовані структури даних та їхній цифровий підпис, підвищуючи безпеку та достовірність операцій.

Така структура контракту Fundraising01SBT забезпечує прозору основу для краудфандингових ініціатив, де учасники можуть не тільки вкладати свої ресурси, а й активно взаємодіяти з екосистемою, змінюючи свою поставу й голосуючи в процесах спільного прийняття рішень, що може відбиватися в "настроях" асоційованих із кожним SBT. Ця функціональність надає можливість затверджувати значущість кожного голосу та сприяє залученню спільноти в процес ухвалення важливих стратегічних рішень, що підкріплюють основи демократичного фінансування в рамках децентралізованого краудфандингу.

Контракт Fundraising01FactoryERC20

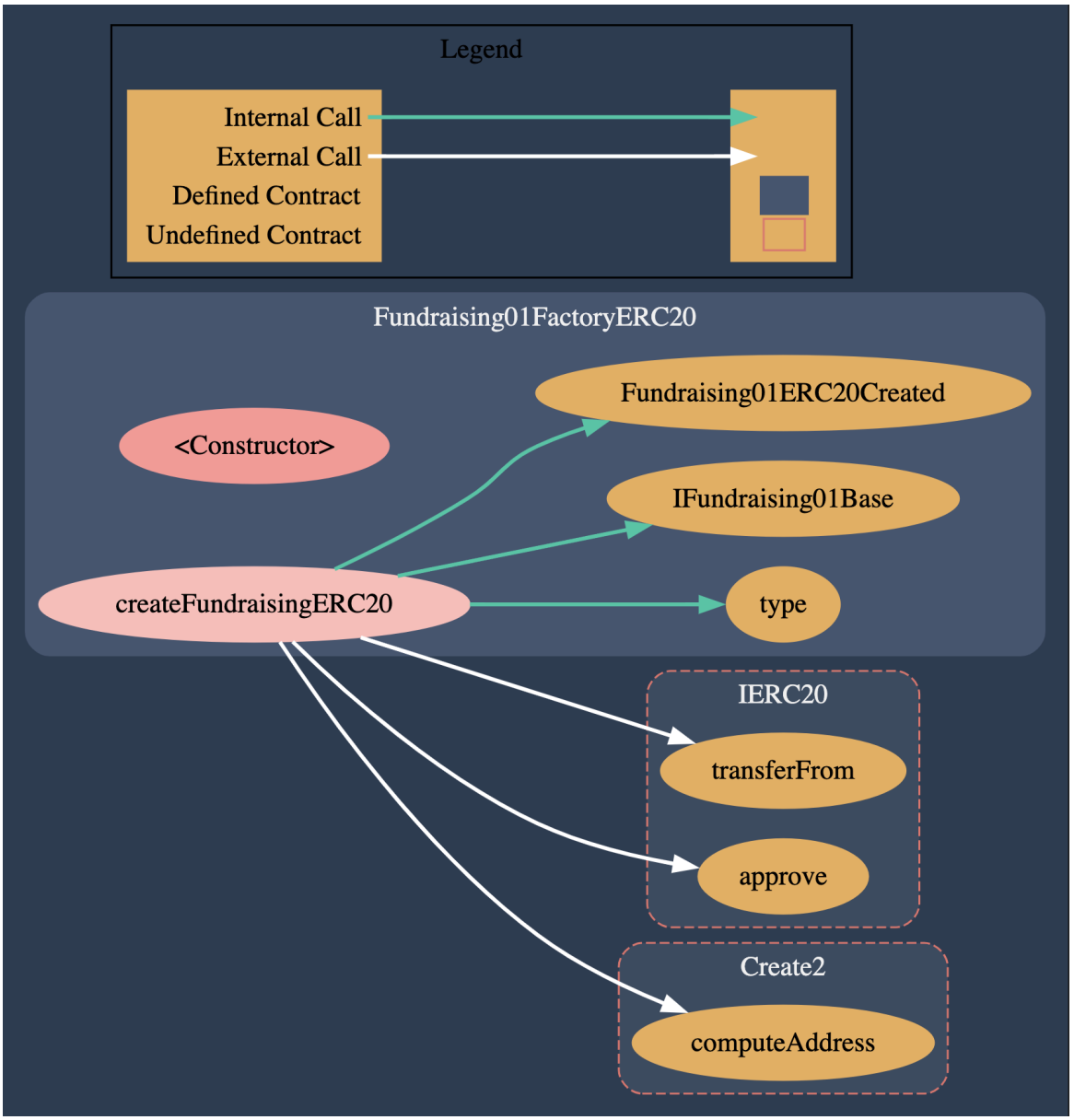


Рисунок 13. Діаграма викликів контракту Fundraising01FactoryERC20

Контракт Fundraising01FactoryERC20 виконує роль фабрики для створення екземплярів контрактів Fundraising01ERC20. Він дозволяє розгортати нові краудфандингові кампанії з використанням токенів ERC20 за визначеними параметрами. Використовуючи протоколи OpenZeppelin та власні інтерфейси, цей контракт спрощує процес створення та керування краудфандинговими кампаніями.

Ключові елементи контракту:

- WETH: Вбудована змінна, яка визначає адресу обгортки Wrapped Ether (WETH), що використовується для спрощення взаємодії з Ether в рамках ERC20 стандарту.

- createFundraisingERC20: Функція, яка викликається для створення нового контракту Fundraising01ERC20. Вона приймає параметри, які описують характеристики краудфандингової кампанії, і генерує новий контракт з використанням засобів OpenZeppelin Create2 для детермінованого розгортання.

Кроки розгортання:

1. Передача Токенів: Розпочинаючи процес, розробник (ініціатор краудфандингу) передає необхідну кількість токенів, яка буде продана, до контракту фабрики.
2. Кодування байткоду: Фабрика кодує байткод нового контракту Fundraising01ERC20, готувачи його до створення.
3. Солювання для Uniqueness: Фабрика використовує хеш від обраного набору параметрів, включаючи токени, за допомогою keccak256 для забезпечення унікальності адреси новоствореного контракту.
4. Approve для Create2: Фабрика викликає функцію approve для надання дозволу на витрату токенів ERC20, що забезпечить здатність контракту краудфандингу управляти токенами після його створення.
5. Створення Контракту: Використовуючи опкод create2, фабрика розгортає новий контракт на адресу, яка була визначена раніше за допомогою процесу солювання.
6. Передача Власності: Після створення контракту власність та попечительство над ним передається ініціатору краудфандингової кампанії.

Цей процес розгортання забезпечує те, що кожна краудфандингова кампанія має свій власний унікальний і незалежний екземпляр контракту, що дозволяє індивідуальне управління та підлаштовування під вимоги конкретного проекту. Завдяки ізоляції кожної кампанії забезпечується безпека активів інвесторів та надається гнучкість у виборі стратегій залучення коштів.

Контракт Fundraising01FactoryERC721

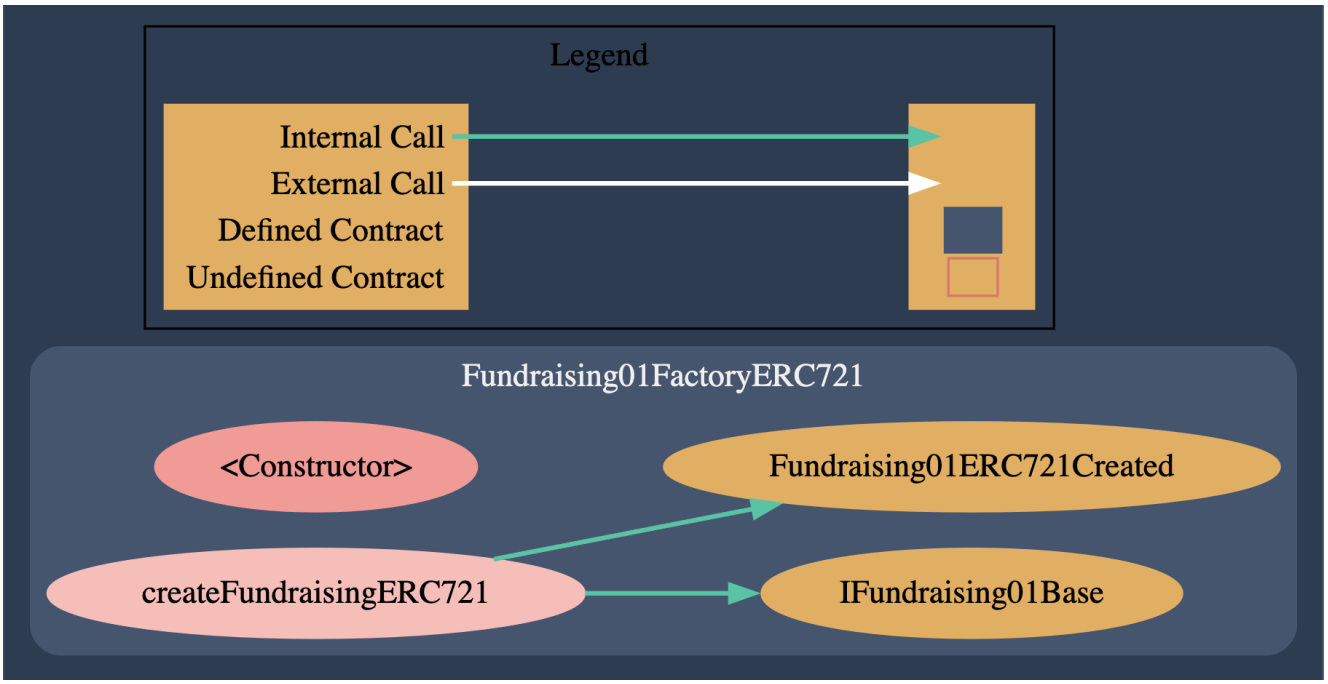


Рисунок 14. Діаграма викликів Fundraising01FactoryERC721

Fundraising01FactoryERC721 є фабричним контрактом, розробленим для створення інстанцій краудфандингових контрактів на базі невзаємозамінних токенів (NFTs) у форматі ERC-721. Контракт полегшує процес розгортання власних краудфандингових кампаній для творців проектів, надаючи шаблон інструментів для ефективного управління раундами залучення коштів і розповсюдження унікальних активів.

Ключові характеристики контракту:

- WETH: Вбудований, незмінюваний держатель, який вказує на адресу контракту Wrapped Ether (WETH), використовуваного як форма оплати або інвестування у вигляді ERC-20 токенів.
- createFundraisingERC721: Головна функція контракту, яка дозволяє створити новий контракт краудфандингу Fundraising01ERC721. Функція пропонує механізм створення контрактів із визначеними параметрами і атрибутами кампанії.

Процес створення краудфандингової кампанії включає кілька кроків:

1. Приготування salt Хешу: Користується унікально сгенерованим хешом з використанням keccak256, який забезпечує створення детермінованих адрес смарт-контрактів, базуючись на переданих аргументах.

2. Розгортання Fundraising01ERC721: За допомогою створеного salt, фабрика ініціалізує і створює новий екземпляр контракту Fundraising01ERC721 із визначеними розробником параметрами для управління кампанією.

3. Передача Власності: Після успішного розгортання нового контракту, фабрика переводить управління його власністю творцю кампанії.

4. Подія Fundraising01ERC721Created: Подія генерується після розгортання нового контракту, що реєструє створення інформує внутрішню систему і користувачів про наявність нової кампанії.

Використання цього контракту відкриває шляхи стандартизації у процесі створення краудфандингових кампаній та зменшує технічні бар'єри для творців, дозволяючи зосередитися на творчих та маркетингових аспектах проекту, замість технічного навантаження. Такий підхід спрощує інтеграцію з блокчейном Ethereum і знижує поріг входу для нетехнічних учасників.

