МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»

ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«СИСТЕМА ЕЛЕКТРОННОГО ГОЛОСУВАННЯ З ВИКОРИСТАННЯМ БЛОКЧЕЙН-ТЕХНОЛОГІЙ»

Рівень «бакалавр»
Галузь знань 12 «Інформаційні технології»,
спеціальність 122 «Комп’ютерні науки»
Виконав здобувач 4 курсу
Дерев’ягін Владислав Сергійович

Керівник к.т.н., доцент
Трофименко Олена Григорівна

Рецензент к.т.н., доцент
Манаков Сергій Юрійович

Одеса – 2024

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
Факультет кібербезпеки та інформаційних технологій
Кафедра інформаційних технологій
Галузь знань 12 «Інформаційні технології»
Спеціальність 122 «Комп'ютерні науки»
Назва освітньої програми «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри інформаційних технологій
доцент Наталія Логінова _____
«___»_____2024 року

ЗАВДАННЯ
на кваліфікаційну (бакалаврську) роботу
здобувачу Дерев'ягіну Владиславу Сергійовичу

1. Тема роботи: «Система електронного голосування з використанням блокчейн-технологій»
   Керівник роботи: к.т.н, доцент Трофименко Олена Григорівна
   затверджені наказом НУ «ОЮА» від «02» квітня 2024 року № 809-06
2. Дата видачі завдання «15» вересня 2023 року
3. Строк подання здобувачем роботи «20» травня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Формування функціональних вимог	15.10.2023	
2	Аналіз та вибір засобів розробки	30.10.2023	
3	Розробка архітектури застосунку	20.11.2023	
4	Розробка смарт-контрактів	25.12.2023	
5	Розробка тестів	12.02.2024	
6	Розробка клієнтської частини	19.03.2024	
7	Аналіз ефективності запропонованого рішення	09.04.2024	
8	Оформлення звітної частини	20.04.2024	

Здобувач _____
(підпис) (прізвище та ініціали)

Керівник роботи _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дерев’ягін В. С. Система електронного голосування з використанням блокчейн-технологій. – Кваліфікаційна робота бакалавра за спеціальністю 122 “Комп’ютерні науки”, освітньою програмою “Комп’ютерні науки”.

Кваліфікаційна робота викладена на 63 сторінках основного тексту та 75 сторінках загального тексту, містить 3 розділи, 14 рисунків, 4 таблиці, 1 додаток, 26 використаних джерел.

Метою роботи є розробка системи електронного голосування, що забезпечує високий рівень безпеки, анонімність голосування та неможливість фальсифікації результатів за допомогою використання блокчейн-технологій.

Методи досліджень базуються на використанні концепцій блокчейн-технологій, криптографії, теорії алгоритмів та інформаційних систем.

Розроблено архітектуру системи електронного голосування на основі блокчейну, що включає механізми шифрування, автентифікації користувачів та забезпечення цілісності даних. Програмно реалізовано прототип системи, проведено експериментальні дослідження на модельних даних для перевірки ефективності та безпеки системи.

Результати роботи можуть бути використані для підвищення прозорості та надійності проведення виборів різного рівня. Можливими напрямками подальших досліджень є розширення функціональності системи, інтеграція з наявними виборчими системами та адаптація до міжнародних стандартів.

Ключові слова: блокчейн, електронне голосування, криптографія, автентифікація, цілісність даних, прозорість, надійність.

ABSTRACT

Dereviahin V. S. Electronic voting system using blockchain technologies. – Bachelor's qualification work in the specialty 122 "Computer Science", educational program "Computer Science".

The qualification work is presented on 63 pages of main text and 75 pages of general text, contains 3 sections, 14 figures, 4 tables, 1 appendice, and 26 references.

The aim of the work is to develop an electronic voting system that ensures a high level of security, anonymity of voting, and impossibility of falsifying results by using blockchain technology.

Research methods are based on the use of blockchain technology concepts, cryptography, algorithm theory, and information systems.

An architecture of the electronic voting system based on blockchain, which includes encryption mechanisms, user authentication, and data integrity assurance, has been developed. A prototype of the system has been implemented in software, and experimental studies on model data have been conducted to test the system's efficiency and security.

The results of the work can be used to increase the transparency and reliability of elections at various levels. Possible directions for further research include expanding the system's functionality, integration with existing electoral systems, and adaptation to international standards.

Key words: blockchain, electronic voting, cryptography, authentication, data integrity, transparency, reliability.

ЗМІСТ

ВСТУП

У сучасному світі, який невпинно рухається шляхом цифровізації всіх аспектів людського життя, важливість технологій у підтримці демократичних процесів, таких як вибори, стає все більшою. Системи електронного голосування, що обіцяють зробити виборчий процес більш доступним, ефективним і прозорим, здатні кардинально змінити спосіб, яким проводяться вибори. Проте, поряд з перевагами, які вони надають, є виклики, зокрема пов'язані з безпекою і недовірою з боку громадськості.

У контексті збільшення кількості кібератак та інцидентів, що ставлять під сумнів цілісність виборчих систем, актуальність розробки безпечних та надійних систем електронного голосування є вищою, ніж будь-коли. Зокрема, використання блокчейн-технологій у системах електронного голосування відкриває нові можливості для захисту від маніпуляцій, забезпечення прозорості та підвищення довіри до електронних виборів.

Метою роботи є створення програмної системи електронного голосування на основі Blockchain для забезпечення прозорості, безпеки і надійності виборчих процесів.

Об’єктом дослідження є системи електронного голосування на базі технології блокчейн.

Предмет дослідження – програмні засоби інтеграції блокчейн-технологій у системи електронного голосування для підвищення безпеки й ефективності виборчих процесів.

Для досягнення поставленої мети у роботі сформовано такі завдання:

– аналітичний огляд наявних рішень електронного голосування;

– вивчення принципів роботи та можливостей блокчейн-технологій, верифікації виборців та забезпечення анонімності голосування;

– визначення функціональних вимог розроблюваного рішення;

– програмна реалізація прототипу системи;

– тестування на вразливості та аналіз ефективності запропонованого рішення.

Для досягнення поставлених завдань використано комплексний підхід, що поєднує такі методи:

– *аналіз наявних рішень*: вивчення та порівняння наявних систем електронного голосування для визначення їхніх сильних і слабких сторін;

– *теоретичне дослідження*: вивчення принципів блокчейн-технологій та методів криптографії для забезпечення безпеки голосування;

– *програмування та реалізація* системи електронного голосування, включно з розробкою інтерфейсу користувача;

– *тестування*: написання та проведення автоматичних тестів для перевірки функціональності, безпеки та надійності системи;

– *аналіз результатів*: оцінка ефективності розробленої системи з урахуванням отриманих даних під час тестувань.

Наукова новизна одержаних результатів. Однією з ключових особливостей запропонованого рішення є інтеграція блокчейну з традиційними методами електронного голосування, що дозволяє забезпечити високий рівень безпеки та прозорості. Розроблено архітектуру системи, яка дозволяє залишатися анонімним та має механізми для унеможливлення зміни голосів після їх реєстрації. Новизна роботи полягає в створенні гнучкої системи, інтеграція якої буде доступна великому загалу.

Практичне значення отриманих результатів. Розроблена система має значний потенціал для впровадження у реальних виборчих процесах, починаючи від місцевих ініціатив і закінчуючи будь-якими проєктами, в яких важливо дізнатися думку громадськості. Прототип системи був успішно протестовано, що демонструє можливість її використання в реальних умовах.

Наукова публікація автора:

Дерев’ягін В.С. Розробка Telegram-бота для зберігання персоналізованих даних. *Інформаційне суспільство: проблеми та перспективи*: матер. VI всеукр. наук.-практ. конф. (м. Одеса, 28 травня 2021 р.). Одеса, 2021. С. 21-25.

1 АНАЛІЗ НАЯВНИХ БЛОКЧЕЙН-ТЕХНОЛОГІЙ

1.1 Основні принципи роботи блокчейну

Блокчейн є децентралізованою технологічною архітектурою, призначеною для забезпечення незмінності, безпеки та прозорості записів даних шляхом організації інформації у послідовність блоків [1]. Кожен блок у ланцюзі містить певну кількість транзакцій або інформаційних записів, що криптографічно пов'язані зі своїми попередниками, формуючи тим самим структуру, що виключає можливість несанкціонованих змін без виявлення. Ця структура не лише забезпечує цілісність та автентичність збереженої інформації, а й гарантує, що будь-які спроби модифікації даних в одному блоці потребуватимуть зміни у всіх наступних блоках, що є обчислювально нездійсненним без одночасного виявлення широкою спільнотою користувачів.

Блокчейн-технології беруть витоки свого розвитку з концепції "криптографічного ланцюга блоків", яка була вперше описана 1991 року С. Хабером і В. Сторнеттом, як метод захисту цифрових документів від підробки [2]. Проте справжній прорив та практичне застосування ця технологія отримала 2008 року, коли група осіб під псевдонімом Сатоши Накамото опублікували документ "Bitcoin: A Peer-to-Peer Electronic Cash System" [3]. У цьому документі описувалося створення децентралізованої валюти – Bitcoin, що функціонує на основі блокчейну. З цього моменту блокчейн почав набирати популярності не тільки як основа для криптовалют, а як технологія з потенціалом революціонізувати різні галузі.

Сьогодні блокчейн вважається перспективною технологією з потенціалом глибоко трансформувати не тільки фінансову індустрію, а й багато інших секторів, включаючи охорону здоров'я, логістику, нерухомість, право та урядові послуги, пропонуючи нові можливості для підвищення прозорості, безпеки та ефективності.

Основні концепції та визначення, які лежать в основі блокчейн-технологій, охоплюють широкий спектр термінології та ідей, критично важливих для розуміння їх принципів роботи та можливостей застосування.

Ключові поняття блокчейну [4]:

– *блок* – структурна одиниця в блокчейні, що містить інформацію або дані, наприклад, транзакції. Кожен блок має унікальний геш попереднього блока, що створює ланцюжок блоків і забезпечує незмінність історії записів;

– *транзакція* – переказ даних або цінностей між учасниками мережі блокчейну. Після верифікації транзакція додається до нового блока в ланцюгу;

– *консенсус* – механізм досягнення угоди між усіма вузлами мережі блокчейну щодо поточного стану бази даних. Є алгоритми консенсусу Proof of Work (PoW) та Proof of Stake (PoS), кожен з яких має свої особливості та призначення;

– *гешування* – процес перетворення вхідних даних будь-якого розміру на унікальний фіксований розмір бітового рядка за допомогою геш-функції. Геші використовуються для забезпечення цілісності даних в блокчейні;

– *смарт-контракт* – програма, що автоматично виконує, контролює або документує значимі події та дії згідно з умовами контракту, код якого розміщено на блокчейні;

– *децентралізація* – принцип організації мережі, за якого керування та обробка даних розподілені між всіма учасниками без потреби в центральному регулюючому органі.

Блок є основною структурною одиницею, яка містить інформацію про одну або декілька транзакцій. Структура блока розроблена так, щоб забезпечити безпеку та незмінність даних всередині мережі. Основні елементи блока:

– *ідентифікатор* – унікальний ідентифікатор, який зазвичай подається у вигляді гешу блока. Цей геш генерується з використанням криптографічного алгоритму, який приймає вхідні дані блока як аргумент і виробляє байтову

послідовність фіксованої довжини. Ідентифікатор слугує як унікальний відбиток блока, який може бути використаний для його верифікації та відшукання в ланцюжку;

– *геш попереднього блока* – кожен блок містить геш попереднього блока в ланцюжку, що створює неперервний зв'язок між блоками. Ця характеристика забезпечує цілісність блокчейну, оскільки будь-яка спроба змінити інформацію в одному блоці потребуватиме перерахунку гешів у всіх подальших блоках, що обчислювально важко реалізувати;

– *часова мітка (Timestamp)* – вказує час створення блока. Часові мітки використовуються для відстеження послідовності створення блоків та допомагають у підтримці прозорості та аудиту в мережі блокчейну;

– *список транзакцій* містить всі транзакції, які були здійснені та підтверджені в мережі блокчейну до моменту створення блока. Кожна транзакція в цьому списку містить інформацію про відправника, отримувача, суму переказу та інші відомості. Список транзакцій у блоці забезпечує детальний запис всіх операцій, що були здійснені, і є ключовим елементом для забезпечення прозорості та верифікованості даних у блокчейні.

Ці компоненти разом формують блок, який після створення додається до ланцюга блоків. Кожен новий блок міцно пов'язаний із попередніми, створюючи незмінний та безпечний запис історії транзакцій.

Криптографічні алгоритми є фундаментальною частиною технології блокчейн, вони забезпечують цілісність даних у розподіленій мережі. Ці алгоритми використовуються для різних аспектів блокчейну, включаючи гешування, шифрування та цифрові підписи, що дозволяє учасникам обмінюватися інформацією в надійний та верифікований спосіб.

Асиметричне шифрування, також відоме як криптографія з відкритим ключем, є іншим критично важливим аспектом безпеки блокчейну. Воно використовує пару ключів: публічний ключ, який може бути вільно розповсюджений та використовуватися для шифрування повідомлень, та приватний ключ, що зберігається

в таємниці та використовується для розшифровування. Асиметричне шифрування дозволяє учасникам блокчейну безпечно обмінюватися повідомленнями та верифікувати транзакції через механізм цифрових підписів, не розкриваючи своїх приватних ключів.

Цифрові підписи використовують асиметричне шифрування для верифікації авторства та інтегральності повідомлень або транзакцій. Вони генеруються за допомогою приватного ключа відправника та можуть бути перевірені будь-ким, хто має публічний ключ відправника. Це забезпечує незмінність інформації після її підписання та достовірність її походження від заявленого відправника.

Завдяки цим криптографічним алгоритмам, блокчейн-технології забезпечують високий рівень безпеки, прозорості та надійності, що робить їх ідеально придатними для широкого спектра застосувань, від фінансових транзакцій до систем голосування та за межами.

Інформація, записана в блокчейн, не може бути змінена без зміни всіх подальших блоків і досягнення консенсусу в мережі, що робить дані в блокчейні незмінними. Ця незмінність є важливою для забезпечення цілісності даних і довіри до системи, оскільки користувачі можуть бути впевнені, що записи ніколи не будуть змінені або видалені. Це не тільки забезпечує високий рівень безпеки та довіри між учасниками мережі, а й відіграє ключову роль у вирішенні цілого ряду проблем, пов'язаних із цифровими транзакціями та обміном даними.

Переваги такого підходу [5]:

– *протидія шахрайству*: оскільки записи не можуть бути змінені ретроспективно без виявлення, це значно ускладнює будь-які спроби шахрайства або маніпуляції даними. Це особливо важливо для фінансових операцій, де непохитність історії транзакцій є критичною;

– *аудит і верифікація*: іммутабельність блокчейну сприяє легкості проведення аудиту та верифікації транзакцій. Кожен учасник мережі може перевірити історію транзакцій, що забезпечує прозорість і підтримує довіру;

– *захист від втрати даних*: у розподіленій мережі кожен вузол зберігає копію всієї історії транзакцій, що мінімізує ризик втрати даних через технічні збої або навмисні атаки.

Зважаючи на специфіку та переваги, блокчейн можна використовувати для створення безпечних та захищених від несанкціонованого доступу цифрових ідентифікаційних даних. Ця технологія допомагає захистити особисті дані від крадіжки. Одною з багатьох сфер застосування блокчейну є системи електронного голосування. Завдяки захищеності та прозорості налаштувань для відбору, запису та підрахунку голосів виборців, блокчейн може потенційно усунути фальсифікації виборів і підвищити явку виборців.

Технологія блокчейн уможливила новий вид розподілених систем. Окрім свого раннього застосування у фінансах, вона сприяла появі нових способів управління та координації – так званих децентралізованих автономних організацій (Decentralized Autonomous Organization або DAO) [6]. Переважно DAO впроваджують системи прийняття рішень, щоб дозволити їх онлайн-спільноті досягати угод.

Архітектуру управління DAO можна описати як трикутник між розподіленим реєстром, смарт-контрактами та механізмами голосування (рис. 1.1) [7]. Смарт-контракти відповідають за впровадження всіх правил і контрактів у DAO, що стосуються процедур у механізмах голосування. З іншого боку голосування є єдиним формальним способом змінити правила, визначені смарт-контрактами, і додати нові правила до смарт-контрактів. Смарт-контракти та механізми голосування не тільки спільно формують управління в DAO, а й спільно обмежують один одного, вони обидва мають вирішальне значення для нормальної роботи DAO.

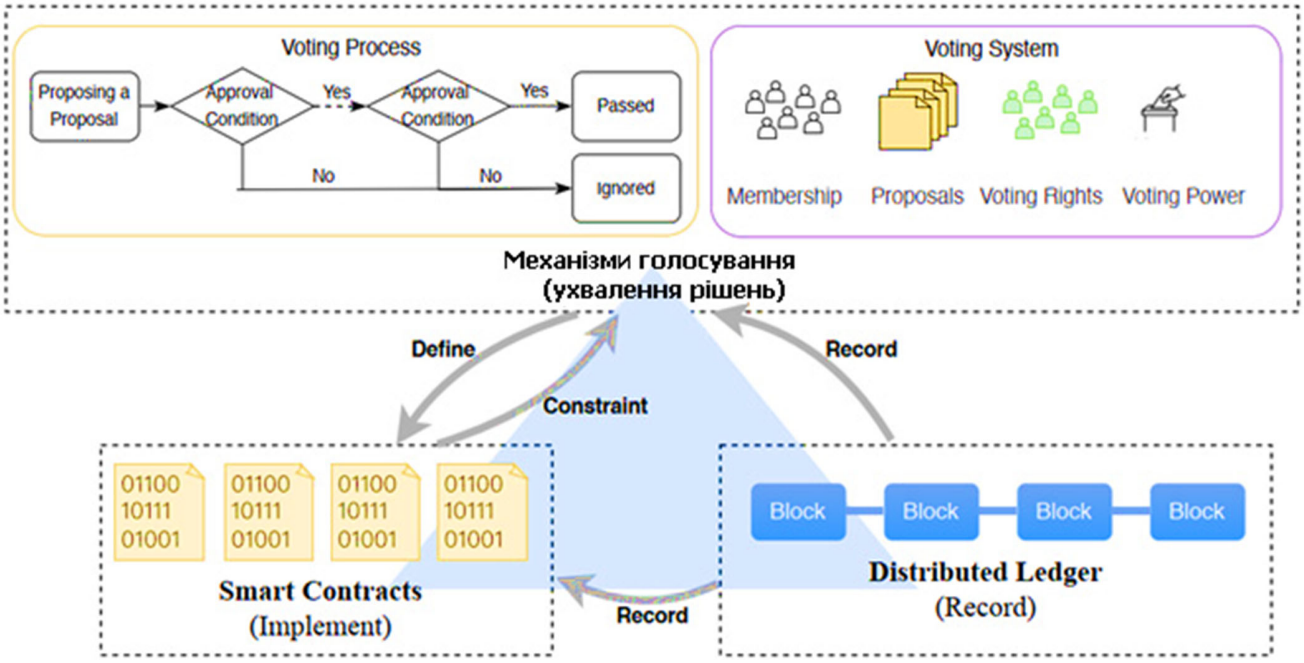

Рисунок 1.1 – Зв'язок між розподіленим реєстром, смарт-контрактами та механізмами голосування в DAO

Критичні дії, які виконуються механізмами голосування та смарт-контрактами, реєструються в базі даних розподіленого реєстру. Розподілений реєстр не тільки відповідає вимогам децентралізації, а й забезпечує послідовні, незмінні записи. Взаємодія та обмеження між розподіленим реєстром, смарт-контрактами та механізмами голосування утворюють стабільну систему управління для DAO.

1.2 Специфіка систем електронного голосування на базі Blockchain

Системи електронного голосування, засновані на блокчейнах, використовують різні концепції та технології для забезпечення безпечних і надійних виборів. Серед цих технологій — мережі блокчейну Ethereum та Hyperledger Fabric, консенсусні алгоритми Proof of Work, Proof of Stake і Practical Byzantine Fault Tolerance, а також методи підвищення конфіденційності, наприклад, гомоморфне шифрування та докази з нульовим знанням [8]. Крім того, механізми автентифікації, наприклад, біометрична перевірка та системи керування ідентифікацією, мають вирішальне значення для

підтвердження легітимності виборця та підтримки цілісності системи голосування.

Перевагами систем електронного голосування на блокчейні Ethereum є [9]:

– *децентралізація керування та валідації:* система не залежить від централізованого контролю, що знижує ризики маніпуляцій результатами голосування з боку окремих осіб або організацій;

– *автоматизація процесів:* усі етапи виборчого процесу, від реєстрації до підрахунку голосів, автоматизовані, що прибирає можливість людської помилки та підвищує ефективність системи;

– *незалежність від сторонніх сервісів:* система використовує блокчейн і смарт-контракти для всіх операцій, мінімізуючи потребу в зовнішніх системах;

– *захист приватності і анонімність:* система використовує криптографічні методи для захисту ідентичності виборців, що забезпечує високий рівень конфіденційності. Ідентичність голосуючих залишається анонімною навіть для публічних транзакцій;

– *прозорість і верифікованість:* весь код є публічним і постійним, що дозволяє будь-кому перевірити правильність виборчих правил, а голоси зберігаються в блокчейні, що гарантує незмінність результатів;

– одноразове *голосування:* система запобігає можливості подвійного голосування за допомогою прив'язки до мобільного номера.

Недоліки запропонованого підходу [10]:

– *відсутність гнучкості у визначенні права голосу*: умови, за якими користувач може брати участь у голосуванні, жорстко задані в смарт-контракті без можливості легкої модифікації або адаптації під змінні умови або вимоги;

– *залежність від централізованого сервісу для SMS автентифікації*: хоча система голосування є децентралізованою, вона все ще покладається на централізований SMS-шлюз для доставки SMS при аутентифікації користувачів; це створює потенційну точку відмови та можливість для маніпуляцій або зовнішнього втручання у процес аутентифікації;

– *складність процесу реєстрації користувачів:* процес реєстрації, що включає введення мобільного номера, генерацію і перевірку PIN за допомогою SMS, може бути складним і займати більше часу, що потенційно знижує зручність користувачів і може відштовхнути їх від участі у голосуванні; такий підхід може не підходити загальній аудиторії через те, що власнику необхідно оплачувати кожне SMS повідомлення;

– *відсутність механізму для відновлення доступу:* у разі втрати доступу до мобільного номера або помилкової реєстрації, в системі не передбачено механізму відновлення доступу або перереєстрації, що може призвести до неможливості голосування для деяких користувачів;

– *складність зміни логіки контракту:* логіка смарт-контракту, яка управляє процесом голосування, є незмінною після його розгортання, будь-які помилки або потреби в модифікації вимагатимуть розгортання нового контракту;

– *потенційна можливість для заблокування учасників власником* голосування: система дозволяє адміністраторові події блокувати учасників (припиняти їх можливість голосувати), що може бути використано для маніпуляції результатами голосування або несправедливого обмеження участі користувачів;

– *необхідність біометричної автентифікації*: система вимагає від користувачів проходження біометричної перевірки для ідентифікації, що може стати перешкодою для анонімності та викликати питання щодо приватності, а також таке рішення не підходить для широкої аудиторії, наприклад, при створенні стартапу або інді-студії для розробки ігор.

1.3 Аналіз платформ електронного голосування на основі блокчейну

Децентралізована природа блокчейну вирішує проблеми, подібні до тих, які є в електронному голосуванні, як-от забезпечення безпеки та масштабованості. За останні роки з'явилося немало децентралізованих платформ для електронного голосування, які прагнуть використовувати технологію блокчейн для забезпечення прозорості, безпеки та надійності виборчих процесів. Відомими прикладами таких платформ є

Aragon, DAOstack, DAOhaus і Voatz, кожна з яких має свої унікальні особливості щодо зростання та активності, а також щодо результатів голосування.