Контракт Fundraising01Factory

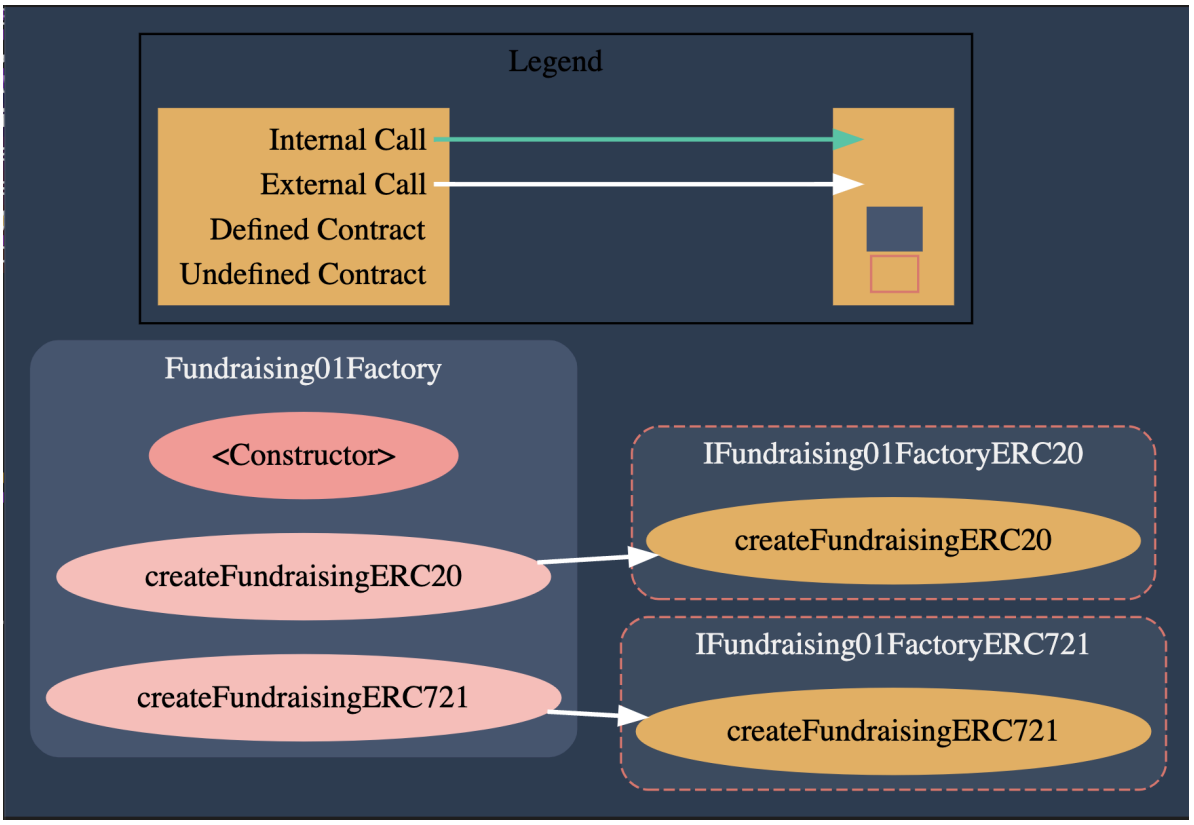


Рисунок 15. Діаграма викликів Fundraising01Factory

Fundraising01Factory є основною фабрикою, призначеною для створення інстанцій двох типів краудфандингових кампаній: з використанням ERC20 токенів та ERC721 токенів (NFTs). Цей контракт служить центральною точкою для взаємодії з фабриками ERC20 та ERC721, спрощуючи для користувачів процес створення кампаній та управління ними.

Ключові особливості контракту:

- sales: Масив, що містить адреси всіх створених краудфандингових кампаній, забезпечуючи індексований доступ та перегляд існуючих краудфандингових проектів.
- factoryERC20: Імутабельна (незмінна) змінна, що містить адресу фабрики Fundraising01FactoryERC20, яка використовується для створення кампаній на основі стандарту ERC20.
- factoryERC721: Аналогічна імутабельна змінна, що містить адресу фабрики Fundraising01FactoryERC721, яка використовується для створення кампаній на основі стандарту ERC721.

Процес створення кампаній включає наступні етапи:

1. Підготовка до створення кампанії ERC20: Користувач (ініціатор кампанії) викликає функцію createFundraisingERC20, передаючи параметри, що описують майбутню краудфандингову кампанію ERC20, до фабрики Fundraising01FactoryERC20. Після успішного створення, адреса створеної кампанії додається до масиву sales.
2. Підготовка до створення кампанії ERC721: Аналогічний процес використовується для створення кампанії ERC721, де користувач викликає функцію createFundraisingERC721, надаючи детальну інформацію про майбутню кампанію. Адреса створеної кампанії також додається до масиву sales.

В обох випадках використовується аргумент _creator для визначення того, хто буде власником кампанії, та викликається метод transferOwnership з базового контракту IFundraising01Base, щоб власником стали саме ініціатори.

Fundraising01Factory відіграє ключову роль у децентралізованому

краудфандингу, оскільки він забезпечує стандартизований механізм створення і управління кампаніями, а також гарантує, що власники мають повний контроль над своїми проектами та можуть керувати їх у міру того, як проект розвивається та змінюється. Інтерфейс Fundraising01Factory дозволяє користувачам вибирати між різними типами токенів та краудфандингових підходів, забезпечуючи гнучкість для інноваційних та традиційних стратегій залучення коштів.

Висновки до розділу

Імплементація краудфандингового протоколу через смарт-контракти на платформі Ethereum використовує потужність ERC-20 та ERC-721 стандартів для залучення коштів та інвестицій. Використання цих стандартів сприяє створенню гнучкого та масштабованого рішення, здатного задовольнити потреби широкого спектру краудфандингових кампаній.

Обрані інструменти для розробки, та підтримки коду включаючи Foundry, Hardhat і Truffle Suite, забезпечують розробникам комплексний набір засобів для створення, тестування та розгортання смарт-контрактів. Вибір конкретного інструменту залежить від потреб проекту, пріоритетів швидкості розвитку, простоти використання та рівня контролю над процесом розгортання.

Fundraising01FactoryERC20 і Fundraising01FactoryERC721 використовуються як центральні точки для створення та управління їх відповідними кампаніями. Процес створення кампанії автоматизований і оптимізований завдяки цим фабричним контрактам, забезпечуючи ефективність і зниження потенційних помилок.

Розгортання та управління версіями смарт-контрактів узгоджено з внутрішньою архітектурою та процесами безпеки Ethereum, забезпечуючи стабільність та надійність реалізованих краудфандингових кампаній.

Платформа розробки надає багатий інструментарій для контролю над процесом впровадження протоколу та його подальшої діяльності, включаючи дії, такі як інвестування, претензії на токени, повернення коштів та управління внутрішньоми заходами користувачів.

Використання інноваційних засобів, як наприклад Create2 для детермінованого розгортання смарт-контрактів і вбудованого функціоналу EIP-712, підкреслює високий рівень інтеграції протоколу з передовими технологіями блокчейну Ethereum.

Враховуючи сучасну динаміку блокчейн-ринку та швидке впровадження децентралізованих фінансових інструментів, протокол Fundraising01 з його гнучкістю та масштабованістю має потенціал стати ключовим рішенням для проектів, що прагнуть залучити капітал безпосередньо від спільноти.

Ці висновки демонструють можливості та перспективи імплементації краудфандингового протоколу на платформі Ethereum і є важливою відправною точкою для забезпечення успішної реалізації та розгортання краудфандингових кампаній, які можуть вигідно відрізнятися за допомогою технологій блокчейну та належно підготовленої інфраструктури розробки.

4 ВИПРОБУВАННЯ РОЗРОБЛЕНОГО ПРОТОКОЛУ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1 Тестування роботи протоколу та аналіз надійності результатів

Для полегшення процесу тестування, була розроблена мінімалістична веб платформа для взаємодії з протоколом. Протестуємо протокол з допомогою веб-платформи.

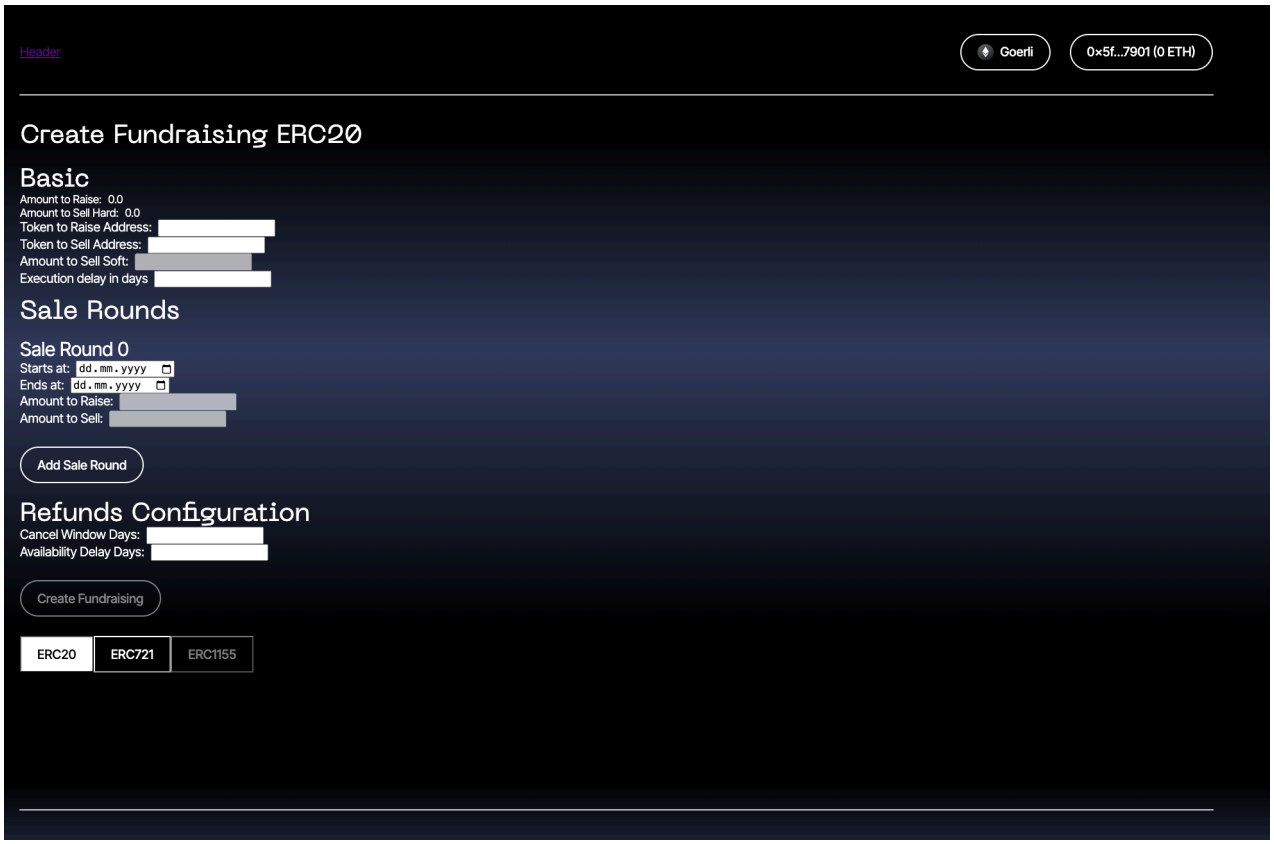


Рисунок 16. Створення краудфандингу.

Для тесту, було створено два токени стандарту ERC20, доступні на тестнет блокчейні Ethereum Goerli за адресами: 0x6B0A14c56F8e7f4a9c9543539E24bB7279DfcA9a та 0x71E589b828bdEf7C8b92BA0bdbd35bD14FB4B663. Сам контракт краудфандингу доступний за адресою 0xC3c79c5764288dDa10c25C7e9A93E518d9C68B11. Вища планка збору коштів була виставлена на 300000000000000000000000000 одиниць, нижча планка збору коштів була виставлена на 100000000000000000000000000 одиниць.