1. **Aragon** (https://aragon.org/) є відкритою платформою, призначеною для створення та керування децентралізованими автономними організаціями (DAO). Aragon є найбільшою платформою DAO [11].

Таблиця 1.1 – SWOT-аналіз платформи Aragon

Сильні сторони	Слабкі сторони
– використовує смарт-контракти для автоматизації управлінських процесів, як-от: голосування, фінансування й керування активами, забезпечуючи прозорість та ефективність у прийнятті колективних рішень; – проєкт спрямований на розширення можливостей децентралізації і дозволяє будь-кому заснувати та керувати DAO з будь-якої точки світу; – завдяки платформі, організації можуть ефективно взаємодіяти, розподіляти ресурси, ухвалювати стратегічні рішення та управляти проєктами без потреби в складних юридичних процедурах або посередниках; – модульна архітектура, яка дозволяє користувачам налаштовувати свої DAO згідно з потребами та цілями; – система керування правами доступу на основі смарт-контрактів дозволяє детально налаштовувати рівні доступу та дозволи для різних учасників, забезпечуючи тим самим ефективне та безпечне керування ресурсами організації; – кожна транзакція, голосування реєструється на блокчейні, забезпечуючи повну прозорість та можливість аудиту для всіх учасників.	– комплексність використання: для ефективного використання та створення DAO потрібні певні передові технічні знання та розуміння принципів функціонування автономних організацій. Це може становити бар'єр для нових користувачів або компаній, які не мають висококваліфікованих розробників; – керування спільнотою: хоча децентралізоване керування є однією з ключових переваг DAO, воно також може ускладнювати швидке прийняття рішень. Консенсус у великих спільнотах може бути важко досягнути, що може призвести до затримок у впровадженні необхідних змін або виконанні проєктів.
Можливості	**Загрози**
– Aragon надає статичний шаблон для створення власного DAO, але також дозволяє створити індивідуальний; – відкриває нові горизонти для колективного керування, де традиційні ієрархічні структури замінюються на більш гнучкі та демократичні моделі взаємодії	– для легітимності голосування в системі є дві умови: 1) відсоток позитивних голосів від усіх поданих голосів має бути не менше необхідного відсотка підтримки; 2) мінімальний кворум прийняття визначає мінімальний відсоток поданих голосів від усіх можливих голосів DAO.

Aragon надає інструменти для запуску та керування організаціями, які функціонують на основі блокчейну, без потреби централізованого контролю. Платформа Aragon прагне забезпечити високий рівень безпеки та стабільності для DAO, впроваджуючи передові практики та технології блокчейну. Aragon надає необхідні інструменти та інфраструктуру для реалізації будь-якої концепції DAO [12]. Використання блокчейну гарантує прозорість у прийнятті рішень та веденні фінансової діяльності DAO. Aragon продовжує розвиватися та вдосконалюватися, інтегруючи нові функції та можливості, щоб задовольнити зростаючі потреби децентралізованих організацій.

2. **DAOstack** — це комплексний фреймворк для спрощення створення та керування децентралізованими автономними організаціями (DAO) у блокчейні Ethereum [13].

Таблиця 1.2 – SWOT-аналіз платформи DAOstack

Сильні сторони	Слабкі сторони
– використовує набір інструментів, які дозволяють колективам ухвалювати рішення без централізованого контролю; – ключовою особливістю є використання алгоритмічного керування, що дозволяє спільноті гнучко налаштовувати правила голосування та вирішувати, як ресурси мають бути розподілені; – надає можливість розробляти децентралізовані застосунки, інтегровані з різними DAO для керування ресурсами, проєктами та ініціативами	– Комплексність керування. Однією з головних проблем DAOstack є складність налаштування та керування DAO, особливо для неспеціалістів у сфері блокчейну. Система вимагає глибокого розуміння її архітектури та компонентів, що може становити бар'єр для широкого впровадження
Можливості	**Загрози**
– оскільки для DAOstack потрібна або абсолютна більшість (51%), або достатня кількість ставок, щоб пропозиція була «розширена» і, тим самим, була схвалена відносною більшістю, вона показує меншу кількість схвалених пропозицій; – система прагне розв'язати проблему "колективного розуму" за допомогою смарт-контрактів, які спрощують процес прийняття рішень великими групами людей	– масштабування. У міру зростання DAO, забезпечення масштабування процесів участі та голосування без втрати ефективності стає важливим викликом. Більша кількість учасників може ускладнити швидке та ефективне прийняття рішень

Фреймворк DAOstack розроблений з урахуванням масштабованості, надає набір модульних і сумісних смарт-контрактів, які можна комбінувати для створення налаштованих DAO. Платформа пропонує низку функцій управління, включаючи власний токен (GEN) для прийняття рішень, має простий у використанні інтерфейс для керування пропозиціями та голосуванням. І хоча DAOstack став одним із найбільших подібних проєктів свого часу, але наразі він не підтримується.

3. **Voatz** (https://voatz.com/) розроблена 2018 року в Західній Вірджинії для участі закордонних військових в голосуванні США. Метою було усунення традиційних бар'єрів, пов’язаних з відсутністю доступу до виборчих дільниць та складністю з фізичною ідентифікацією виборців.

Таблиця 1.3 – SWOT-аналіз платформи Voatz

Сильні сторони	Слабкі сторони
– дозволяє користувачам голосувати за допомогою своїх смартфонів з будь-якої точки світу, значно спрощуючи участь в демократичних процесах; – голосування проводиться шляхом верифікації біометричних даних (наприклад, відбитків пальців або розпізнавання обличчя), що забезпечує надійну автентифікацію виборців та запобігає фальсифікації голосів; – записи про голосування зберігаються на блокчейні, що гарантує незмінність та перевірку результатів без можливості зовнішнього втручання	– використання біометричних даних для верифікації учасників голосування створює потенційні ризики для приватності виборців, оскільки зберігання та обробка біометричної інформації вимагають високого рівня захисту; – необхідність використання мобільного застосунку для участі в голосуванні може обмежити доступність голосування для осіб без смартфонів або з обмеженим доступом до Інтернету, а також для тих, хто не володіє необхідними технічними знаннями для користування такими застосунками
Можливості	Загрози
– має біометричну перевірку (відбитки пальців або сканування сітківки ока), щоб виборці перевіряли своїх заявників і себе в заявці; – пропонує значний потенціал для реформування способів, за допомогою яких проводяться виборчі процеси	– має серйозні недоліки в безпеці, які дозволяють зловмисникам відстежувати голоси та редагувати або блокувати бюлетені у великих кількостях; – не всі користувачі готові надавати свої біометричні дані через занепокоєння щодо їхньої конфіденційності

Voatz є інноваційною платформою для децентралізованого голосування, яка використовує комбінацію блокчейн-технології та біометричної верифікації для

забезпечення безпеки, прозорості та надійності виборчих процесів. Вона є привабливою платформою для організацій, які шукають безпечні та прозорі способи проведення виборів, опитувань або інших форм колективного прийняття рішень. Здатність Voatz забезпечити доступність, безпеку та прозорість може відіграти ключову роль у розвитку сучасних демократичних процедур та у впровадженні нових стандартів для голосування у майбутньому.

4. **DAOhaus** (https://daohaus.club) — зручна платформа для створення та керування DAO з акцентом на простоті та доступності. Вона дозволяє користувачам швидко налаштувати DAO і його параметри керування, а також керувати пропозиціями та голосуванням [14].

Таблиця 1.4 – SWOT-аналіз платформи DAOhaus

Сильні сторони	Слабкі сторони
– архітектура DAOhaus побудована на багаторівневому підході для забезпечення гнучкості, масштабованості та простоти використання; – DAOhaus реалізує просту систему голосування, яка є системою без кворуму, де завжди достатньо відносної більшості для схвалення пропозиції; – має унікальну функцію, яка надає учасникам можливість вийти з DAO і зберегти всі свої вкладені кошти	– учасники, які не є DAO, можуть подавати пропозиції для голосування лише через спонсорство (депозит) учасника DAO, де учасник-спонсор отримає частину депозиту назад після завершення голосування
Можливості	Загрози
– DAOhaus не вимагає кворуму, відносна більшість може схвалити пропозицію, тому вона показує більшу кількість схвалених пропозицій; – якщо більше ⅓ членів відмовляються від пропозиції, вона не буде прийнята	– пропозиція потребує спонсорства члена спільноти, це також стосується пропозицій, які можуть бути відхилені

Використання технології блокчейн у системах електронного голосування (е-голосування) привертає значну увагу через її здатність підвищувати прозорість, безпеку та цілісність цифрового голосування. Незважаючи на властивості технології блокчейн і переваги, які вона пропонує, ці системи за своєю суттю не можна застосовувати в усіх контекстах голосування через певні перешкоди.

Дослідження популярних платформ на технології блокчейн у сфері електронного голосування дозволили визначити потенційні проблеми та спрогнозувати можливі напрямки розвитку. Хоча системи голосування на основі блокчейну забезпечують безпеку, прозорість і можуть запобігти маніпулюванню даними та проблемам цілісності, найбільш часто проблемами є масштабованість, економічна ефективність, автентифікація та анонімність користувача, конфіденційність і безпека транзакцій в системах електронного голосування на основі блокчейну.

Усвідомлення недоліків є важливим кроком на шляху до розробки більш ефективних, безпечних і легких для користувачів децентралізованих застосунків і платформ для голосування. Основними недоліками вже створених платформ є складність їх використання, необхідність в специфічній автентифікації (наприклад біометричній), а також відсутність гнучкості в умовах голосування. Це значно обмежує кількість потенційних користувачів, бо, наприклад для інтеграції голосування у якийсь стартап, розробнику потрібно буде витратитись на розгортання серверу для автентифікації виборців.

Враховуючи ці обмеження, стає очевидною потреба в розробці та впровадженні нової архітектури для додаткових заходів безпеки та забезпечення приватності, а також створення більш доступних методів участі в голосуваннях для широкого кола виборців. Розширення варіантів доступу до голосування, зокрема через вебінтерфейси або альтернативні методи верифікації, могло б допомогти подолати ці виклики і зробити децентралізоване голосування більш прийнятним та практичним для різноманітних проєктів та ініціатив.

1.4 Визначення функціональних вимог до створюваної системи електронного голосування

Після аналізу наявних рішень можна визначити основні функціональні вимоги голосування.

Основні функціональні вимоги створюваної системи електронного голосування:

– *Забезпечення високого рівня безпеки*. Використання блокчейн-технології дозволяє забезпечити незмінність голосів та їх захист від зовнішніх та внутрішніх спроб маніпуляцій. Смарт-контракти створюють безпечне середовище, де кожен голос реєструється, верифікується та зберігається без можливості його зміни.

– *Прозорість виборчого процесу*. Прозорість забезпечується завдяки публічному характеру блокчейну, що дозволяє всім учасникам виборів перевіряти хід голосування та підрахунку голосів в реальному часі. Це значно підвищує довіру до системи голосування, зменшуючи можливості для сумнівів чи звинувачень у фальсифікаціях.

– *Надійність*. Система проєктується з урахуванням високих стандартів надійності. Автоматизовані процеси, що керуються смарт-контрактами, мінімізують людські помилки та забезпечують стабільність та ефективність виборчих процесів.

– *Доступність*. Система дозволяє залучати ширший круг виборців, зокрема людей, які через фізичні обмеження або віддаленість від виборчих дільниць раніше не могли взяти участь у голосуванні. Інтерфейс системи розробляється таким чином, щоб бути зрозумілим і зручним для користувачів будь-якого віку та рівня технологічної освіченості.