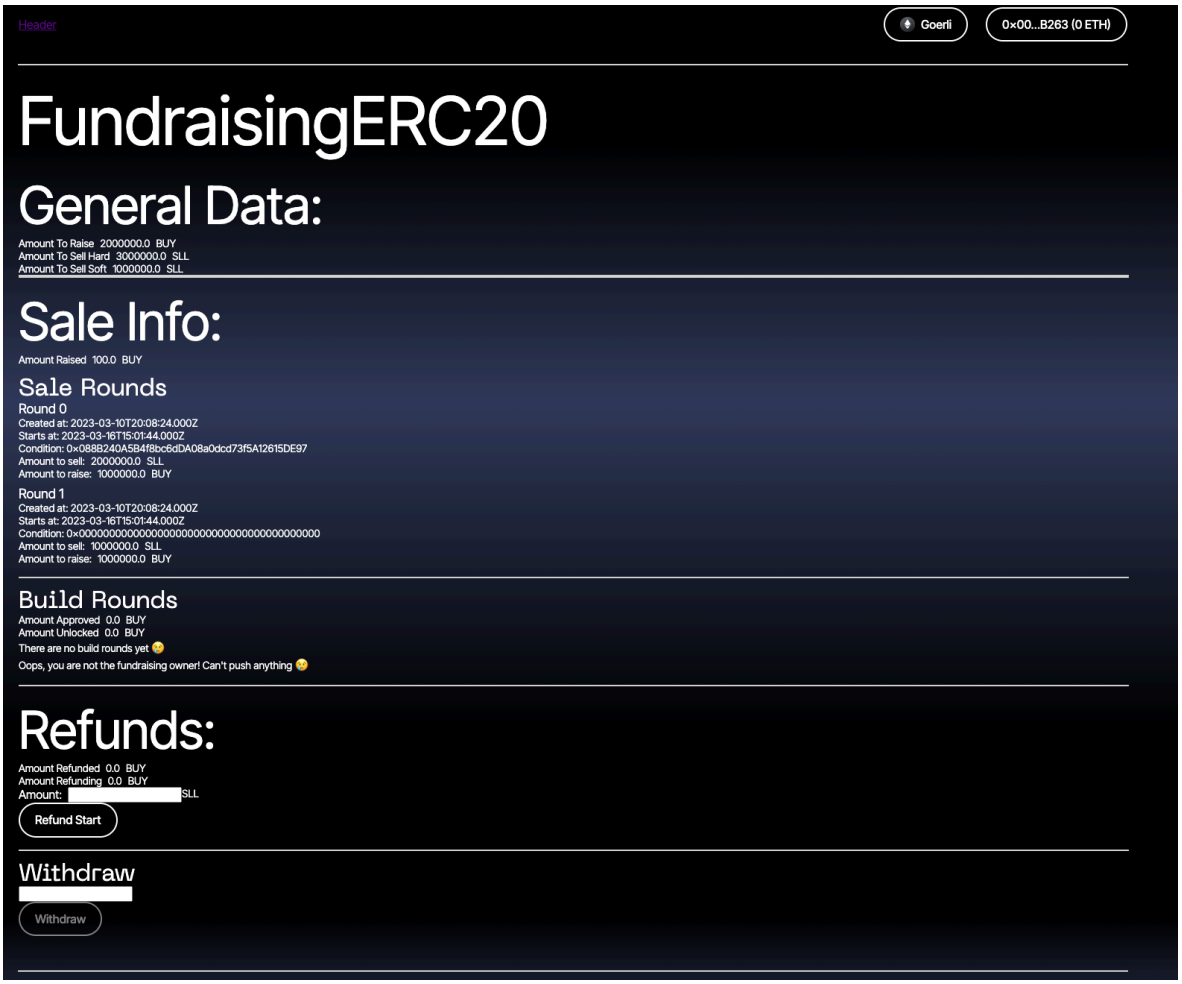


Рисунок 17. Деталі краудфандингу

Після створення краудфандингу ми бачимо сторінку з детальною інформацією про краудфандинг, створені посівні раунди, скільки вони зібрали коштів і при яких умовах. Також ми бачимо загальну суму всіх підтверджених коштів учасниками краудфандингу, і суму витрачених коштів. Окремо для краудфандера є меню виводу коштів з протоколу на власний гаманець.

Рисунок 18. Повернення коштів з краудфандингу

4.2 Порівняння результатів від протоколу з теоретичними розрахунками

Для порівняння результатів з теоретичними розрахунками була розроблена низка тестів, які перевіряють розроблені смарт-контракти на коректність. Нижче наведені логи, які були надані програмою Foundry:

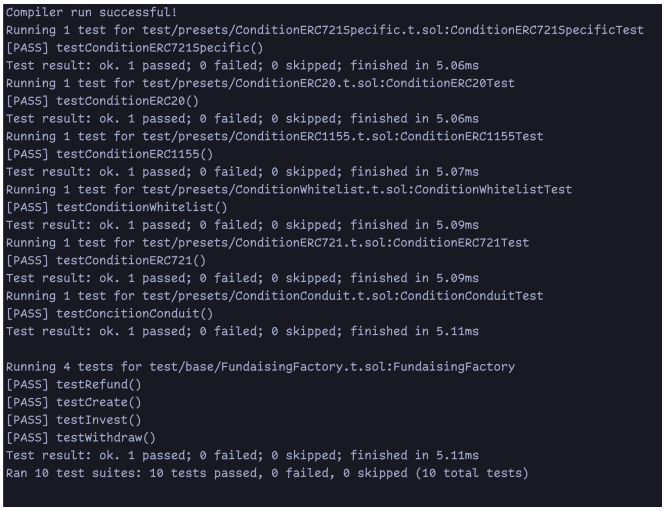


Рисунок 19. Логи програми прикладного тестування смарт-контракті “Foundry”

Висновки до розділу

Було імплементовано мінімальний веб додаток для ручного тестування платформи, та були імплементовані тести до окремих смарт контрактів. Теоретичні розрахунки повністю зівпали з практичними. Весь функціонал смарт контрактів було імплементовано та відвантажено на тестнет блокчейн Ethereum Goerli. Були наведені програмні логи тестів.

ВИСНОВКИ

В ході дослідження децентралізованих краудфандинг-протоколів, що було представлено у цій магістерській дисертації, було ретельно розглянуто сутність та сучасні досягнення у цій області. Вивчення принципово нових методів ітеративної витрати коштів, застосування Solidity та використання можливостей блокчейну Ethereum демонструє перспективність розробленого протоколу для реалізації децентралізованих фінансових операцій. Впровадження такого підходу може забезпечити більш високий рівень прозорості та контролю для учасників, одночасно знижуючи ризики, пов'язані з традиційними централізованими системами.

Програмне забезпечення, розроблене в рамках цієї дисертації, було піддано ретельному тестуванню та аналізу, результати якого підтвердили його надійність та ефективність. Порівняння емпіричних даних з теоретичними розрахунками вказує на високу відповідність і дає підстави стверджувати про доцільність подальшої роботи в цьому напрямку. Зауважимо, що знайдені обмеження сучасних методів і запропоновані засоби їх подолання вказують на шляхи удосконалення протоколу, зокрема, підвищення його масштабованості та вдосконалення механізмів безпеки.

Враховуючи зроблені спостереження та результати дослідження, можна констатувати, що запропонований протокол є вагомим внеском у розвиток децентралізованих фінансових систем і має значний потенціал для подальшої розробки та застосування. Сподіваємося, що розробка та впровадження даних інноваційних рішень стануть корисним внеском у сферу децентралізованих технологій і сприятимуть їх розвитку в напрямку забезпечення більшої безпеки, прозорості та ефективності.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Fabian Vogelsteller <fabian@ethereum.org>, Vitalik Buterin <vitalik.buterin@ethereum.org>, "ERC-20: Token Standard," Ethereum Improvement Proposals, no. 20, November 2015. [Online serial]. Available: https://eips.ethereum.org/EIPS/eip-20.
2. William Entriken, Dieter Shirley <dete@axiomzen.co>, Jacob Evans <jacob@dekz.net>, Nastassia Sachs <nastassia.sachs@protonmail.com>, "ERC-721: Non-Fungible Token Standard," Ethereum Improvement Proposals, no. 721, January 2018. [Online serial]. Available: https://eips.ethereum.org/EIPS/eip-721.
3. Wood, G. (n.d.). Ethereum: A secure decentralised generalised transaction ledger.
4. Bilash Saha, Md Mehedi Hasan, Nafisa Anjum, Sharaban Tahora, Aiasha Siddika, Hossain Shahriar, Protecting the Decentralized Future: An Exploration of Common Blockchain Attacks and their Countermeasures. Available: https://arxiv.org/pdf/2306.11884.pdf
5. Filip Johnson f@filip.world, Revnets. Available: https://github.com/rev-net/whitepaper/blob/master/whitepaper/revnet.pdf
6. Myles Lewis, Chris Crawford, Architectural Design for Secure Smart Contract Development. Available: https://arxiv.org/abs/2401.01891v1
7. Bruno Mazorra Roig, On the optimality of Shapley mechanism under Sybil strategies. Available https://arxiv.org/abs/2312.17058
8. Joshua Z. Tan, Tara Merk, Sarah Hubbard, Eliza R. Oak, Joni Pirovich, Ellie Rennie, Rolf Hoefer, Michael Zargham, Jason Potts, Chris Berg, Reuben Youngblom, Primavera De Filippi, Seth Frey, Jeff Strnad, Morshed Mannan, Kelsie Nabben, Silke Noa Elrifai, Jake Hartnell, Benjamin Mako Hill, Alexia Maddox, Woojin Lim, Tobin South, Ari Juels, Dan Boneh, Open Problems in DAOs. Available: https://arxiv.org/abs/2310.19201
9. Henry K. Dambanemuya, Eunseo Choi, Darren Gergle, Emőke-Ágnes Horvát, Hidden Influences of Crowd Behavior in Crowdfunding: An Experimental Study. Available: https://arxiv.org/abs/2206.07210

10. Hongze Liu, Jie Li, Shijing Yuan, Wenqi Cao, Bowen Li, A Smart Contract based Crowdfunding Mechanism for Hierarchical Federated Learning. Available: https://arxiv.org/abs/2205.06101
11. Beatrice Perez, Sara R. Machado, Jerone T. A. Andrews, Nicolas Kourtellis, I call BS: Fraud Detection in Crowdfunding Campaigns. Available: https://arxiv.org/abs/2006.16849
12. Mirko Zichichi, Michele Contu, Stefano Ferretti, Gabriele D'Angelo, LikeStarter: a Smart-contract based Social DAO for Crowdfunding. Available: https://arxiv.org/abs/1905.05560
13. Yilin Wang, Xiangping Chen, Yuan Huang, Hao-Nan Zhu, Jing Bian, Zibin Zheng, An Empirical Study on Real Bug Fixes from Solidity Smart Contract Projects. Available: https://arxiv.org/abs/2210.11990
14. Mirko Zichichi, Luca Serena, Stefano Ferretti, Gabriele D'Angelo, Governing Decentralized Complex Queries Through a DAO. Available: https://arxiv.org/abs/2107.06790
15. Lu Liu, Sicong Zhou, Huawei Huang, Zibin Zheng, From Technology to Society: An Overview of Blockchain-based DAO. Available: https://arxiv.org/abs/2011.14940
16. Lucas Martín Grasso Ramos (@LucasGrasso), Matias Arazi (@MatiArazi), "ERC-5516: Soulbound Multi-owner Tokens [DRAFT]," *Ethereum Improvement Proposals*, no. 5516, August 2022. [Online serial]. Available: https://eips.ethereum.org/EIPS/eip-5516.
17. Joshua Tan (@thelastjosh), Isaac Patka (@ipatka), Ido Gershtein <ido@daostack.io>, Eyal Eithcowich <eyal@deepdao.io>, Michael Zargham (@mzargham), Sam Furter (@nivida), "ERC-4824: Common Interfaces for DAOs [DRAFT]," *Ethereum Improvement Proposals*, no. 4824, February 2022. [Online serial]. Available: https://eips.ethereum.org/EIPS/eip-4824.

ДОДАТОК А

Презентація до магістерської роботи

“Децентралізований краудфандинг протокол для фінансових транзакцій на базі технології блокчейн”

Далечин В.О.
Науковий керівник: к.т.н., доц. Петрашенко А.В.

Актуальність тематики

На сьогодні краудфандинг є популярним способом залучення фінансування для різноманітних проектів. Проте існуючі централізовані платформи краудфандингу мають низку недоліків, таких як високі комісії, ризики шахрайства, відсутність прозорості та ін, а децентралізовані варіанти підлягають вектору атаки Sybil – створення декількох акаунтів в системі для отримання переваг.

Використання технології блокчейн дозволяє створити децентралізований краудфандинг-протокол, який усуває ці недоліки. Тому розробка такого протоколу є актуальною задачею.

Постановка задачі

- Проаналізувати існуючі рішення
- Визначити поточні перешкоди краудфандинг протоколів
- Визначити інструментарій для роботи над протоколом
- Визначити блокчейн на якому протокол оперуватиме
- Розробити протокол децентралізованого краудфандингу
- Протестувати розроблений протокол

Мета, об’єкт та предмет дослідження

Метою є розробка та реалізація децентралізованого краудфандинг-протоколу, який на основі технології блокчейн забезпечує безпеку, прозорість, приватність та ефективність фінансових транзакцій. Дослідження спрямоване на налагодження внутрішніх механізмів протоколу, забезпечення масштабованості та швидкості операцій, розробку алгоритмів консенсусу та захисту від шахрайства, а також дослідження можливостей для інтеграції з існуючими блокчейн-платформами.