Забезпечити гнучкість і водночас розширити потенційну аудиторії можна шляхом розділення логіки голосування від його правил та умов, за якими учасники можуть брати в ньому участь. Використання окремого контракту для голосування та іншого контракту для керування умовами участі та іншими аспектами проєкту (наприклад, краудфандингова ініціатива, що встановлює можливість голосування лише тим, хто вклав необхідну суму) дозволяє забезпечити гнучкість та масштабованість системи.

Основні переваги такої декомпозиції:

– *гнучкість:* система може бути легко адаптована під різні умови використання без потреби переписування коду контракту голосування;

– *масштабованість:* незалежні контракти дозволяють обробляти велику кількість голосувань та учасників ефективно, розподіляючи навантаження та ресурси;

– *безпека:* ізоляція контрактів знижує ризики безпеки, оскільки зловмисники не можуть вплинути на весь процес через вразливості в одному контракті; крім того, вона полегшує аудити безпеки;

– *відповідність до стандартів:* кожен контракт може бути оптимізований для виконання конкретної цілі без впливу на інші аспекти системи;

Цей підхід не тільки покращує адміністрування системи голосування, а й робить її доступнішою для ширшої аудиторії, оскільки відмовляється від потреби складних методів аутентифікації у формі біометрії або SMS-підтвердження, замість цього використовуючи криптографічно захищені Ethereum адреси.

Виключення людського фактору досягається через зазначення умов завершення голосування у контракті з його правилами. Ця функціональність значно спрощує керування голосуванням та покращує його ефективність, завдяки тому, що голосування автоматично вважається завершеним, коли виконуються вказані правила.

1.5 Висновки до розділу 1

Проведений аналіз наявних блокчейн-технологій дозволив з’ясувати, що розробка і впровадження системи електронного голосування на базі блокчейн не тільки вирішує багато наявних проблем, але й відкриває нові можливості для зміцнення демократичних процесів. Це є особливо актуальним у контексті сучасних викликів, пов'язаних із забезпеченням цілісності та надійності виборів.

Розглянуто ключові поняття, які лежать в основі блокчейн-технологій, оскільки це критично важливо для розуміння їх принципів роботи та можливостей застосування. З’ясовано, що системи електронного голосування, засновані на

блокчейнах, використовують різні концепції та технології для забезпечення безпечних і надійних виборів. Серед цих технологій — мережі блокчейну Ethereum та Hyperledger Fabric, консенсусні алгоритми Proof of Work, Proof of Stake і Practical Byzantine Fault Tolerance, а також методи підвищення конфіденційності, наприклад, гомоморфне шифрування та докази з нульовим знанням. Крім того, механізми автентифікації, наприклад, біометрична перевірка та системи керування ідентифікацією, мають вирішальне значення для підтвердження легітимності виборця та підтримки цілісності системи голосування. Сформульовано переваги та недоліки систем електронного голосування на блокчейні Ethereum.

Проведено SWOT-аналіз відомих конкурентних платформ для електронного голосування на технології блокчейн: Aragon, DAOstack, DAOhaus і Voatz. Кожна з яких має свої унікальні особливості щодо зростання та активності, а також щодо результатів голосування. Дослідження популярних платформ на технології блокчейн у сфері електронного голосування дозволили визначити потенційні проблеми та спрогнозувати можливі напрямки розвитку.

Аналіз наявних рішень допоміг визначити основні функціональні вимоги до створюваної системи електронного голосування. Реалізація такої системи має ряд значних переваг, які відображають важливість та ефективність запроваджених технологічних рішень.

2 ПРОЄКТУВАННЯ СИСТЕМИ ГОЛОСУВАННЯ

2.1 Аналіз мови програмування Solidity

Solidity є високорівневою мовою програмування, спеціально розробленою для створення смарт-контрактів, які виконуються на Ethereum Virtual Machine (EVM) [15]. Як основний інструмент для розробників Ethereum вона дозволяє створювати складні контракти з різними правилами та умовами, автоматизуючи обмін валютами, власністю, акціями або будь-якими цінними активами без посередників та за високого рівня безпеки. З моменту свого запуску в 2014 році, вона пройшла значний шлях розвитку. Спочатку була задумана як інструмент, який би забезпечував розробникам максимальну гнучкість при створенні децентралізованих застосунків на блокчейні Ethereum.

Синтаксис Solidity нагадує JavaScript, що робить цю мову доступною для широкого кола програмістів. Основною ціллю її створення було спростити розробку смарт-контрактів, надаючи потужні, але зрозумілі інструменти для роботи зі станами, об'єктами та функціями в рамках блокчейну [16]. Однією з ключових характеристик Solidity є її здатність моделювати складні фінансові угоди, корпоративне керування, децентралізоване фінансування (DeFi) та інші операції, які потребують надійного виконання умов без зовнішнього втручання (рис. 2.1).

Використання Solidity сприяло широкому прийняттю Ethereum як платформи для розробки dApps, оскільки вона забезпечувала розробникам потужні та гнучкі інструменти для втілення їхніх ідей в життя. Протягом років розвитку спільнота розробників активно працювала над вдосконаленням мови, додаючи нові функції, покращуючи безпеку та збільшуючи ефективність виконання контрактів. На сьогодні Solidity залишається основною мовою для розробки на Ethereum, а її постійний розвиток і вдосконалення допомагають вирішувати актуальні виклики, пов'язані з безпекою, масштабуванням та інтеграцією смарт-контрактів в сучасний цифровий світ [17].

contracts > Voting.sol

```
1   // SPDX-License-Identifier: MIT
2   pragma solidity ^0.8.19;
3
4   import "@openzeppelin/contracts/access/Ownable.sol";
5
6   contract Voting is Ownable {
7       struct Candidate {
8           string description;
9           string imageLink;
10          uint32 votes;
11      }
12
13      string public description;
14      uint public startTimestamp;
15      uint public endTimestamp;
16      uint public createdTimestamp;
17      mapping(address => uint) public votedNumber;
18      Candidate[] public candidates;
19
20      constructor(···
26      ) Ownable(msg.sender) {···
40      }
41
42      function vote(address voter, uint candidateIndex) external onlyOwner {
43          require(votedNumber[voter] == 0, "Already voted");
44          require(
45              startTimestamp <= block.timestamp && block.timestamp < endTimestamp,
46              "Voting is not active"
47          );
48
49          Candidate storage candidate = candidates[candidateIndex];
50          votedNumber[voter] = candidateIndex + 1;
51          candidate.votes += 1;
52      }
53
54      function getCandidatesCount() public view returns (uint) {
55          return candidates.length;
56      }
57  }
58
```

Рисунок 2.1 – Код написаний мовою програмування Solidity

Через стрімкий розвиток технологій блокчейн Solidity набула особливого значення, як місток між традиційним програмуванням та новітніми можливостями децентралізованих фінансів. Роль Solidity у цьому контексті полягає не лише у забезпеченні інструментів для створення смарт-контрактів, але й у формуванні нового

підходу до розуміння власності, угод та корпоративного керування через децентралізацію. Завдяки її міцному підґрунтю в блокчейн екосистемі, вона також сприяє експериментам та інноваціям у сфері децентралізованих застосунків.

Розробники створюють складні децентралізовані автономні організації, децентралізовані бірж, а також ігри та інші застосунки, що використовують різноманітні токени. Ця мова програмування не тільки спрощує розробку в межах екосистеми Ethereum, але й відкриває шлях для адаптації та прийняття блокчейн-технологій у широкому спектрі галузей [18]. Наприклад, в сфері нерухомості, може використовуватися для створення смарт-контрактів, які автоматизують процеси купівлі-продажу, забезпечуючи безпеку та прозорість угод. У фінансовому секторі дозволяє реалізовувати складні фінансові інструменти, такі як деривативи та кредитні угоди, на децентралізованих платформах. Solidity постійно розвивається, щоб відповідати зростаючим потребам ринку та адресувати виклики, пов'язані з безпекою смарт-контрактів.

Спільнота розробників активно працює над оптимізацією її функціональності, поліпшенням інструментів для тестування та налагодження. Ці зусилля не тільки підвищують якість та надійність розроблених на програм, але й сприяють ширшому прийняттю блокчейн-технологій серед розробників і користувачів по всьому світу.

Для подальшого розуміння матеріалу необхідно дати визначення таким термінам:

– *смарт-контракт* – це програма, яка автоматично виконує, контролює або документує значущі події або дії згідно з умовами контракту, використовуючи блокчейн-технології. Смарт-контракти дозволяють здійснювати надійні транзакції без потреби у посередниках, забезпечуючи високий рівень безпеки та знижуючи можливість шахрайства [19];

– *транзакція* у контексті блокчейну визначається як запис, який ініціює зміну стану в розподіленому реєстрі, наприклад: передача криптовалютних активів між адресами, виклик функції смарт-контракту або реєстрація нового смарт-контракту.

Транзакції є основними будівельними блоками мережі Ethereum, забезпечуючи здатність до здійснення децентралізованих обчислень та передачі цінностей [20];

– *газ* (gas) у контексті блокчейну Ethereum означає одиницю вимірювання, яка використовується для визначення кількості обчислювальних ресурсів, необхідних для виконання операцій у мережі, як-от: транзакції або виклик функції смарт-контракту. Вартість газу визначається на основі складності виконуваних операцій та сплачується у формі криптовалюти Ether [21];

– *майнінг* – процес додавання транзакцій до історії блокчейну, який використовується в багатьох наявних мережах. Цей процес здійснюється шляхом розв'язання складної математичної задачі, що вимагає значних обчислювальних ресурсів. Розв'язання цієї задачі, відоме як доказ роботи (Proof Of Work), є необхідним для забезпечення консенсусу в децентралізованій мережі [22];

– *децентралізована автономна організація* (DAO) – форма організації, керована смарт-контрактами на блокчейні, яка функціонує без централізованого керування. DAO створюється та управляється за допомогою коду, забезпечуючи автоматизацію управлінських рішень та операцій, що дозволяє учасникам колективно ухвалювати рішення на основі прозорих правил та голосувань [23];

– *децентралізована біржа* (DEX) – платформа для обміну криптовалют, яка дозволяє користувачам торгувати криптовалютами безпосередньо між собою без втручання централізованого посередника. DEX забезпечує високий рівень приватності, безпеки та контролю над особистими коштами, використовуючи технологію блокчейн для керування транзакціями [24].

Безпека є критично важливим аспектом при розробці системи голосування, враховуючи здатність програм управляти значними фінансовими активами та виконувати важливу бізнес-логіку без можливості відкату виконаних транзакцій. Solidity містить широкий набір інструментів та практик для зміцнення безпеки програм. Серед них: модифікатори доступу, шаблони безпечних математичних

операцій для уникнення переповнень, вбудовані функції безпеки для перевірки умов перед виконанням операцій, а також значення за замовчуванням для всіх типів даних.

Одним з основних інструментів безпеки є система типів Solidity, яка допомагає виявляти помилки на ранніх етапах розробки. Використання інтерфейсів та наслідування дозволяє розробникам створювати чіткі структури контрактів, що сприяє кращому розумінню та контролю над поведінкою контрактів.

Масштабованість та ефективність є ключовими факторами, що впливають на вибір Solidity для розробки систем голосування. Завдяки оптимізації коду та ефективному використанню газу, смарт-контракти, написані на Solidity, можуть обробляти великі обсяги транзакцій, що є важливим для голосувань з великою кількістю учасників. Розробники мають можливість використовувати широковідомі патерни проєктування та оптимізації для зменшення вартості транзакцій, збільшення швидкості обробки та забезпечення масштабування системи в міру збільшення кількості голосів.

Solidity надає розробникам широкі можливості для розробки та тестування смарт-контрактів. Інтеграція з такими інструментами, як Truffle Suite, Hardhat та Remix, дозволяє розробникам легко писати, тестувати та розгортати свої програми. Ці інструменти надають потужні середовища розробки з вбудованими функціями для модульного тестування, міграції контрактів та взаємодії з розгорнутими контрактами, що сприяє підвищенню якості та надійності кінцевих продуктів.

Активна та зростаюча спільнота розробників навколо Solidity грає важливу роль у виборі цієї мови програмування для розробки систем голосування. Розширені можливості для обміну знаннями, досвідом та кращими практиками дозволяють розробникам швидко знаходити рішення для викликів, з якими вони стикаються. Через постійні оновлення, документацію, форуми та робочі групи, розробники можуть залишатися в курсі останніх трендів і інновацій в блокчейн розробці, що безпосередньо впливає на успішність та ефективність систем голосування, створених на основі Solidity.

Програми, розроблені на цій мові програмування, можуть бути повністю прозорими для всіх учасників голосування. Кожен має можливість перевірити код контракту, що дозволяє забезпечити справедливість та відкритість процесу голосування. Така прозорість допомагає підвищити довіру учасників до системи та забезпечити їхню впевненість у чесності та в тому, що результати не були підтасовані [25].

Розробники можуть створювати смарт-контракти з високим рівнем гнучкості та налаштовуваності. Це означає, що системи голосування можуть бути точно адаптовані під специфічні вимоги та потреби кожного проєкту, включаючи різні механізми голосування, правила участі та процедури підрахунку голосів.

Використання програм для систем голосування значно спрощує процес керування виборами, автоматизуючи багато кроків, від реєстрації учасників до підрахунку голосів та оголошення результатів. Це зменшує можливість людських помилок та оптимізує використання ресурсів, роблячи голосування більш ефективним та доступним.

Отже, Solidity відіграє ключову роль у розвитку блокчейн екосистеми, дозволяючи створювати новітні та революційні застосунки, наприклад, системи голосування.

2.2 Аналіз середовища розробки в системі Ethereum Hardhat

Hardhat (https://hardhat.org/) є сучасним інструментом розробки для Ethereum, який надає розробникам можливості для написання, тестування, відладки, та розгортання смарт-контрактів. Як середовище, що цілком фокусується на розробці смарт-контрактів, воно включає в себе інтегроване керування смарт-контрактами, автоматичне виконання тестів, і вбудовані функції для взаємодії з блокчейном Ethereum. Основною його метою є спрощення процесу розробки dApps (децентралізованих застосунків) та смарт-контрактів, надання широких можливостей для тестування та розгортання, а також вдосконалення зворотного зв'язку під час

розробки, що дозволяє розробникам швидше виявляти і виправляти помилки (рис. 2.2).

```
describe("Deployment", async () => {
  it("Should correctly start voting on deployment", async () => {
    const { contractConstructor, contract, owner, account2 } =
      await loadFixture(deployVotingFixture);
    const contractStartTimestamp = await contract.startTimestamp();
    const contractEndTimestamp = await contract.endTimestamp();
    const contractCandidates = await Promise.all(
      [0, 1, 2].map((ind) => contract.candidates(ind))
    );

    expect(contractStartTimestamp).to.eq(contractConstructor.startTimestamp);
    expect(contractEndTimestamp).to.eq(contractConstructor.endTimestamp);
    expect(contractStartTimestamp).to.eq(contractConstructor.startTimestamp);
    expect(contractCandidates.map((c) => c.description)).all.members(
      contractConstructor.candidatesDescriptions
    );
    expect(contractCandidates.map((c) => c.imageLink)).all.members(
      contractConstructor.candidatesImages
    );
  });
});
```

Рисунок 2.2 – Тест, написаний мовою програмування TypeScript на платформі Hardhat

Платформа Hardhat підтримує багато плагінів, що розширюють її функціональність і забезпечують інтеграцію з іншими популярними інструментами і бібліотеками, а саме:

– Ethers.js – бібліотека, яка надає об'єкт ethers в HRE для взаємодії з блокчейном Ethereum, виконання транзакцій, зчитування стану контрактів тощо;

– Waffle – інструмент для тестування смарт-контрактів, який надає розширені можливості для написання і виконання модульних тестів;

– TypeChain – плагін, який генерує TypeScript типи з Solidity контрактів, спрощуючи розробку за допомогою строгої типізації і відлагодження коду.

Hardhat було розроблено компанією Nomic Labs, яка прагне зробити процес розробки Ethereum більш інтуїтивно зрозумілим та доступним для розробників на всіх

рівнях досвіду. З моменту свого запуску, цей інструмент швидко став популярним серед розробників завдяки своїй простоті використання та глибокій інтеграції з Ethereum екосистемою.

Однією з основних його переваг є можливість запуску власної локальної мережі. Вона використовується для розробки і тестування смарт-контрактів. Вона емулює поведінку реального блокчейну Ethereum, включаючи майнінг блоків, виконання транзакцій, і взаємодії з контрактами, але без необхідності витрачати реальні кошти на газ і без ризику впливу на реальні активи. Ця мережа підтримує розширену відладку та дозволяє виконати миттєвий відкат змін, що корисно при тестуванні. Розробники можуть використовувати цю мережу для повного циклу тестування смарт-контрактів, від модульних тестів до інтеграційного тестування, забезпечуючи високу якість і надійність продукту.

Hardhat наразі становить основу для сучасної розробки децентралізованих застосунків на Ethereum, забезпечуючи розробникам потужний набір інструментів, що дозволяє ефективно та безпечно створювати інноваційні застосунки в блокчейн просторі.

Hardhat Runtime Environment (HRE) є ключовим компонентом Hardhat, який надає централізований доступ до всіх функціональних можливостей інструменту, включаючи виконання смарт-контрактів, запуск тестів, і інтерактивну взаємодію з контрактами. HRE автоматично створюється при запуску будь-якої задачі цією платформи або скрипта і є глобальним, що означає легкий доступ до нього з будь-якого місця у проєкті. HRE інтегрує у себе плагіни та засоби командного рядка, забезпечуючи розробникам гнучкість у використанні зовнішніх бібліотек і інструментів, таких як ethers.js для взаємодії з блокчейном Ethereum. Це середовище виконання знижує складність розробки і дозволяє розробникам зосередитися на логіці смарт-контрактів та надає потужні та інтуїтивно зрозумілі інтерфейси для різноманітних задач.

Скрипти розгортання дозволяють автоматизувати процес розгортання смарт-контрактів у мережі блокчейн. Вони використовують конфігураційні файли і скрипти, написані на JavaScript або TypeScript, для взаємодії з мережею, відправлення контрактів, і керування версіями.

Використання цього середовища при розробці смарт-контрактів для систем голосування є критично важливим для забезпечення високих стандартів безпеки та надійності. У контексті голосування, де кожен голос має значення і потенційно може бути предметом маніпуляції, воно надає розробникам інструменти для ретельного тестування, яке дозволяє підтвердити, що всі смарт-контракти виконуються точно як очікується.

Hardhat включає функції, такі як вбудоване керування станами, симуляція мережевих умов, і детальне логування викликів функцій, які дозволяють розробникам виявляти і виправляти потенційні вразливості у контрактах. Ці можливості критично важливі для систем голосування, які повинні передбачати потенційні проблеми, такі як: підкуп, подвійне голосування, та інші схеми фальсифікації.

Ця платформа значно спрощує процес розробки смарт-контрактів через своє інтуїтивно зрозуміле середовище виконання та інструменти для локального тестування і розгортання. Розробники можуть використовувати Hardhat Network, для відтворення умов експлуатації без потреби звертання до громадських тестових мереж. Це не лише збільшує швидкість розробки, а й значно знижує бар'єри для експериментування і ітерації, оскільки розробники можуть швидко розгортати та тестувати зміни в реальному часі, без витрат на газ чи часу очікування на майнінг транзакцій.

Автоматизація рутинних процесів і масштабування систем голосування є важливими аспектами, де Hardhat може значно допомогти. Завдяки інтеграції з іншими інструментами і плагінами, стає можливим автоматизувати процеси від розгортання контрактів до керування версіями та збору статистики виконання. Це дозволяє ефективно масштабувати системи голосування для обробки великих обсягів даних та транзакцій без втрати продуктивності чи безпеки.

У контексті Ethereum, витрати на газ можуть значно впливати на загальні витрати операцій, особливо в системах голосування, де кожна транзакція вимагає обробки смарт-контрактом. Платформа включає інструменти для моніторингу і оптимізації витрат на газ, такі як візуалізація використання газу і інструменти для аналізу транзакцій. Розробники можуть використовувати ці інструменти для покращення контрактів, мінімізуючи необхідний газ і тим самим знижуючи вартість транзакцій у масштабах цілої системи голосування.

Спільнота розробників зіграла вирішальну роль у розвитку та популяризації середовища як ведучого інструменту для розробки на Ethereum. Внесок спільноти охоплює різні аспекти, від розробки плагінів та інструментів до створення докладної документації і надання підтримки новим користувачам. Вони не лише розширюють функціональні можливості платформи, але й значно спрощують її використання та доступність:

– *Створення плагінів*. Плагіни можуть включати все, від простих утиліт, які полегшують повсякденні задачі, до комплексних систем, що інтегрують інші блокчейн-екосистеми та розширюють їх можливості. Вони дозволяють розробникам адаптувати проєкт до своїх конкретних потреб, що робить платформу універсальним інструментом для широкого спектра блокчейн-задач.

– *Документація та ресурси*. Спільнота активно працює над створенням і оновленням документації, навчальних матеріалів, і практичних посібників. Ця документація є незамінною для нових користувачів і допомагає їм швидко зрозуміти основи роботи та ефективно використовувати всі можливості.

– *Підтримка нових користувачів*. Також, спільнота надає обширну підтримку через форуми, групи в соціальних мережах, та інші платформи. Розробники з усього світу можуть отримати відповіді на свої питання, поділитися досвідом, або навіть отримати професійні консультації.

Отже, Hardhat не просто полегшує розробку і тестування смарт-контрактів, а й забезпечує необхідні інструменти для ефективного масштабування, автоматизації, і

оптимізації систем голосування на блокчейні, роблячи їх більш доступними, безпечними та економічно ефективними.

2.3 Аналіз фронтенд бібліотеки React

React (https://react.dev) – це декларативна, ефективна, гнучка JavaScript-бібліотека для побудови інтерфейсів користувача, яка дозволяє розробникам ефективно створювати складні, інтерактивні UI з використанням компонентів [26]. Створена командою Facebook, вона широко використовується для побудови односторінкових застосунків, де необхідно забезпечити швидке оновлення інтерфейсу в реальному часі без перезавантаження сторінки.

Однією з унікальних особливостей є використання JSX-синтаксису, що дозволяє включати HTML-структури в код JavaScript (рис. 2.3).

crowdfunding-dapp > src > components > voting > Voting.tsx > ...

```tsx
import React from 'react';
import { Card } from 'react-bootstrap';
import CandidateListComponent from './CandidateList';
import { Voting } from '../../models/Voting';
import { CrowdFundingInfoShared } from '../../models/CrowdFunding';

interface VotingProps {
  voting: Voting;
  index: number;
  sharedInfo: CrowdFundingInfoShared;
  onVote: (candidateIndex: number) => Promise<void>;
}

const VotingComponent: React.FC<VotingProps> = ({ voting, index, sharedInfo, onVote }) => {
  return (
    <Card className="mb-4 shadow-sm">
      <Card.Header as="h4" className="bg-primary text-white">Голосування {index + 1}. {voting.description}</Card.Header>
      <Card.Body>
        <CandidateListComponent
          candidates={voting.candidates}
          sharedInfo={sharedInfo}
          votedFor={voting.votedFor}
          onVote={onVote}
        />
      </Card.Body>
    </Card>
  );
};

export default VotingComponent;
```

Рисунок 2.3 – Компонент, написаний за допомогою синтаксису JSX

JSX робить код більш читабельним і спрощує процес визначення вигляду інтерфейсу. За допомогою JSX, розробники можуть легко визначати структуру UI компонента в рамках JavaScript-коду, що знижує комплексність та підвищує продуктивність розробки.