Об'єктом є процес децентралізованого краудфандингу на основі технології блокчейн

Предметом є методи та засоби розробки децентралізованого краудфандинг-протоколу для фінансових транзакцій.

Розроблені підходи та алгоритми

Була розроблена математична модель ітеративної витрати коштів краудфандером. Основа протоколу полягає в наступній формулі:

$$amountAvailableToWithdraw = \sum_{i=1}^{n}(amountInvested_i - amountRefunded_i - amountFreezed_i)$$

де n - кількість інвесторів. Це враховує інвестиції (amountInvested[i], повернені кошти (amountRefunded[i]) та заморожені суми в очікуючі на можливе повернення (amountFreezed[i]).

Розроблений підхід полягає в ітеративній комунікації між краудфандером та інвесторами, де вони співпрацюють в довірливому середовищі так як успіх краудфандингу напряму залежить від інвесторів, які в будь-який момент можуть запросити повернення коштів.

Порівняльний аналіз

- Децентралізовані краудфандинг платформи: Paid Network, Seedify, Juicebox – працюють за принципами DAO (децентралізована автономна організація), де учасники вкладують кошти в загальний гаманець і разом вирішують куди вкласти гроші механізмом голосування, де великі гравці давлять на менших інвесторів маючи більшу силу голосування (у найбільшій децентралізованій біржі Uniswap, Binance володіє 5.9% голосів, а венчурний фонд Andersen Horrowitz володіє 6.7%.
  Протокол у свою ж чергу працює за механізмом “Кожний керує своїми коштами”, тому тиск на менших інвесторів зі сторони великих гравців відсутній.

Порівняльний аналіз

Запропонований протокол для краудфандингу коштів відрізняється від централізованих і децентралізованих конкурентів.

- Централізовані краудфандинг платформи: BSCPad, Dao Maker – потребують ліцензій, KYC (знай свого користувача) та KYB (знай свій бізнес) процедур і мають великі регуляторні ризики через централізоване зберігання залучених коштів та їх розподіл.
  Протокол у свою ж чергу децентралізований і не підлягає таким регуляторним ризикам через відкритість бізнес-логіки завдяки впровадженні технології блокчейн.

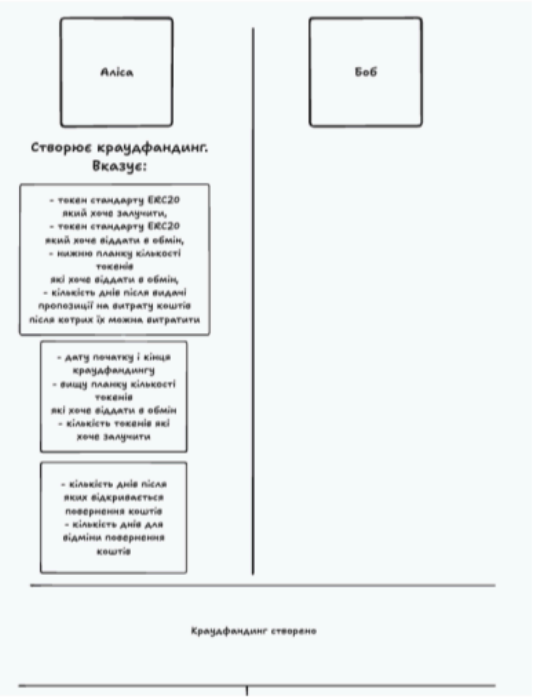
Одержані результати

Був розроблений алгоритм ітеративної витрати грошей, де є два актори – краудфандер і інвестор.

Розглянемо спрощену схему роботи протоколу.

Нехай краудфандером буде Аліса, а інвестором буде Боб.

Аліса вказує всю необхідну інформацію її краудфандингової кампанії та розпочинає її.



Боб інвестує в краудфандинг Аліси, після чого остання розпочинає першу ітерацію їх витрат.

Проходить час, і Аліса просить другий транш, проте Боб залишився не задоволеним результатом першого траншу, тому розпочинає процес повернення коштів.

Протокол імплементує захист від “FUD” (“Fear, Uncertainty, Doubt” – “Страх, невизначеність, жаль”), що є притаманним до блокчейн інвестицій через легкість і швидкість доступу до вкладених коштів, при яких інвестори під час надзвичайної події стримко виводять кошти з проектів спричиняючи хвилю поверненнь коштів іншими інвесторами розводячи цілеспрямовані психологічні атаки на менших інвесторів отримуючи з них фінансову вигоду маючи інсайдерську інформацію. Протокол заморожує кошти що підлягаються поверненню на вказаний краудфандером при створенні кампанії час, і якщо інвестор не відміняє процедуру, автоматично розблокуються для повернення інвестором.

Протокол був розроблений з урахуванням будь-яких побажань краудфандера, при цьому дотримуючись принципів відкритості.

Не всі краудфандери бажатимуть щоб їх інвестора мали доступ до функціоналу поверненню коштів, і для того щоб заблокувати їх повернення, досить просто вказати доволі велику затримку перед поверненням коштів після першої ітерації їх витрат, наприклад 100 років.

Такий параметр задається краудфандером під час створення кампанії.

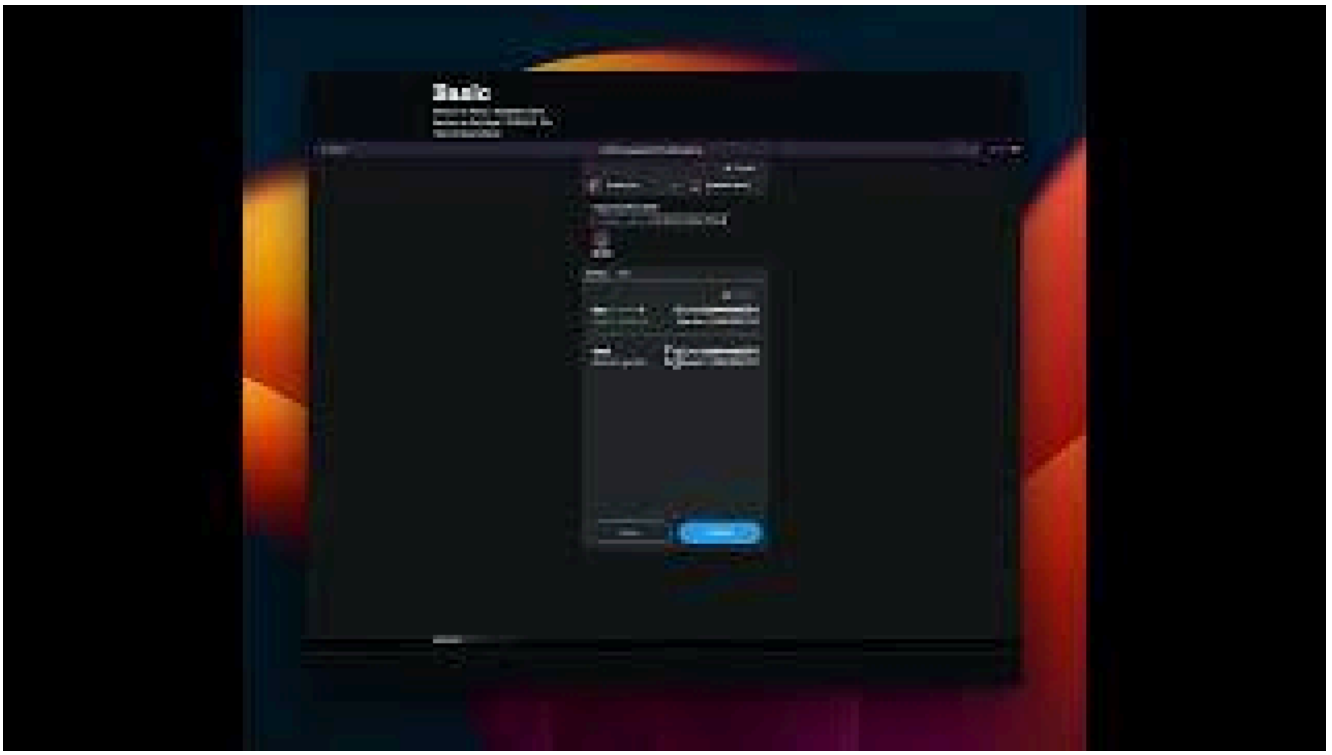
Також коригується затримка перед поверненням коштів, краудфандер може відрегулювати час “заморозки” коштів, що було наведено раніше як механізм захисту від FUD.

Якщо краудфандер хоче включити функціонал миттєвого повернненя коштів, він має вказати 0 днів як затримку повернення коштів.

В той же час, протокол має певні лімітації до виставлення такого повернення коштів, і краудфандер не може виставити більше ніж 60 днів як затримку повернення коштів, бо останній може таким чином в принципі заморозити кошти навічно.

В рамках тестування протоколу була розроблена мінімальний веб-додаток для комунікації з протоколом.

На наступному слайді з відео можна бачити процес підпису транзакції для її створення краудфандингу на базі токену ERC20 та їх відправка за допомогою некастодіального криптовалютного гаманця MetaMask, на тестовому блокчейні що був локально запущений.



На наступному відео ми бачимо процес інвестування коштів у краудфандинг з боку інвестора.

За стандартом токену ERC-20, виконувач транзакції має надати дозвіл децентралізованому протоколу, що є по суті сукупністю смарт-контрактів викладених на блокчейн, на витрату коштів зі сторони программного забезпечення (див. https://eips.ethereum.org/EIPS/eip-20#approval)

На наступному відео ми можемо бачити створення раунду витрат коштів зі сторони краудфандера, що запускає процес ітерації витрати коштів.

Краудфандер вказує сумму та додає замітку для свого траншу.

Публікації за темою магістерської дисертації:

Роботу було опубліковано в збірнику наукових матеріалів CXVIII Міжнародної інтернет —конференції «ЗИМОВІ НАУКОВІ ЧИТАННЯ — 2023», що відбулася 22 грудня 2023 року в м. Київ.

Також роботу було апробовано у XVI науково-практичній конференції магістрантів та аспірантів ПМК-2023 факультету прикладної математики.

Висновки

В процесі написання магістерської дисертації було:

- Проведено аналіз сучасних краудфандинг протоколів та виявлені їх проблеми.
- Проведено аналіз інструментів для розробки та аналіз блокчейну для розробки смарт-контрактів.
- Розроблений протокол, тестування та мінімальний веб додаток для взаємодії з ним.