React використовує компонентний підхід до розробки, де кожен компонент має свій стан і логіку, які можна повторно використовувати в інших частинах застосунку без повторення коду. Цей підхід не тільки сприяє модулярності і зменшенню залежностей між частинами застосунку, але й дозволяє легко тестувати окремі компоненти. Компоненти можуть бути як функціональними, так і класовими, залежно від потреб розробника та комплексності логіки, що в них втілюється.

Ця бібліотека має концепцію віртуального DOM, що є легкою копією реального DOM у браузері. Ця технологія дозволяє оптимізувати процес внесення змін у інтерфейс користувача, оскільки фактичні зміни в DOM (які є відносно витратними з точки зору продуктивності) відбуваються лише після порівняння змісту віртуального і реального DOM та визначення необхідних оновлень. Це значно збільшує швидкість відгуку інтерфейсів, особливо в застосунках з великою кількістю динамічного контенту.

React застосовує односпрямований потік даних, що робить поведінку застосунку більш передбачуваним та легшим для розуміння. Він використовує концепцію "props" для передачі даних від батьківських компонентів до дочірніх (вниз по ієрархії) і "state" для керування станом всередині самого компоненту. Ця архітектура не тільки спрощує потік даних у складних застосунках, але й сприяє більшій масштабованості та легшому управлінню станом застосунку.

React підтримується однією з найбільших розробницьких спільнот. Це не тільки сприяє швидкому вирішенню проблем та питань, які можуть виникнути у розробників, але й забезпечує стабільне оновлення бібліотеки з урахуванням новітніх технологічних трендів і змін у вебстандартах. Багатство доступних навчальних ресурсів, від документації і статей до відеоуроків і спеціалізованих курсів, робить його

доступним для новачків у програмуванні, водночас дозволяючи досвідченим розробникам поглиблювати свої знання і вміння.

Хоча JavaScript є мовою з динамічною типізацією, React може використовуватися разом з TypeScript, який додає строгу статичну типізацію. Це не тільки сприяє кращій організації коду та його безпеці, але й забезпечує додаткові переваги під час розробки складних застосунків, такі як раннє виявлення помилок та більш ефективна робота з редакторами коду.

Ця бібліотека не просто залишається популярним вибором у світі фронтенд розробки, вона продовжує еволюціонувати і адаптуватися до змінних вимог ринку, забезпечуючи розробників потужним, ефективним та гнучким інструментом для створення динамічних вебінтерфейсів. Його здатність інтегруватися з різними сервісами, підтримка широкого спектра плагінів, і велика активна спільнота забезпечують успішне впровадження в проєкти будь-якої складності і масштабу, від стартапів до великих корпоративних систем. Також, завдяки своїй модульності та ефективності вона, став популярним вибором для розробки фронтенду децентралізованих застосунків, що використовують блокчейн-технології, такі як Ethereum. Використання React дозволяє створювати швидкісні, відгукові інтерфейси, які можуть ефективно взаємодіяти з блокчейн та відображати дані в реальному часі.

2.4 Архітектура системи голосування

Системи голосування базується на блокчейні Ethereum, це забезпечує безпеку та незмінність процесу голосування. На відміну від більшості інших рішень, де голосування являє собою один смарт-контракт, розроблена система розділена на два основних типи смарт-контрактів: контракт для голосування та контракт для керування умовами голосування (рис. 2.4).

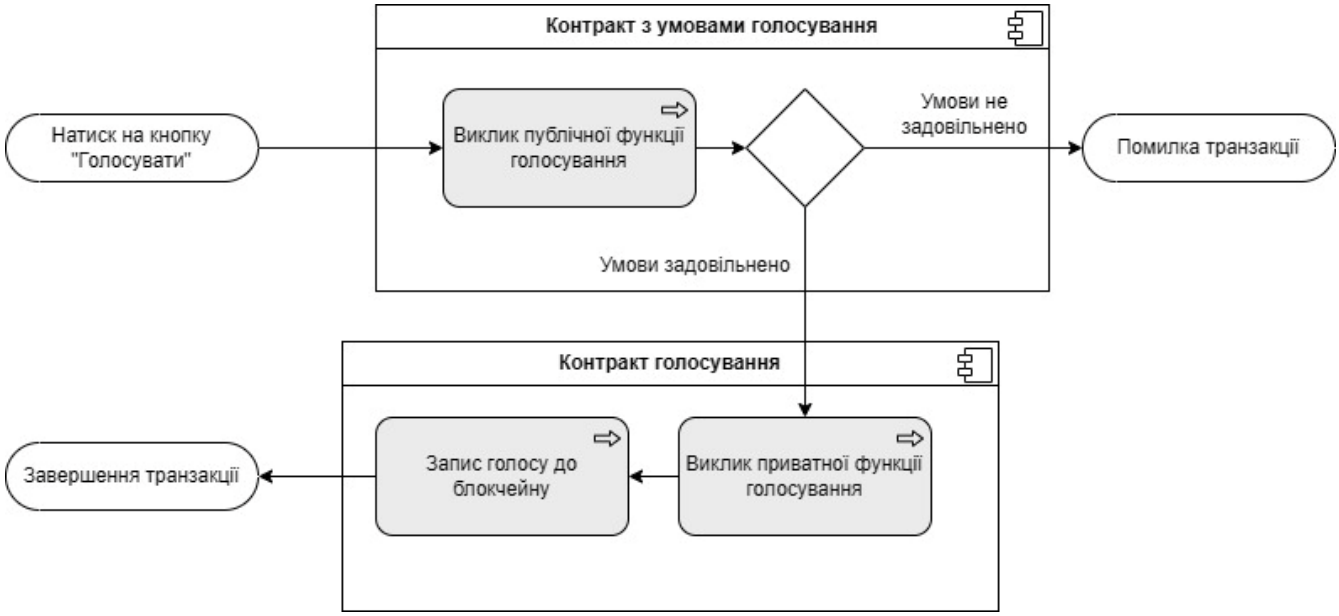

Рисунок 2.4 – UML-діаграма послідовності процесу голосування

У даній роботі програма з умовами голосування по суті буде краудфандинговою кампанією.

Контракт для голосування реалізує логіку підрахунку голосів. Для кожного голосування створюється окремий екземпляр цього контракту. Основні його етапи:

– *ініціалізація*: організатор створює голосування, вказуючи питання, список кандидаті, та часовий період, в якому голосування активне;

– *проведення голосування*: учасники, що мають право голосу, можуть обирати варіанти відповідно до власних переконань, кожен голос реєструється і зберігається в блокчейні так, що можна перевірити за кого було надано голос;

– *завершення голосування*: після закінчення визначеного часу голосування автоматично вважається закритим і нові спроби проголосувати видаватимуть помилку.

Контракт для умов голосування визначає основні умови для участі у голосуванні. У цій роботі він включає логіку для внесення коштів учасниками краудфандингової кампанії, а також критерії, які необхідно виконати для активації їхнього права голосу. Функції цієї частини системи:

– *ініціалізація*: організатор запускає контракт, встановлюючи мінімальну суму внеску, необхідну для участі у голосуванні і вказуючи часові рамки заходу і ціль збору;

– *внесення коштів*: користувачі відправляють кошти на контракт, який зберігає їх внески;

– *участь у голосуванні*: користувачі, які внесли достатньо коштів, отримують право голосу в будь-якому голосуванні, яке створив організатор;

– *виведення коштів*: у разі невдачі кампанії (ціль по коштах не досягнута) користувачі можуть забрати свої кошти, а якщо кампанія успішна, то організатор може вивести кошти з контракту.

Фронтенд частина проєкту матиме зручний інтерфейс для *взаємодії* зі смарт-контрактом.

Інтерфейс організатора:

– *створення голосування*: організатор може легко створити нове голосування через користувацький інтерфейс, вказавши всі необхідні деталі (опис, кандидати, тривалість голосування);

– *виведення коштів*: якщо кампанія вдалася, то після завершення, організатор матиме можливість вивести кошти через спеціальну кнопку;

Інтерфейс для користувача:

– *внесення коштів:* користувачі можуть легко внести кошти на вказаний смарт-контракт через вебінтерфейс;

– *участь у голосуванні*: після достатнього внеску, користувачі можуть брати участь у голосуванні, вибираючи кандидатів зі списку;

– *перегляд результатів*: користувачі можуть переглядати поточні результати голосування в реальному часі;

– *моніторинг статусу*: користувачі можуть переглядати активні голосування, статус внесків, і час до закінчення кампанії;

– *виведення коштів*: якщо кампанія провалилася, то після завершення, користувач матиме можливість вивести внесені кошти через спеціальну кнопку.

Систему призначена для запуску у екосистемі Arbitrum - новому рішенні масштабування Ethereum. Вона дозволяє використовувати всі переваги основної мережі, однак значно понижує витрати користувачів на транзакції необхідні для взаємодії з смарт-контрактами.

Архітектура пропонованої системи голосування базується на використанні блокчейн-технології, це гарантує високу безпеку та прозорість процесу. Всі голоси зберігаються у блокчейні, що не тільки забезпечує незмінність результатів, але й відкриває публічний доступ до повної інформації про голосування. Це дозволяє будь-якій зацікавленій особі перевіряти результати в будь-який час, що підвищує довіру і відкритість всього процесу. Така система не лише забезпечує ефективне проведення голосувань, але і значно підвищує їх надійність завдяки технічним перевагам блокчейну.

2.5 Висновки до розділу 2

У цьому розділі було розглянуто основні інструменти та технології, що застосовуються в розробці системи голосування на базі блокчейну Ethereum. Аналіз програмної мови Solidity, середовища розробки Hardhat, фронтенд бібліотеки React та архітектури системи голосування підкреслює важливість кожного компонента у створенні ефективної, безпечної та масштабованої системи.

Solidity як основа для смарт-контрактів забезпечує потужний інструментарій для створення смарт-контрактів, що є невід'ємною частиною будь-якої системи голосування на блокчейні. Її схожість з JavaScript спрощує вивчення та застосування мови для широкого кола розробників. Ключові особливості, такі як контроль типів, модифікатори доступу, і система обробки помилок, забезпечують високий рівень безпеки та надійності створених контрактів.

Середовище розробки Hardhat відіграє важливу роль у спрощенні процесів написання, тестування, відладки та розгортання смарт-контрактів. Воно інтегрує різні інструменти та плагіни, які поліпшують процес розробки і допомагають у виявленні помилок на ранніх стадіях. Можливість запуску локальної версії блокчейну дозволяє розробникам ефективно тестувати контракти без витрат реальних коштів.

Фронтенд бібліотека React дозволяє створювати гнучкі та відгукові користувацькі інтерфейси для систем голосування. JSX та віртуальний DOM сприяють ефективній розробці інтерфейсів, які можуть динамічно оновлюватись без перезавантаження сторінки. Це забезпечує високу продуктивність та гарний користувацький досвід.

Щодо архітектури системи голосування, то використання блокчейну Ethereum гарантує безпеку та прозорість системи голосування. Смарт-контракти дозволяють впроваджувати складні логічні умови та забезпечувати автоматичний підрахунок голосів. Модульність системи дозволяє ефективно масштабувати голосування, забезпечуючи стабільність та високу доступність сервісу.

Завдяки поєднанню цих інструментів, система голосування на блокчейні забезпечує не лише технічну надійність та безпеку, але й високий рівень довіри та задоволення користувачів. Подальший розвиток та інтеграція нових технологічних рішень зможуть ще більше розширити можливості та ефективність таких систем, відкриваючи нові горизонти для демократизації процесів прийняття рішень у суспільстві.

3 РЕАЛІЗАЦІЯ СИСТЕМИ ГОЛОСУВАННЯ

3.1 Розробка децентралізованого застосунку

Програма складається з двох смарт-контрактів: Voting та CrowdFunding. Перша програма реалізує основну логіку голосування, в той час, як друга містить в собі умови, за якими голосування буде відбуватися.

3.1.1 Модуль голосування

Цей модуль реалізує основну логіку голосування, таку як запис голосу за обраного кандидата до блокчейну.