ДОДАТОК Б

Лістинг програми.

```
/**
 * @title ICondition
 * @dev Interface for the Condition contract.
 */
interface ICondition {
    /**
     * @notice Checks if the condition is met.
     * @param _data The data needed to perform a check.
     * @return True if the condition is met, false otherwise.
     */
    function check(bytes memory _data) external view returns (bool);
}

/// @title Sale Stages
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
interface ISaleRounds {
    struct SaleRound {
        uint256 startsAt;
        uint256 endsAt;
        uint256 amountToRaise;
        uint256 amountToSell;
        ICondition condition;
    }

    function saleRounds(uint256 _index)
        external
        view
        returns (uint256 startsAt, uint256 endsAt, uint256 amountToRaise, uint256 amountToSell, ICondition condition);

    function saleRoundsCount() external view returns (uint256);

    function applicableSaleRoundIndexOf(bytes memory data) external view returns (bool, uint256);
}

interface IRefunds {
    error CannotCreateFundraisingIfAvailabilityDelayIsGreaterThan60DaysError();
    error CannotCreateFundraisingIfCancelWindowIsGreaterThan30DaysError();

    struct RefundConfiguration {
        uint32 cancelWindow; // max 30 days
        uint32 availabilityDelay; // max 60 days
    }

    function refundConfiguration() external view returns (uint32 cancelWindow, uint32 availabilityDelay);

    function isRefundAvailable() external view returns (bool);

    function refundStart(bytes calldata _data) external;
```

```
    function refundFinish() external;

    function refundCancel() external;
}

/// @title Vestable
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice Abstract contract for calculating claimable amounts with unlock
/// periods.
interface IVestable {
    struct Period {
        uint256 start;
        uint16 bps;
        bool cliffs;
    }

    error TotalBPSUnfulfilledError();
    error MinimumOfPeriodsNotMetError();
}

/// @title IFundraising01FactoryERC20Actions
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice Actions for factory contract for Fundraising01 contracts.
interface IFundraising01FactoryERC20Actions {
    /// @notice Creates a new sale.
    /// @param tokenToSell The token to sell.
    /// @param tokenToRaise The token to raise.
    /// @param amountToSellSoft The amount of tokens to sell soft (Soft Cap).
    /// @param saleRounds Sale Rounds.
    /// @param periods Vesting periods.
    /// @param refundConfiguration Refund configuration.
    /// @param executionDelay Refund configuration.
    /// @return The address of the sale.
    function createFundraisingERC20(
        address creator,
        IERC20 tokenToSell,
        IERC20 tokenToRaise,
        uint256 amountToSellSoft,
        ISaleRounds.SaleRound[] memory saleRounds,
        IVestable.Period[] memory periods,
        IRefunds.RefundConfiguration memory refundConfiguration,
        uint256 executionDelay
    ) external returns (address);
}

/// @title IFundraising01FactoryERC721Actions
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice Actions for factory contract for Fundraising01 contracts.
interface IFundraising01FactoryERC721Actions {
    /**
     * @notice Creates a new ERC721 sale contract.
     * @param tokenToSell The ERC721 token to be sold.
     * @param tokenToRaise The ERC20 token to be used to buy the ERC721 token.
```

```
     * @param amountToSellSoft The soft cap of the ERC721 token to be sold.
     * @param saleRounds Sale Rounds.
     * @param refundConfiguration Refund configuration.
     * @param executionDelay Refund configuration.
     */
    function createFundraisingERC721(
        address creator,
        IERC721Mintable tokenToSell,
        IERC20 tokenToRaise,
        uint256 amountToSellSoft,
        ISaleRounds.SaleRound[] memory saleRounds,
        IRefunds.RefundConfiguration memory refundConfiguration,
        uint256 executionDelay
    ) external returns (address);
}

/// @title IFundraising01FactoryActions
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice Actions for factory contract for Fundraising01 contracts.
interface IFundraising01FactoryActions {
    /// @notice Creates a new sale.
    /// @param tokenToSell The token to sell.
    /// @param tokenToRaise The token to raise.
    /// @param amountToSellSoft The amount of tokens to sell soft (Soft Cap).
    /// @param saleRounds Sale Rounds.
    /// @param periods Vesting periods.
    /// @param refundConfiguration Refund configuration.
    /// @param executionDelay Refund configuration.
    /// @return The address of the sale.
    function createFundraisingERC20(
        IERC20 tokenToSell,
        IERC20 tokenToRaise,
        uint256 amountToSellSoft,
        ISaleRounds.SaleRound[] memory saleRounds,
        IVestable.Period[] memory periods,
        IRefunds.RefundConfiguration memory refundConfiguration,
        uint256 executionDelay
    ) external returns (address);

    /**
     * @notice Creates a new ERC721 sale contract.
     * @param tokenToSell The ERC721 token to be sold.
     * @param tokenToRaise The ERC20 token to be used to buy the ERC721 token.
     * @param amountToSellSoft The soft cap of the ERC721 token to be sold.
     * @param saleRounds Sale Rounds.
     * @param refundConfiguration Refund configuration.
     * @param executionDelay Refund configuration.
     */
    function createFundraisingERC721(
        IERC721Mintable tokenToSell,
        IERC20 tokenToRaise,
        uint256 amountToSellSoft,
        ISaleRounds.SaleRound[] memory saleRounds,
```

```
        IRefunds.RefundConfiguration memory refundConfiguration,
        uint256 executionDelay
    ) external returns (address);
}

/// @title IFundraising01FactoryState
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice State of the factory that can change.
interface IFundraising01FactoryState {
    /// @notice All ongoing sales.
    /// @param index The index of the sale.
    /// @return The IFundraising01 sale contract.
    function sales(uint256 index) external view returns (address);
}

interface IWETH {
    function deposit() external payable;
    function transfer(address to, uint256 value) external returns (bool);
    function withdraw(uint256) external;
}

/// @title IFundraising01ERC20FactoryImmutableState
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice Immutable state of the Factory that will never change.
interface IFundraising01FactoryERC20ImmutableState {
    /// @notice The address of the WETH contract.
    /// @return The address of the WETH contract.
    function WETH() external view returns (IWETH);
}

/// @title IFundraising01ERC20FactoryEvents
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice Events emitted by factory.
interface IFundraising01FactoryERC20Events {
    event Fundraising01ERC20Created(address indexed fundraising, address indexed creator);
}

interface IFundraising01FactoryERC20Errors {
    error TransferFailedError();
}

/// @title IFundraising01FactoryERC20
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice This interface combines all interfaces for FactoryERC20.
interface IFundraising01FactoryERC20 is
    IFundraising01FactoryERC20ImmutableState,
    IFundraising01FactoryERC20Actions,
    IFundraising01FactoryERC20Events,
    IFundraising01FactoryERC20Errors
{}

/// @title IFundraising01ERC721FactoryImmutableState
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
```

```
/// @notice Immutable state of the Factory that will never change.
interface IFundraising01FactoryERC721ImmutableState {
    /// @notice The address of the WETH contract.
    /// @return The address of the WETH contract.
    function WETH() external view returns (IWETH);
}

/// @title IFundraising01ERC721FactoryEvents
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice Events emitted by factory.
interface IFundraising01FactoryERC721Events {
    event Fundraising01ERC721Created(address indexed fundraising, address indexed creator);
}

/// @title IFundraising01FactoryERC721
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice This interface combines all interfaces for FactoryERC721.
interface IFundraising01FactoryERC721 is
    IFundraising01FactoryERC721ImmutableState,
    IFundraising01FactoryERC721Actions,
    IFundraising01FactoryERC721Events
{}

/// @title IFundraising01ERC721FactoryImmutableState
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice Immutable state of the Factory that will never change.
interface IFundraising01FactoryImmutableState {
    /// @notice The address of the Fundraising Factory ERC20.
    /// @return The address of the Fundraising Factory ERC20.
    function factoryERC20() external view returns (IFundraising01FactoryERC20);

    /// @notice The address of the Fundraising Factory ERC721.
    /// @return The address of the Fundraising Factory ERC721.
    function factoryERC721() external view returns (IFundraising01FactoryERC721);
}

/// @title IFundraising01Factory
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice This interface combines all interfaces for Factory.
interface IFundraising01Factory is
    IFundraising01FactoryState,
    IFundraising01FactoryImmutableState,
    IFundraising01FactoryActions
{}

/// @title Build Rounds
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
interface IBuildRounds {
    event BuildingRoundPushed(uint256 index);

    error CannotPushRoundIfAnotherRoundIsPendingError();
    error CannotPushRoundBeforePreviousRoundIsExecutedError();
    error CannotPushRoundIfInsufficientReservesForProposalError(
```

```
        uint256 amountExceeds
    );

    struct BuildRound {
        uint256 amountAsked;
        string details;
        uint256 createdAt;
        uint256 amountUnlocked;
        bool isExecuted;
        bool isFulfilled;
    }

    /**
     * @notice Returns build round by index.
     * @param _index index.
     */
    function buildRounds(uint256 _index)
        external
        view
        returns (
            uint256 amountAsked,
            string memory details,
            uint256 createdAt,
            uint256 amountUnlocked,
            bool isExecuted,
            bool isFulfilled
        );

    /**
     * @notice Returns build rounds count.
     */
    function buildRoundsCount() external view returns (uint256);

    /// @notice Ask for unlock
    /// @dev Only the issuer can call this function.
    /// @param amount The amount of tokens to unlock.
    /// @param details The unlock details
    function pushBuildRound(uint256 amount, string memory details) external;

    function approveBuildRound() external;

    function hasPendingBuildRound() external view returns (bool);
}

/// @title IFundraising01BaseActions
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice Public Fundraising01 actions that anyone can call
interface IFundraising01BaseActions {
    /// @notice Invest in the sale.
    /// @param amount The amount of tokens to invest.
    function invest(uint256 amount, bytes memory data) external;

    /// @notice Forces balances to match reserves.
```

```
    /// @dev If some tokens were transferred directly to the contract, they can
    /// be claimed.
    function skim() external;
}

/// @title IFundraising01BaseConstants
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice State of the sale that will never change and encoded in the
/// contract.
interface IFundraising01BaseConstants {
    /// @notice The time period at which a sell should be up and running.
    /// If this period has lasted after the sale start, the sale will be
    /// destroyed.
    /// @return Start period.
    function SALE_STAGE_MIN_PERIOD() external pure returns (uint256);
}

/// @title IFundraising01BaseDerivedState
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice State of the sale that is computed on fly.
interface IFundraising01BaseDerivedState {
    /// @notice Returns available amount for claim.
    /// @return Claimable amount
    function claimable() external view returns (uint256);

    /// @notice Is the sale failed
    /// @return True if sale is failed
    function isFailed() external view returns (bool);
}

interface IFundraising01BaseErrors {
    error CannotCreateFundraisingIfFactoryIsZeroAddressError();

    error InsufficientTTLError();

    error CannotCreateFundraisingIfAmountToSellSoftIsZeroError();
    error CannotCreateFundraisingIfAmountToSellHardIsZeroError();
    error CannotCreateFundraisingIfAmountToRaiseIsZeroError();
    error CannotCreateFundraisingIfTokenToRaiseIsZeroAddressError();
    error CannotCreateFundraisingIfTokenToSellIsZeroAddressError();
    error CannotCreateFundraisingIfTokenToSellAndTokenToRaiseAreSameError();
    error TransferFailedError();

    error AmountCannotBeZeroError();

    error CannotInvestIfSaleRoundIsNotInProgressError();
    error SaleIsFailedError();
    error SaleIsNotFinishedError();

    error CannotUnlockIfDesiredUnlockAmountExceedsInvestedError(uint256 max);

    error CannotInvestIfInvestAmountExceedsRaiseHardAmountError(
        uint256 amountExceeds
```

```
    );
    error CannotInvestIfThereIsNoApplicableSaleRoundError();
    error CannotInvestWithETHError(address tokenToRaise);

    error CannotRefundStartIfRefundIsNotAvailableYetError();
    error CannotRefundYet(uint256 availableAt);

    error CannotRefundStartIfAmountToRefundExceedsAvailableToRefundError(
        uint256 amountExceeds
    );

    error CannotCancelRefundIfNotStartedError();
    error CannotCancelRefundIfWindowPassedError();

    error CannotExecuteRoundIfNoRoundIsPendingError();
    error CannotExecuteRoundIfDelayNotPassedYetError();
    error CannotApproveBuildRoundIfThereIsPendingProposalError();
    error CannotApproveBuildRoundIfProposalIsExecutedError();

    error CannotWithdrawZeroTokensError();
    error CannotWithdrawIfRoundIsFulfilledError();

    error ProposalToLiquidateIsInProgressError();

    error LiquidationIsNotAvailableYetError();

    error CannotClaimIfExceedsClaimableMaximumError();

    error CannotInvestIfSaleRoundConditionIsNotPassedError();
}

interface IFundraising01BaseEvents {
    event SaleFailed();
    event ApproveBuildRound(address indexed executor, uint256 index);

    event Invest(address indexed investor, uint256 amount);
    event Withdraw(uint256 amount);
    event Unlock(address indexed investor, uint256 amount);

    event SkimRaiseReserves(address indexed skimmer, uint256 reserveRaiseSkim);
}

/// @title IFundraising01BaseImmutableState
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice Immutable state of the sale that will never change.
interface IFundraising01BaseImmutableState {
    /// @notice Timestamp when the sale took place
    /// @return Timestamp
    function createdAt() external view returns (uint256);

    /// @notice Amount of tokens to raise
    /// @return Amount
    function amountToRaise() external view returns (uint256);
```

```
    /// @notice Amount of tokens to sell
    /// If sale doesn't sell the `amountToSellSoft` – it is considered as
    /// failed.
    /// @return Amount
    function amountToSellSoft() external view returns (uint256);

    /// @notice Amount of tokens to sell
    /// Fundraising cannot sell more than this amount.
    /// @return Amount
    function amountToSellHard() external view returns (uint256);

    /// @notice The token to raise
    /// @return The token to raise
    function tokenToRaise() external view returns (IERC20);
}

/// @title Owner Actions of Fundraising01
/// @notice Actions that can be performed by the owner of the sale.
interface IFundraising01BaseOwnerActions {
    function transferOwnership(address newOwner) external;

    /// @notice Withdraw available raised funds.
    /// @dev Only the owner can call this function.
    /// @param index Index of the unlock proposal.
    function withdraw(uint256 index) external;
}

/// @title IFundraising01BaseState
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice State of the sale that can change.
interface IFundraising01BaseState {
    /// @notice Raise Reserves
    /// @dev Used in skim() to equalize the tokens with real balance
    /// @return Raise Reserves
    function reservesRaise() external view returns (uint256);

    /// @notice Amount raised
    /// @return Amount raised
    function amountRaised() external view returns (uint256);
}

/// @title IFundraising01Base
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice Interface for Fundraising01Base contract.
interface IFundraising01Base is
    ISaleRounds,
    IBuildRounds,
    IFundraising01BaseActions,
    IFundraising01BaseConstants,
    IFundraising01BaseDerivedState,
    IFundraising01BaseErrors,
    IFundraising01BaseEvents,
```

```
    IFundraising01BaseImmutableState,
    IFundraising01BaseOwnerActions,
    IFundraising01BaseState
{}

/// @title IFundraising01ERC20State
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice State of the sale that can change.
interface IFundraising01ERC20State {
    /// @notice Vesting periods of the raise token
    /// @return start Start of the period
    /// @return bps BPS (basis points) of the total amount that will be unlocked
    /// @return cliffs Wether it cliffs to the next period or not
    function vestingPeriods(uint256 index) external view returns (uint256 start, uint16 bps, bool cliffs);

    /// @notice Amount of vesting periods
    /// @return length Amount of vesting periods
    function vestingPeriodsLength() external view returns (uint256 length);

    /// @notice Sell Reserves
    /// @dev Used in skim() to equalize the tokens with real balance
    /// @return Sell Reserves
    function reservesSell() external view returns (uint256);

    /// @notice Frozen tokens of the investor
    /// @dev Used to store the tokens that have been sent for a refund
    /// @param account Investor address
    /// @return Amount of frozen tokens
    function frozenTokensOf(address account) external view returns (uint256);
}

/// @title IFundraising01ERC20ImmutableState
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice Immutable state of the sale that will never change.
interface IFundraising01ERC20ImmutableState {
    /// @notice The token to sell
    /// @return The token to sell
    function tokenToSell() external view returns (IERC20);
}

interface IFundraising01ERC20Events {
    event SkimSellReserves(address indexed skimmer, uint256 reserveSellSkim);
    event Claim(address indexed investor, uint256 amount);
    event RefundStart(address indexed investor, uint256 amount);
    event RefundFinish(address indexed investor, uint256 amount);
    event RefundCancel(address indexed investor, uint256 amount);
}

interface IFundraising01ERC20Actions {
    /// @notice Claim available tokens.
    /// @param amountToClaim Amount To Claim
    function claim(uint256 amountToClaim) external;
}
```

```
/// @title IFundraising01
/// @notice A simple sale contract interface.
interface IFundraising01ERC20 is
    IFundraising01Base,
    IFundraising01ERC20Actions,
    IFundraising01ERC20ImmutableState,
    IFundraising01ERC20State,
    IFundraising01ERC20Events
{}

/// @title Vestable
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice Abstract contract for calculating claimable amounts with unlock
/// periods.
abstract contract Vestable is IVestable {
    uint32 constant TOTAL_BPS = 10_000;

    /// @dev Vesting periods
    Period[] internal _periods;

    constructor(Period[] memory periods_) {
        if (periods_.length < 2) revert MinimumOfPeriodsNotMetError();

        uint256 totalBps;
        for (uint256 i = 0; i < periods_.length; i++) {
            totalBps += periods_[i].bps;
            _periods.push(periods_[i]);
        }
        if (totalBps != TOTAL_BPS) revert TotalBPSUnfulfilledError();
    }

    /// @dev Calculates the releaseable amount for a given timestamp and
    /// deposited amount.
    function _available(uint256 timestamp, uint256 deposited) internal view virtual returns (uint256) {
        if (timestamp < _periods[0].start) return 0;

        uint256 availableBps;

        {
            uint256 i = 0;
            do {
                if (_periods[i].cliffs) {
                    availableBps += _periods[i].bps * (_periods[i + 1].start - timestamp)
                        / (_periods[i + 1].start - _periods[i].start);
                } else {
                    availableBps += _periods[i].bps;
                }
                i++;
            } while (i < _periods.length && timestamp < _periods[i + 1].start);
        }

        return deposited * availableBps / TOTAL_BPS;
```

```
    }
}

// OpenZeppelin Contracts (last updated v4.7.0) (access/Ownable.sol)

// OpenZeppelin Contracts v4.4.1 (utils/Context.sol)

/**
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes calldata) {
        return msg.data;
    }
}

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
abstract contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor() {
        _transferOwnership(_msgSender());
    }

    /**
     * @dev Throws if called by any account other than the owner.
```

```
     */
    modifier onlyOwner() {
        _checkOwner();
        _;
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view virtual returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if the sender is not the owner.
     */
    function _checkOwner() internal view virtual {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby disabling any functionality that is only available to the owner.
     */
    function renounceOwnership() public virtual onlyOwner {
        _transferOwnership(address(0));
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Internal function without access restriction.
     */
    function _transferOwnership(address newOwner) internal virtual {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
}

library AddressArrayUtils {
    function filter(address[] memory self, function (address) view returns (bool) predicate)
```

```
        internal
        view
        returns (address[] memory)
    {
        bool[] memory includeMap = new bool[](self.length);
        uint256 count = 0;
        for (uint256 i = 0; i < self.length; i++) {
            if (predicate(self[i])) {
                includeMap[i] = true;
                count++;
            }
        }
        address[] memory filtered = new address[](count);
        uint256 j = 0;
        for (uint256 i = 0; i < self.length; i++) {
            if (includeMap[i]) {
                filtered[j] = self[i];
                j++;
            }
        }
        return filtered;
    }

    function filteredLength(address[] memory self, function (address) view returns (bool) predicate)
        internal
        view
        returns (uint256)
    {
        uint256 count = 0;
        for (uint256 i = 0; i < self.length; i++) {
            if (predicate(self[i])) {
                count++;
            }
        }
        return count;
    }
}

/// @title Sale Stages
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
abstract contract SaleRounds is ISaleRounds {
    ISaleRounds.SaleRound[] public override saleRounds;

    constructor(ISaleRounds.SaleRound[] memory _saleRounds) {
        for (uint256 i = 0; i < _saleRounds.length; i++) {
            saleRounds.push(_saleRounds[i]);
        }
    }

    function saleRoundsCount() external view override returns (uint256) {
        return saleRounds.length;
    }
```

```
    function applicableSaleRoundIndexOf(bytes memory data) public view override returns (bool, uint256) {
        bool conditionFound = false;
        for (uint256 i = 0; i < saleRounds.length; i++) {
            if (
                saleRounds[i].startsAt <= block.timestamp && block.timestamp <= saleRounds[i].endsAt
                    && address(saleRounds[i].condition) != address(0)
            ) {
                conditionFound = true;
                if (saleRounds[i].condition.check(abi.encode(msg.sender, data))) {
                    return (true, i);
                }
            }
        }
        if (!conditionFound) {
            for (uint256 i = 0; i < saleRounds.length; i++) {
                if (saleRounds[i].startsAt <= block.timestamp && block.timestamp <= saleRounds[i].endsAt) {
                    return (true, i);
                }
            }
        }
        return (false, 0);
    }
}

/// @title Build Rounds
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
abstract contract BuildRounds is IBuildRounds {
    /// @inheritdoc IBuildRounds
    IBuildRounds.BuildRound[] public override buildRounds;
    uint256 public executionDelay;

    constructor(uint256 _executionDelay) {
        executionDelay = _executionDelay;
    }

    /// @inheritdoc IBuildRounds
    function buildRoundsCount() external view override returns (uint256) {
        return buildRounds.length;
    }

    function _beforePushBuildRound(uint256) internal virtual {
        if (hasPendingBuildRound()) {
            revert CannotPushRoundIfAnotherRoundIsPendingError();
        }

        /**
         * @notice buildRounds.length intended to be 1 as we're allowing
         * the first build round to be approved automatically.
         */
        if (
            buildRounds.length > 1
                && !buildRounds[buildRounds.length - 1].isExecuted
        ) {
```

```
            revert CannotPushRoundBeforePreviousRoundIsExecutedError();
        }
    }

    function _afterPushBuildRound(uint256 amount) internal virtual {}

    function pushBuildRound(uint256 amount, string memory details)
        external
        override
    {
        _beforePushBuildRound(amount);

        buildRounds.push(
            BuildRound({
                amountAsked: amount,
                amountUnlocked: 0,
                details: details,
                createdAt: block.timestamp,
                isFulfilled: false,
                isExecuted: false
            })
        );
        emit BuildingRoundPushed(buildRounds.length - 1);

        _afterPushBuildRound(amount);
    }

    function hasPendingBuildRound() public view override returns (bool) {
        return buildRounds.length > 0
            && block.timestamp
                <= buildRounds[buildRounds.length - 1].createdAt + executionDelay;
    }
}

abstract contract Refunds is IRefunds {
    IRefunds.RefundConfiguration public override refundConfiguration;

    constructor(IRefunds.RefundConfiguration memory _refundConfiguration) {
        if (_refundConfiguration.availabilityDelay > 60 days) {
            revert CannotCreateFundraisingIfAvailabilityDelayIsGreaterThan60DaysError();
        }

        if (_refundConfiguration.cancelWindow > 30 days) {
            revert CannotCreateFundraisingIfCancelWindowIsGreaterThan30DaysError();
        }
        refundConfiguration = _refundConfiguration;
    }

    function isRefundAvailable() public view virtual override returns (bool);
}

/// @title Fundraising01Base
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
```

```
/// @notice Third-party to make sale democratic.
abstract contract Fundraising01Base is
    Refunds,
    SaleRounds,
    BuildRounds,
    IFundraising01Base,
    Ownable
{