Структура Candidate містить поля, які визначають кандидата:

– *description*: текстовий опис;

– *imageLink*: посилання на зображення;

– *votes*: кількість голосів;

Змінні стану голосування:

– *description*: опис;

– *startTimestamp*: час початку;

– *endTimestamp*: час закінчення;

– *createdTimestamp*: час створення;

– *votedNumber*: зберігає інформація про те, хто за якого кандидата проголосував (0 – голос відсутній);

– *candidates*: список кандидатів;

При створенні контракту встановлюються всі змінні, окрім votedNumber. Далі йде перелік функцій контракту.

– *getCandidatesCount* – повертає кількість кандидатів (необхідно для фронтенду).

– *vote* – приймає адресу користувача та індекс кандидата. Перевіряє, чи може користувач голосувати та чи не закінчилось голосування, після чого збільшує

кількість голосів обраного кандидата та зберігає інформацію про те, за кого користувач проголосував.

Функція *vote* модуля голосування доступна лише власнику контракту, тобто власником цього модулю повинен бути модуль з умовами контракту, який в свою чергу буде надавати публічний доступ до цієї функції завдяки заданим умовам.

3.1.2 Модуль краудфандингу

Цей модуль відповідає за умови голосування (в цьому проєкті краудфандингова кампанія).

Змінні стану краудфандингу, що визначають стан контракту:

– *goal*: ціль кампанії (кількість коштів у wei);

– *startTimestamp*: час початку;

– *endTimestamp*: час закінчення;

– *votingThreshold*: мінімальна кількість внесених коштів для можливості голосування;

– *votings*: перелік створених голосувань

– *locked*: булева змінна необхідна для запобігання reentrancy атак;

При створенні контракту встановлюються час початку та закінчення кампанії, ціль і мінімальна кількість внесених коштів для можливості голосування.

Функції контракту:

– *setVotingThreshold* – встановлює значення змінної votingThreshold;

– *getVotingsCount* – повертає кількість створених голосувань;

– *checkFailed* – перевіряє чи кампанія провалилась та повертає відповідне булеве значення;

– *checkSucceeded* – перевіряє чи кампанія завершилась успішно та повертає відповідне булеве значення;

– *vote* – приймає індекс голосування та індекс кандидата. Перевіряє, чи може користувач голосувати та викликає функцію vote обраного голосування, що зберігає голос користувача (рис. 3.1);

– *refund* – повертає кошти користувачу, якщо кампанія провалилася;

– *withdraw* – пересилає всі наявні кошти власнику контракту, якщо кампанія завершилась успішно;

– *receive* – отримує кошти та зберігає інформацію про внесок користувача, якщо кампанія не провалилася.

– *fallback* – дозволяє контракту отримувати кошти навіть, якщо було викликано неіснуючу функцію.

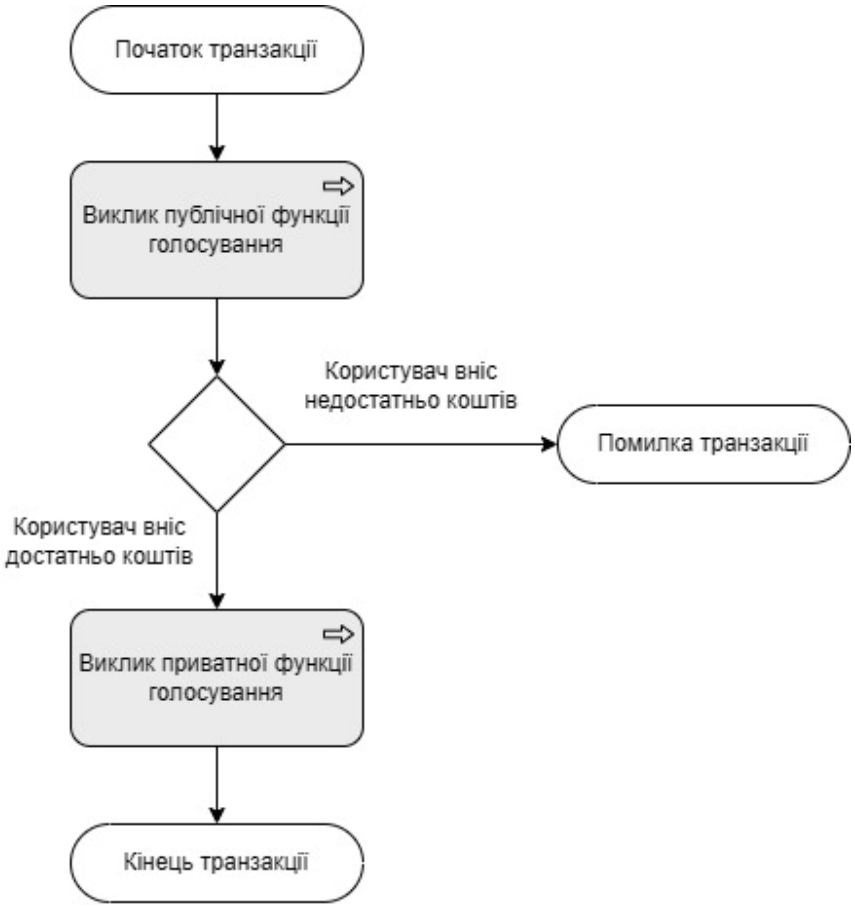

Рисунок 3.1 – UML-діаграма послідовності функції голосування в краудфандингу

Цей смарт-контракт отримує кошти поки не збереться вказана ціль, або не завершиться відведений для збору час (рис. 3.1). Якщо краудфандинг вдався, то власник може забрати зібрані кошти, але якщо він провалився, то всі, хто робив будь-який внесок зможуть повернути свої кошти. Також це контракт дозволяє власнику створювати нові голосування, та перевіряє, що користувач вніс достатньо коштів у кампанію для можливості голосування.

3.2 Розробка інтерфейсу

Інтерфейс застосунку розроблено на бібліотеці React із використанням мови програмування Typescript. Стилізація застосунку використовує платформу Bootstrap. React має функціонально-компонентний підхід до створення фронтенду, тому далі буде перелік всіх розроблених компонентів.

3.2.1 Компонент App

Цей компонент являє собою весь застосунок, тому в ньому оголошено ряд функцій, необхідних для ініціалізації web3-застосунку.

Для виконання коду перед рендером змісту компоненту використовується хук *useEffect.* В ньому насамперед створюється інстанс розгорнутого в мережі блокчейн контракту, а також відбувається підключення до web3-провайдера (наприклад Metamask), що дозволяє взаємодіяти із мережею Ethereum. Також тут виконується функція *fetchContractData*, яка зберігає всю необхідну інформацію про поточний стан краудфандингу. Також в цьому компоненті оголошено ще 3 функції:

- *onVote:* викликає функцію *vote* контракту із необхідними параметрами;
- *onRefund:* викликає функцію *refund* контракту;
- *onWithdraw:* викликає функцію *withdraw* контракту;

У самому компоненті відображається назва краудфандингу, адреса розгорнутого контракту, кнопка для внесення коштів контракту. В залежності від стану кампанії остання кнопка може бути замінена на кнопку повернення або збору

коштів. Далі йдуть компонент з прогресом кампанії, відліком до її завершення. У власника краудфандингу є кнопка для створення нового голосування. Потім йде перелік всіх наявних голосувань (рис. 3.2).

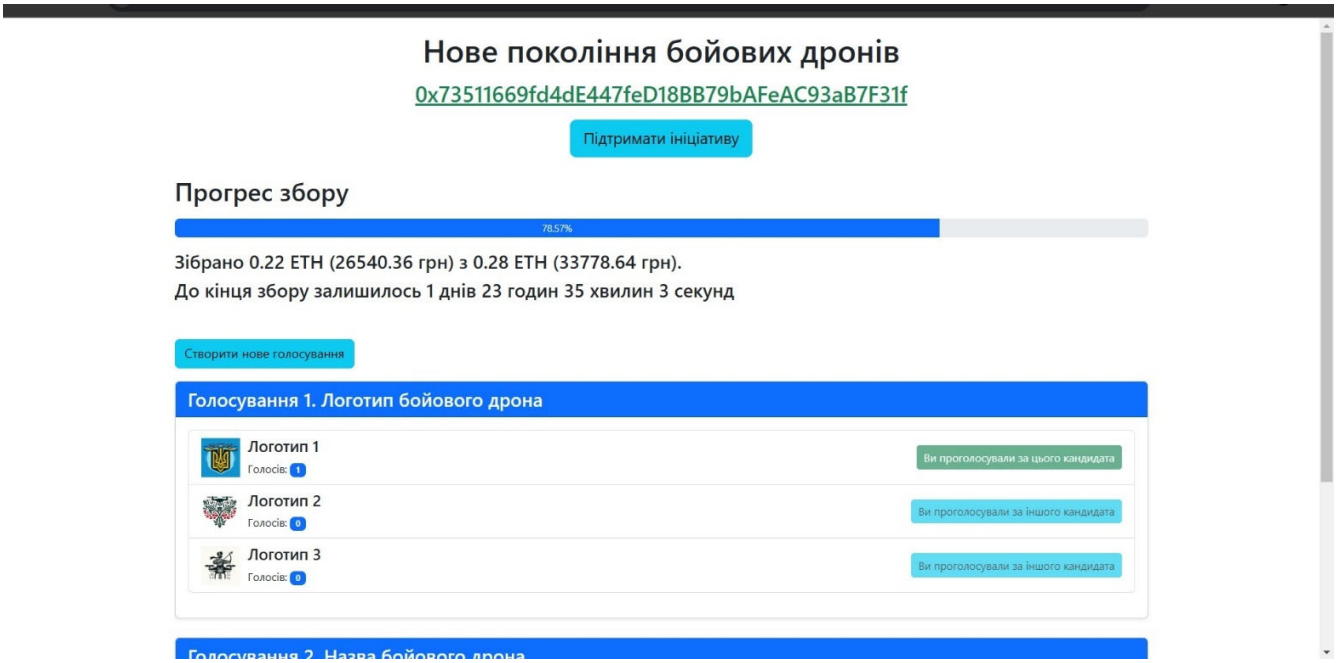

Рисунок 3.2 – Загальний вигляд фронтенд частини (збір на розробку дронів)

3.2.2 Компоненти застосунку

DonateButton – кнопка для внесення коштів, при натисканні на яку відображається модальне вікно із інформацією про внесок.

DonateModal – модальне вікно з інформацією про внесок коштів. Дозволяє користувачу ввести кількість ETH, скільки б він хотів внести та побачити скільки це буде в гривнях. Після підтвердження створюється нова транзакція в провайдері користувача, яка надсилає кошти на адресу контракту (рис. 3.3).