    /*.•°:°.+.*•´.*:.°.°*.°•´.°:°•.°•.*•´.*:.°.°*.°•´.°:°•.°+.*•´.*:.•°:°.´+°.*°.*/
    /*                                 LIBRARIES                                  */
    /*.•°:°.+.*•´.*:.°.°*.°•´.°:°•.°•.*•´.*:.°.°*.°•´.°:°•.°+.*•´.*:.•°:°.´+°.*°.*/

    using AddressArrayUtils for address[];

    /*.•°:°.+.*•´.*:.°.°*.°•´.°:°•.°•.*•´.*:.°.°*.°•´.°:°•.°+.*•´.*:.•°:°.´+°.*°.*/
    /*                                 CONSTANTS                                  */
    /*.•°:°.+.*•´.*:.°.°*.°•´.°:°•.°•.*•´.*:.°.°*.°•´.°:°•.°+.*•´.*:.•°:°.´+°.*°.*/

    /// @inheritdoc IFundraising01BaseConstants
    uint256 public constant SALE_STAGE_MIN_PERIOD = 30 days;

    /*.•°:°.+.*•´.*:.°.°*.°•´.°:°•.°•.*•´.*:.°.°*.°•´.°:°•.°+.*•´.*:.•°:°.´+°.*°.*/
    /*                                 IMMUTABLES                                 */
    /*.•°:°.+.*•´.*:.°.°*.°•´.°:°•.°•.*•´.*:.°.°*.°•´.°:°•.°+.*•´.*:.•°:°.´+°.*°.*/

    /// @inheritdoc IFundraising01BaseImmutableState
    uint256 public immutable override createdAt;
    /// @inheritdoc IFundraising01BaseImmutableState
    uint256 public immutable override amountToSellSoft;
    /// @inheritdoc IFundraising01BaseImmutableState
    IERC20 public immutable override tokenToRaise;

    /*.•°:°.+.*•´.*:.°.°*.°•´.°:°•.°•.*•´.*:.°.°*.°•´.°:°•.°+.*•´.*:.•°:°.´+°.*°.*/
    /*                                 MUTABLES                                   */
    /*.•°:°.+.*•´.*:.°.°*.°•´.°:°•.°•.*•´.*:.°.°*.°•´.°:°•.°+.*•´.*:.•°:°.´+°.*°.*/

    /// @inheritdoc IFundraising01BaseState
    uint256 public amountRaised;
    uint256 public amountApproved;
    uint256 public amountRefunded;
    uint256 public amountRefunding;

    /// @inheritdoc IFundraising01BaseState
    uint256 public override reservesRaise;

    struct InvestorData {
        uint256 amountDeposited;
        uint256 refundStartedAt;
        uint256 amountClaimed;
    }

    mapping(address => InvestorData) internal investorsData;
```

```
/*·.°•.°+.*•´.*:·.°*.°•´.°:°•.°•.*•´.*:·.°*.°•´.°:°•.°+.*•´.*:..•°:°.´+°.*°.*/
/*                          MODIFIERS                          */
/*·.°•.°+.*•´.*:·.°*.°•´.°:°•.°•.*•´.*:·.°*.°•´.°:°•.°+.*•´.*:..•°:°.´+°.*°.*/

modifier notFailed() {
    if (isFailed()) revert SaleIsFailedError();
    _;
}

/*·.°•.°+.*•´.*:·.°*.°•´.°:°•.°•.*•´.*:·.°*.°•´.°:°•.°+.*•´.*:..•°:°.´+°.*°.*/
/*                          CONSTRUCTOR                          */
/*·.°•.°+.*•´.*:·.°*.°•´.°:°•.°•.*•´.*:·.°*.°•´.°:°•.°+.*•´.*:..•°:°.´+°.*°.*/

constructor(
    IERC20 _tokenToRaise,
    uint256 _amountToSellSoft,
    ISaleRounds.SaleRound[] memory _saleRounds,
    RefundConfiguration memory _refundConfiguration,
    uint256 _executionDelay
)
    Ownable()
    Refunds(_refundConfiguration)
    SaleRounds(_saleRounds)
    BuildRounds(_executionDelay)
{
    if (address(_tokenToRaise) == address(0)) {
        revert CannotCreateFundraisingIfTokenToRaiseIsZeroAddressError();
    }
    if (_amountToSellSoft == 0) {
        revert CannotCreateFundraisingIfAmountToSellSoftIsZeroError();
    }

    createdAt = block.timestamp;

    tokenToRaise = _tokenToRaise;

    amountToSellSoft = _amountToSellSoft;
}

/*·.°•.°+.*•´.*:·.°*.°•´.°:°•.°•.*•´.*:·.°*.°•´.°:°•.°+.*•´.*:..•°:°.´+°.*°.*/
/*                          INVEST                          */
/*·.°•.°+.*•´.*:·.°*.°•´.°:°•.°•.*•´.*:·.°*.°•´.°:°•.°+.*•´.*:..•°:°.´+°.*°.*/

function _afterInvest(uint256 amount, uint256, bytes memory)
    internal
    virtual
{
    amountRaised += amount;
}

/// @inheritdoc IFundraising01BaseActions
function invest(uint256 amount, bytes memory _data)
    public
```

```
    virtual
    override
{
    _beforeInvest(amount);

    (bool matched, uint256 applicableSaleRoundIndex) =
        applicableSaleRoundIndexOf(abi.encodePacked(msg.sender, _data));
    if (!matched) revert CannotInvestIfThereIsNoApplicableSaleRoundError();

    _afterInvest(amount, applicableSaleRoundIndex, _data);
}

function _beforeInvest(uint256 amount) internal view notFailed {
    if (amount == 0) revert AmountCannotBeZeroError();
    if (block.timestamp > createdAt + SALE_STAGE_MIN_PERIOD) {
        revert CannotInvestIfSaleRoundIsNotInProgressError();
    }
    if (amountToRaise() < reservesRaise + amount) {
        revert CannotInvestIfInvestAmountExceedsRaiseHardAmountError(
            reservesRaise + amount - amountToRaise()
        );
    }
}

/*·.°•.°+.*•´.*:·.°*.°•´.°:°•.°•.*•´.*:·.°*.°•´.°:°•.°+.*•´.*:.•°:°.´+°.*°.*/
/*                          BUILD ROUNDS                          */
/*·.°•.°+.*•´.*:·.°*.°•´.°:°•.°•.*•´.*:·.°*.°•´.°:°•.°+.*•´.*:.•°:°.´+°.*°.*/

function _beforePushBuildRound(uint256 amount)
    internal
    override(BuildRounds)
    onlyOwner
    notFailed
{
    super._beforePushBuildRound(amount);

    // specific cheks for maximum unlock ask amount can be implemented below
    if (amount > reservesRaise) {
        revert CannotPushRoundIfInsufficientReservesForProposalError(
            amount - reservesRaise
        );
    }
}

function approveBuildRound() external override(IBuildRounds) {
    if (buildRounds.length > 1 && !hasPendingBuildRound()) {
        revert CannotApproveBuildRoundIfThereIsPendingProposalError();
    }
    BuildRound storage round = buildRounds[buildRounds.length - 1];
    if (round.isExecuted) {
        revert CannotApproveBuildRoundIfProposalIsExecutedError();
    }
```

```
    round.isExecuted = true;
    round.amountUnlocked = round.amountAsked;

    amountApproved += round.amountAsked;

    emit ApproveBuildRound(msg.sender, buildRounds.length - 1);
}

/*:°•.°+.*•´.*:˙.°*.˙•´.°:°•.°•.*•´.*:˙.°*.˙•´.°:°•.°+.*•´.*:.•°:°.´+˙.*°.*/
/*                          WITHDRAW                          */
/*:°•.°+.*•´.*:˙.°*.˙•´.°:°•.°•.*•´.*:˙.°*.˙•´.°:°•.°+.*•´.*:.•°:°.´+˙.*°.*/

/// @inheritdoc IFundraising01BaseOwnerActions
function withdraw(uint256 index) external {
    BuildRound storage round = buildRounds[index];

    if (round.amountUnlocked == 0) revert CannotWithdrawZeroTokensError();
    if (round.isFulfilled) revert CannotWithdrawIfRoundIsFulfilledError();

    round.isFulfilled = true;
    reservesRaise -= round.amountUnlocked;

    if (!tokenToRaise.transfer(owner(), round.amountUnlocked)) {
        revert TransferFailedError();
    }

    emit Withdraw(round.amountAsked);
}

/*:°•.°+.*•´.*:˙.°*.˙•´.°:°•.°•.*•´.*:˙.°*.˙•´.°:°•.°+.*•´.*:.•°:°.´+˙.*°.*/
/*                          REFUND                            */
/*:°•.°+.*•´.*:˙.°*.˙•´.°:°•.°•.*•´.*:˙.°*.˙•´.°:°•.°+.*•´.*:.•°:°.´+˙.*°.*/

function isRefundAvailable() public view override returns (bool) {
    // Refund becomes available either after two months of fundraising
    // living or after the first proposal
    return buildRounds.length > 1
        || createdAt + refundConfiguration.availabilityDelay > block.timestamp;
}

/**
 * @dev This function is called before a refund has been created.
 */
function _beforeRefundStart(bytes calldata) internal virtual {
    if (!isRefundAvailable()) {
        revert CannotRefundStartIfRefundIsNotAvailableYetError();
    }
}

/**
 * @dev This function is called after a refund has been created.
 */
```

```
function _afterRefundStart(bytes calldata _data) internal virtual {}

function refundStart(bytes calldata _data) external override {
    _beforeRefundStart(_data);
    investorsData[msg.sender].refundStartedAt = block.timestamp;
    _afterRefundStart(_data);
}

/**
 * @dev This function is called after a refund has been finished.
 */
function _afterRefundFinish() internal virtual {}

function refundFinish() external virtual override {
    bool failed = isFailed();
    if (
        !failed
            && block.timestamp
                < investorsData[msg.sender].refundStartedAt
                    + refundConfiguration.cancelWindow
    ) {
        revert CannotRefundYet(
            investorsData[msg.sender].refundStartedAt
                + refundConfiguration.cancelWindow
        );
    }
    _afterRefundFinish();
}

/**
 * @dev This function is called after a refund has been cancelled.
 */
function _afterRefundCancel() internal virtual;

function refundCancel() external override {
    if (investorsData[msg.sender].refundStartedAt == 0) {
        revert CannotCancelRefundIfNotStartedError();
    }
    if (
        block.timestamp
            > investorsData[msg.sender].refundStartedAt
                + refundConfiguration.cancelWindow
    ) {
        revert CannotCancelRefundIfWindowPassedError();
    }
    investorsData[msg.sender].refundStartedAt = 0;
    _afterRefundCancel();
}

/*·.°•.°+.••´.*:·.°*.°•´.°·.°•.°•.••´.*:·.°*.°•´.°·.°•.°+.•*´.*:.•°:.°.´+°.*°.*/
/*                              MISC                                */
/*·.°•.°+.••´.*:·.°*.°•´.°·.°•.°•.••´.*:·.°*.°•´.°·.°•.°+.•*´.*:.•°:.°.´+°.*°.*/
```

```
/// @inheritdoc Ownable
function transferOwnership(address newOwner)
    public
    virtual
    override(Ownable, IFundraising01BaseOwnerActions)
    onlyOwner
{
    super.transferOwnership(newOwner);
}

/// @inheritdoc IFundraising01BaseActions
function skim() public virtual override {
    uint256 reserveRaiseSkim =
        tokenToRaise.balanceOf(address(this)) - reservesRaise;

    if (reserveRaiseSkim > 0) {
        if (!tokenToRaise.transfer(msg.sender, reserveRaiseSkim)) {
            revert TransferFailedError();
        }
    }

    emit SkimRaiseReserves(msg.sender, reserveRaiseSkim);
}

/*:°•.°+.*•´.*:˚.°*.˚•´.°:°•.°•.*•´.*:˚.°*.˚•´.°:°•.°+.*•´.*:.•°:°.´+˚.*°.*/
/*                              GETTERS                              */
/*:°•.°+.*•´.*:˚.°*.˚•´.°:°•.°•.*•´.*:˚.°*.˚•´.°:°•.°+.*•´.*:.•°:°.´+˚.*°.*/

/// @inheritdoc IFundraising01BaseDerivedState
function claimable() public view virtual override returns (uint256) {
    return (investorsData[msg.sender].amountDeposited * amountToSellHard())
        / amountToRaise() - investorsData[msg.sender].amountClaimed;
}

function amountToRaise() public view returns (uint256 amount) {
    for (uint256 i = 0; i < saleRounds.length; ++i) {
        amount += saleRounds[i].amountToRaise;
    }
}

function amountToSellHard() public view returns (uint256 amount) {
    for (uint256 i = 0; i < saleRounds.length; ++i) {
        amount += saleRounds[i].amountToSell;
    }
}

function amountUnlocked() public view returns (uint256) {
    return amountApproved - amountRefunding - amountRefunded;
}

function depositedOf(address investor) public view returns (uint256) {
    return investorsData[investor].amountDeposited;
}
```

```
    function refundStartedAtOf(address investor)
        public
        view
        returns (uint256)
    {
        return investorsData[investor].refundStartedAt;
    }

    function unlockedOf(address investor) public view returns (uint256) {
        uint256 amount = amountUnlocked() * depositedOf(investor);
        if (amount == 0) return 0;
        return amount / amountRaised;
    }

    function leftToClaimOf(address investor) public view returns (uint256) {
        return depositedOf(investor) - unlockedOf(investor);
    }

    function availableToRefund(address investor)
        public
        view
        returns (uint256)
    {
        return (leftToClaimOf(investor) * amountToSellHard()) / amountToRaise();
    }

    /// @inheritdoc IFundraising01BaseDerivedState
    function isFailed() public view override returns (bool) {
        // this looks tricky, but the `reservesRaise` won't be updated if the
        // `SALE_STAGE_MIN_PERIOD` has passed and the `amountToSellSoft` was not
        // reached
        return block.timestamp > createdAt + SALE_STAGE_MIN_PERIOD
            && reservesRaise
                < (amountToSellSoft * amountToRaise()) / amountToSellHard();
    }
}

/// @title Fundraising01ERC20
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice Third-party to make sale democratic.
contract Fundraising01ERC20 is
    IFundraising01ERC20,
    Fundraising01Base,
    Vestable
{
    // ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
    // Immutables
    // ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
    /// @inheritdoc IFundraising01ERC20ImmutableState
    IERC20 public immutable override tokenToSell;
    // ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑
```

```
// ==============================================================================

// ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
// Reserves
// ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
/// @inheritdoc IFundraising01ERC20State
uint256 public override reservesSell;
// ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑
/// @inheritdoc IFundraising01ERC20State
mapping(address => uint256) public frozenTokensOf;

// ==============================================================================

constructor(
   IERC20 _tokenToSell,
   IERC20 _tokenToRaise,
   uint256 _amountToSellSoft,
   ISaleRounds.SaleRound[] memory _saleRounds,
   IVestable.Period[] memory _periods,
   IRefunds.RefundConfiguration memory _refundConfiguration,
   uint256 _executionDelay
)
   Fundraising01Base(
      _tokenToRaise,
      _amountToSellSoft,
      _saleRounds,
      _refundConfiguration,
      _executionDelay
   )
   Vestable(_periods)
{
   if (address(_tokenToSell) == address(0)) {
      revert CannotCreateFundraisingIfTokenToSellIsZeroAddressError();
   }
   if (address(_tokenToRaise) == address(_tokenToSell)) {
      revert
         CannotCreateFundraisingIfTokenToSellAndTokenToRaiseAreSameError();
   }

   reservesSell = amountToSellHard();
   tokenToSell = _tokenToSell;

   if (
      !tokenToSell.transferFrom(
         msg.sender, address(this), amountToSellHard()
      )
   ) {
      revert TransferFailedError();
   }
}

// ==============================================================================
```

```
// ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
// Invest Logic
// ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
function _afterInvest(
    uint256 amount,
    uint256 applicableSaleRoundIndex,
    bytes memory _data
) internal virtual override {
    super._afterInvest(amount, applicableSaleRoundIndex, _data);

    reservesRaise += amount;

    if (!tokenToRaise.transferFrom(msg.sender, address(this), amount)) {
        revert TransferFailedError();
    }

    investorsData[msg.sender].amountDeposited += amount;

    uint256 amountToSend = (
        amount * saleRounds[applicableSaleRoundIndex].amountToSell
    ) / saleRounds[applicableSaleRoundIndex].amountToRaise;

    reservesSell -= amountToSend;

    if (!tokenToSell.transfer(msg.sender, amountToSend)) {
        revert TransferFailedError();
    }

    emit Invest(msg.sender, amount);
}

// ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑

// ============================================================================

// ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
// Claim Logic
// ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
/// @inheritdoc IFundraising01ERC20Actions
function claim(uint256 amountToClaim) external override notFailed {
    uint256 maxAmountToClaim = claimable();
    if (amountToClaim > maxAmountToClaim) {
        revert CannotClaimIfExceedsClaimableMaximumError();
    }

    investorsData[msg.sender].amountClaimed += amountToClaim;

    if (!tokenToSell.transfer(msg.sender, amountToClaim)) {
        revert TransferFailedError();
    }

    emit Claim(msg.sender, amountToClaim);
    reservesSell -= amountToClaim;
```

```
}

// ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑

// ==========================================================================

// ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
// Refund Logic
// ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓

function _beforeRefundStart(bytes calldata _data)
    internal
    virtual
    override
{
    uint256 amountToRefund = abi.decode(_data, (uint256));
    {
        uint256 availableToRefund = availableToRefund(msg.sender);
        if (availableToRefund < amountToRefund) {
            revert
                CannotRefundStartIfAmountToRefundExceedsAvailableToRefundError(
                amountToRefund - availableToRefund
            );
        }
    }

    if (
        !tokenToSell.transferFrom(msg.sender, address(this), amountToRefund)
    ) {
        revert TransferFailedError();
    }

    reservesSell += amountToRefund;

    emit RefundStart(msg.sender, amountToRefund);
}

function _afterRefundStart(bytes calldata _data)
    internal
    virtual
    override
{
    frozenTokensOf[msg.sender] = (
        abi.decode(_data, (uint256)) * amountToRaise()
    ) / amountToSellHard();
}

function _afterRefundFinish() internal virtual override {
    uint256 amountToRefund = frozenTokensOf[msg.sender];
    investorsData[msg.sender].amountDeposited -= amountToRefund;
    reservesRaise -= amountToRefund;

    if (!tokenToRaise.transfer(msg.sender, amountToRefund)) {
```

```
        revert TransferFailedError();
    }

    emit RefundFinish(msg.sender, amountToRefund);
    delete frozenTokensOf[msg.sender];
}

function _afterRefundCancel() internal virtual override {
    investorsData[msg.sender].refundStartedAt = 0;

    if (
        !tokenToSell.transfer(
            msg.sender,
            (frozenTokensOf[msg.sender] * amountToSellHard())
                / amountToRaise()
        )
    ) {
        revert TransferFailedError();
    }

    emit RefundCancel(msg.sender, frozenTokensOf[msg.sender]);
    delete frozenTokensOf[msg.sender];
}

// ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑

// ===========================================================================

// ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
// Misc
// ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
/// @inheritdoc IFundraising01BaseActions
function skim()
    public
    virtual
    override(Fundraising01Base, IFundraising01BaseActions)
    notFailed
{
    uint256 reserveSellSkim =
        tokenToSell.balanceOf(address(this)) - reservesSell;

    if (reserveSellSkim > 0) {
        if (!tokenToSell.transfer(msg.sender, reserveSellSkim)) {
            revert TransferFailedError();
        }
    }

    super.skim();
    emit SkimSellReserves(msg.sender, reserveSellSkim);
}

// ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑
```

```
    // ============================================================================

    // ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
    // Getters
    // ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
    /// @inheritdoc IFundraising01ERC20State
    function vestingPeriods(uint256 index)
        external
        view
        override
        returns (uint256 start, uint16 bps, bool cliffs)
    {
        return
            (_periods[index].start, _periods[index].bps, _periods[index].cliffs);
    }

    /// @inheritdoc IFundraising01ERC20State
    function vestingPeriodsLength() external view returns (uint256) {
        return _periods.length;
    }

    /// @inheritdoc IFundraising01BaseDerivedState
    function claimable()
        public
        view
        override(Fundraising01Base, IFundraising01BaseDerivedState)
        returns (uint256)
    {
        return (
            super._available(block.timestamp, unlockedOf(msg.sender))
                * amountToSellHard()
        ) / amountToRaise() - investorsData[msg.sender].amountClaimed;
    }

    // ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑

    // ============================================================================

    /// @inheritdoc Ownable
    function transferOwnership(address newOwner)
        public
        virtual
        override(Fundraising01Base, IFundraising01BaseOwnerActions)
        onlyOwner
        notFailed
    {
        super.transferOwnership(newOwner);
    }
}
/// @title Fundraising01ERC721
/// @dev Opposed to Fundraising01ERC20, the NFTs are not being transferred in
/// construction,
/// but instead should be filled manually by calling `replenish(...)` function
```

```
/// @author Vladyslav Dalechyn <h0tw4t3r.eth>
/// @notice Third-party to make sale democratic.
contract Fundraising01ERC721 is IFundraising01ERC721, Fundraising01Base {
    // ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
    // Immutables
    // ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
    /// @inheritdoc IFundraising01ERC721ImmutableState
    IERC721Mintable public immutable override tokenToSell;

    // ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑

    // =======================================================================

    // ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
    // Mutables
    // ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
    /// @inheritdoc IFundraising01ERC721State
    uint256[] public override sellTokens;

    /// @inheritdoc IFundraising01ERC721State
    mapping(address => uint256[]) public tokensOf;
    /// @inheritdoc IFundraising01ERC721State
    mapping(address => uint256[]) public frozenTokensOf;
    // ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑

    // =======================================================================

    // ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
    // Reserves
    // ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
    /// @inheritdoc IFundraising01ERC721State
    uint256[] public override reservesSellTokens;
    // ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑

    using Uint256ArrayUtils for uint256[];

    // =======================================================================

    constructor(
        IERC721Mintable _tokenToSell,
        IERC20 _tokenToRaise,
        uint256 _amountToSellSoft,
        ISaleRounds.SaleRound[] memory _saleRounds,
        RefundConfiguration memory _refundConfiguration,
        uint256 _executionDelay
    )
        Fundraising01Base(
            _tokenToRaise,
            _amountToSellSoft,
            _saleRounds,
            _refundConfiguration,
            _executionDelay
        )
```

```
{
    if (address(_tokenToSell) == address(0)) {
        revert CannotCreateFundraisingIfTokenToSellIsZeroAddressError();
    }
    if (address(_tokenToRaise) == address(_tokenToSell)) {
        revert
            CannotCreateFundraisingIfTokenToSellAndTokenToRaiseAreSameError();
    }

    tokenToSell = _tokenToSell;
}

// ========================================================================

// ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
// Invest Logic
// ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
function _afterInvest(
    uint256 amount,
    uint256 applicableSaleRoundIndex,
    bytes memory _data
) internal virtual override {
    super._afterInvest(amount, applicableSaleRoundIndex, _data);

    uint256 pricePerToken = saleRounds[applicableSaleRoundIndex]
        .amountToRaise / saleRounds[applicableSaleRoundIndex].amountToSell;
    if (amount % pricePerToken != 0) {
        revert IncorrectInvestmentAmountError();
    }
    reservesRaise += amount;

    if (!tokenToRaise.transferFrom(msg.sender, address(this), amount)) {
        revert TransferFailedError();
    }

    for (uint256 i = 0; i < amount / pricePerToken; i++) {
        // uint256 randomToken = _random(msg.sender, i) %
        // amountToSellHardLocal;
        tokensOf[msg.sender].push(i);
        tokenToSell.mint(msg.sender, i);
    }

    investorsData[msg.sender].amountDeposited += amount;

    emit Invest(msg.sender, amount);
}

// ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑

// ========================================================================

// ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
// Claim Logic
```

```
// ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
// function _random(address to, uint256 nonce) internal view returns
// (uint256 num) {
//     num = uint256(
//         keccak256(
//             abi.encode(
//                 to,
//                 tx.gasprice,
//                 block.number,
//                 block.timestamp,
//                 block.prevrandao,
//                 blockhash(block.number - 1),
//                 address(this),
//                 nonce
//             )
//         )
//     );
// }
// ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑

// ============================================================================

// ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
// Refund Logic
// ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓

function _beforeRefundStart(bytes calldata _data)
    internal
    virtual
    override
{
    uint256[] memory tokenIds = abi.decode(_data, (uint256[]));
    {
        uint256 availableToRefund = availableToRefund(msg.sender);
        if (availableToRefund < tokenIds.length) {
            revert
                CannotRefundStartIfAmountToRefundExceedsAvailableToRefundError(
                tokenIds.length - availableToRefund
            );
        }
    }
    for (uint256 i = 0; i < tokenIds.length; i++) {
        tokenToSell.transferFrom(msg.sender, address(this), tokenIds[i]);
    }
}

function _afterRefundStart(bytes calldata _data)
    internal
    virtual
    override
{
    uint256[] memory tokensToFreeze = abi.decode(_data, (uint256[]));
    for (uint256 i = 0; i < tokensToFreeze.length; i++) {
```

```
            frozenTokensOf[msg.sender].push(tokensToFreeze[i]);
        }
        emit RefundStart(msg.sender, tokensToFreeze);
    }

    function _afterRefundFinish() internal virtual override {
        uint256 amountToRefund = (
            leftToClaimOf(msg.sender) * frozenTokensOf[msg.sender].length
        ) / tokensOf[msg.sender].length;
        investorsData[msg.sender].amountDeposited -= amountToRefund;
        reservesRaise -= amountToRefund;

        if (!tokenToRaise.transfer(msg.sender, amountToRefund)) {
            revert TransferFailedError();
        }

        for (uint256 i = 0; i < frozenTokensOf[msg.sender].length; i++) {
            reservesSellTokens.push(frozenTokensOf[msg.sender][i]);
            tokensOf[msg.sender].removeStorage(
                tokensOf[msg.sender].indexOf(frozenTokensOf[msg.sender][i])
            );
        }

        emit RefundFinish(msg.sender, frozenTokensOf[msg.sender]);
        delete frozenTokensOf[msg.sender];
    }

    function _afterRefundCancel() internal virtual override {
        investorsData[msg.sender].refundStartedAt = 0;
        for (uint256 i = 0; i < frozenTokensOf[msg.sender].length; i++) {
            tokenToSell.transferFrom(
                address(this), msg.sender, frozenTokensOf[msg.sender][i]
            );
        }

        emit RefundCancel(msg.sender, frozenTokensOf[msg.sender]);
        delete frozenTokensOf[msg.sender];
    }

    // ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑

    // ================================================================================

    function reservesSellTokensLength() public view returns (uint256) {
        return reservesSellTokens.length;
    }

    /// @inheritdoc Ownable
    function transferOwnership(address newOwner)
        public
        virtual
        override(Fundraising01Base, IFundraising01BaseOwnerActions)
        onlyOwner
```

```
        notFailed
    {
        super.transferOwnership(newOwner);
    }
}
```