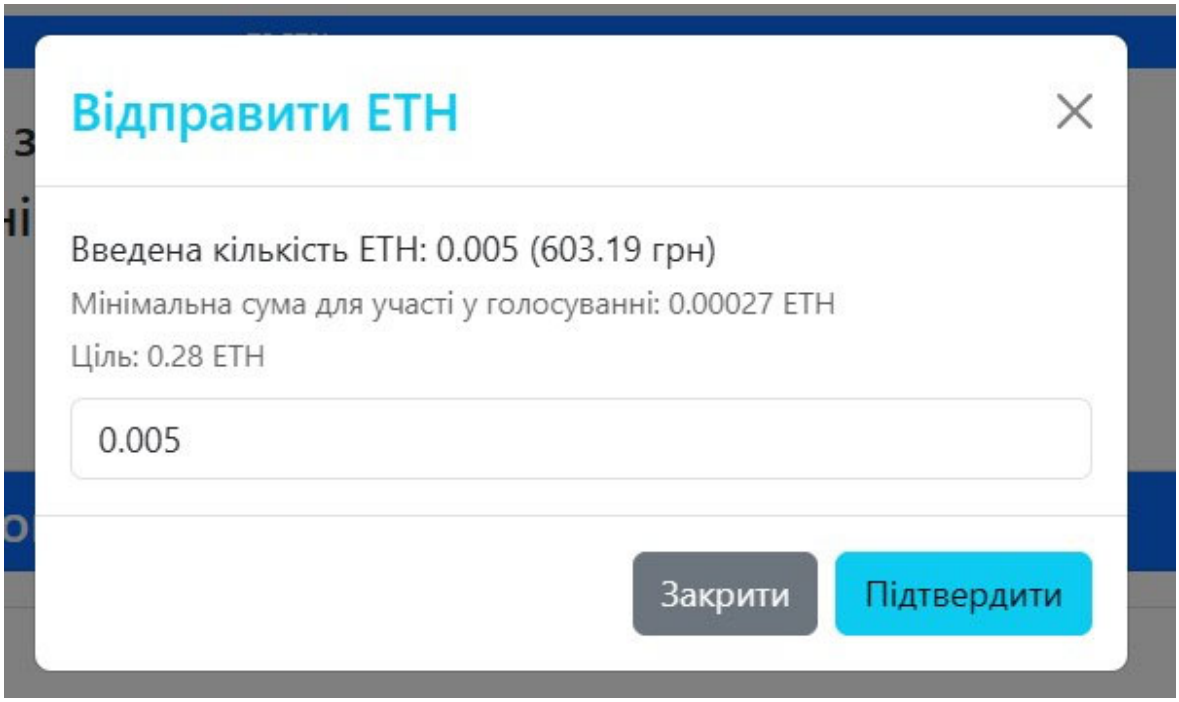

Рисунок 3.3 – Модальне вікно DonateModal

Компонент Progress – прогрес збору відображає індикатор виконання збору, також показує скільки вже зібрано та скільки всього необхідно зібрати в ETH та грн..

Компонент ProgressBar – індикатор виконання із налаштованою стилізацією.

Компонент CrowdfundingCountdown – зворотній відлік до завершення кампанії у днях, годинах, хвилинах та секунда. Якщо вона вже завершена, то відображається повідомлення про це.

Компонент CreateVotingButton – кнопка для створення голосування, доступна лише власнику кампанії. При натисканні на неї відкривається модальне вікно для створення голосування.

Компонент CreateVotingModal – модальне вікно для створення нового голосування (рис. 3.4). Дозволяє власнику вказати опис, час початку, час завершення, а також кандидатів. Після підтвердження створюється транзакція до контракту на блокчейні.

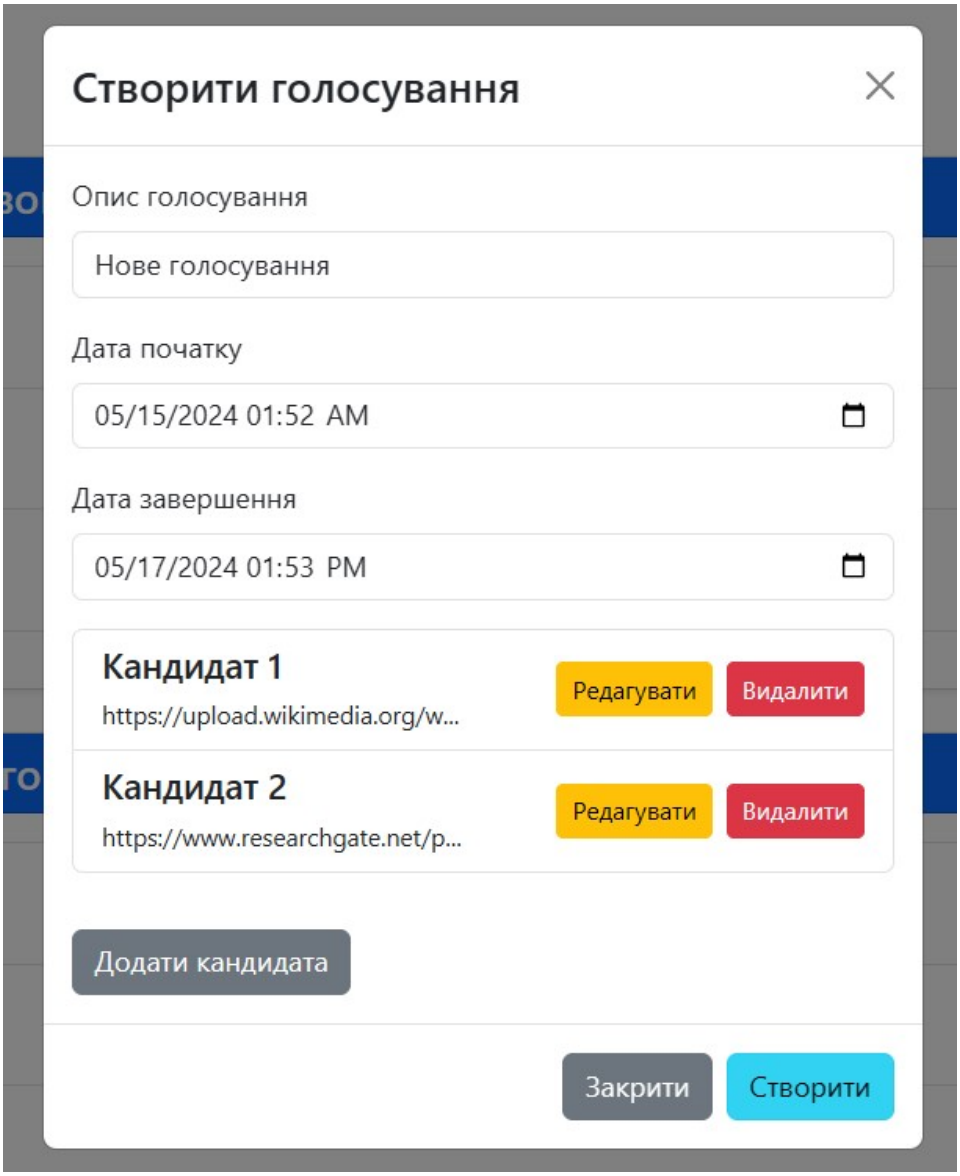

Рисунок 3.4 – Модальне вікно CreateVotingModal

Компонент VotingList – перелік всіх голосувань краудфандингу.

Компонент Voting – картка голосування (рис. 3.5 і 3.6). Заголовок містить опис голосування, а у тіло – список всіх кандидатів.

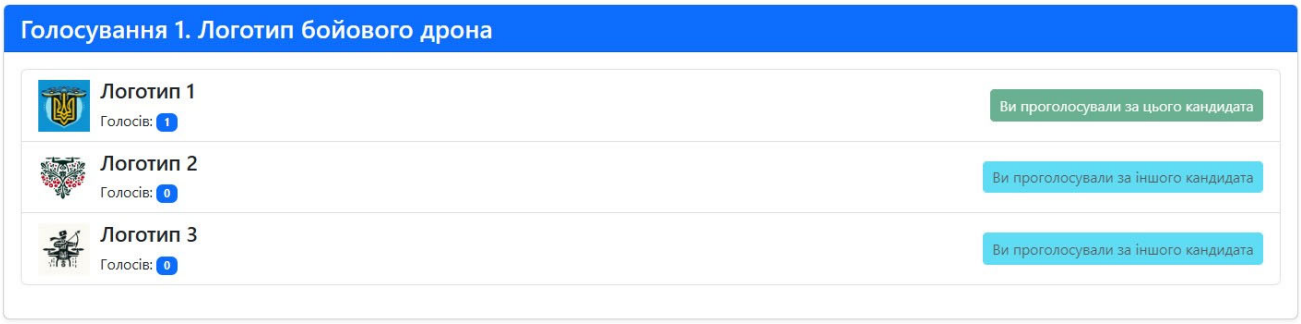

Рисунок 3.5 – Картка Voting після вибору кандидата

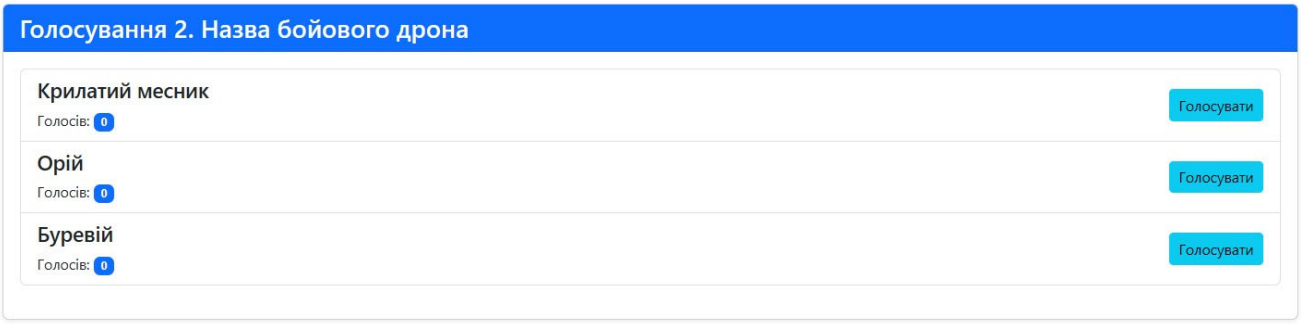

Рисунок 3.6 – Картка Voting до вибору кандидата

Компонент CandidateList – перелік кандидатів. Містить в собі карусель з кандидатів, а також кнопки для її навігації. Якщо користувач ще не вніс мінімальну суму для голосування, то при натисканні на кнопку “*Голосувати*” з’явиться модальне вікно із помилкою і повідомленням, що він ще не вніс достатньо коштів для голосування.

Компонент Candidate – картка кандидата. Містить зображення кандидата (якщо наявне). У заголовку вказано опис кандидату. Також містить кнопку для голосування за цього кандидату. Якщо користувач вже проголосував, то кнопка буде неактивна, і буде повідомляти про те, що він вже проголосував за цього або іншого кандидата. Ще містить загальну кількість голосів кандидата. Також є можливість переглянути зображення детальніше (рис. 3.7)

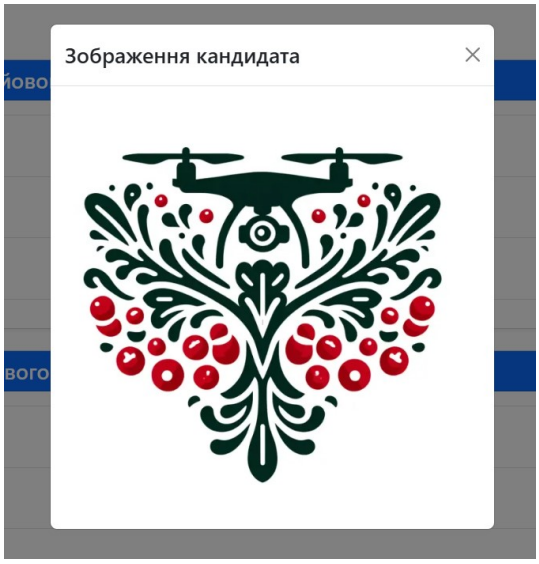

Рисунок 3.7 – Детальний перегляд зображення кандидата

3.3 Тестування децентралізованого застосунку

Тести для проєкту створені на платформі Hardhat мовою програмування TypeScript. Вона дозволяє легко розробляти ізольовані між собою тести. А також надає можливість локального запуску блокчейну, що значно пришвидшує процес тестування.

3.3.1 Тести контракту голосування

Спочатку йде перевірка створення голосування. В цьому тесті створюється голосування с трьома кандидатами, після чого переглядаються всі змінні програми, щоб впевнитись в їх правильності (рис. 3.8).

Наступний тест перевіряє, що тільки власник контракту (інший контракт) може викликати функцію vote і що вона працює належним чином. Для цього створюється новий контракт у мережі Ethereum. Далі перевіряється, що голос користувача було збережено в змінній votedNumber.

Наступний тест необхідний для підтвердження того, що звичайний користувач не може викликати функцію vote. В цьому випадку транзакція має відмінитися з помилкою “*OwnableUnauthorizedAccount*”.

Тести в блоці “Voting” перевіряють основний функціонал застосунку.

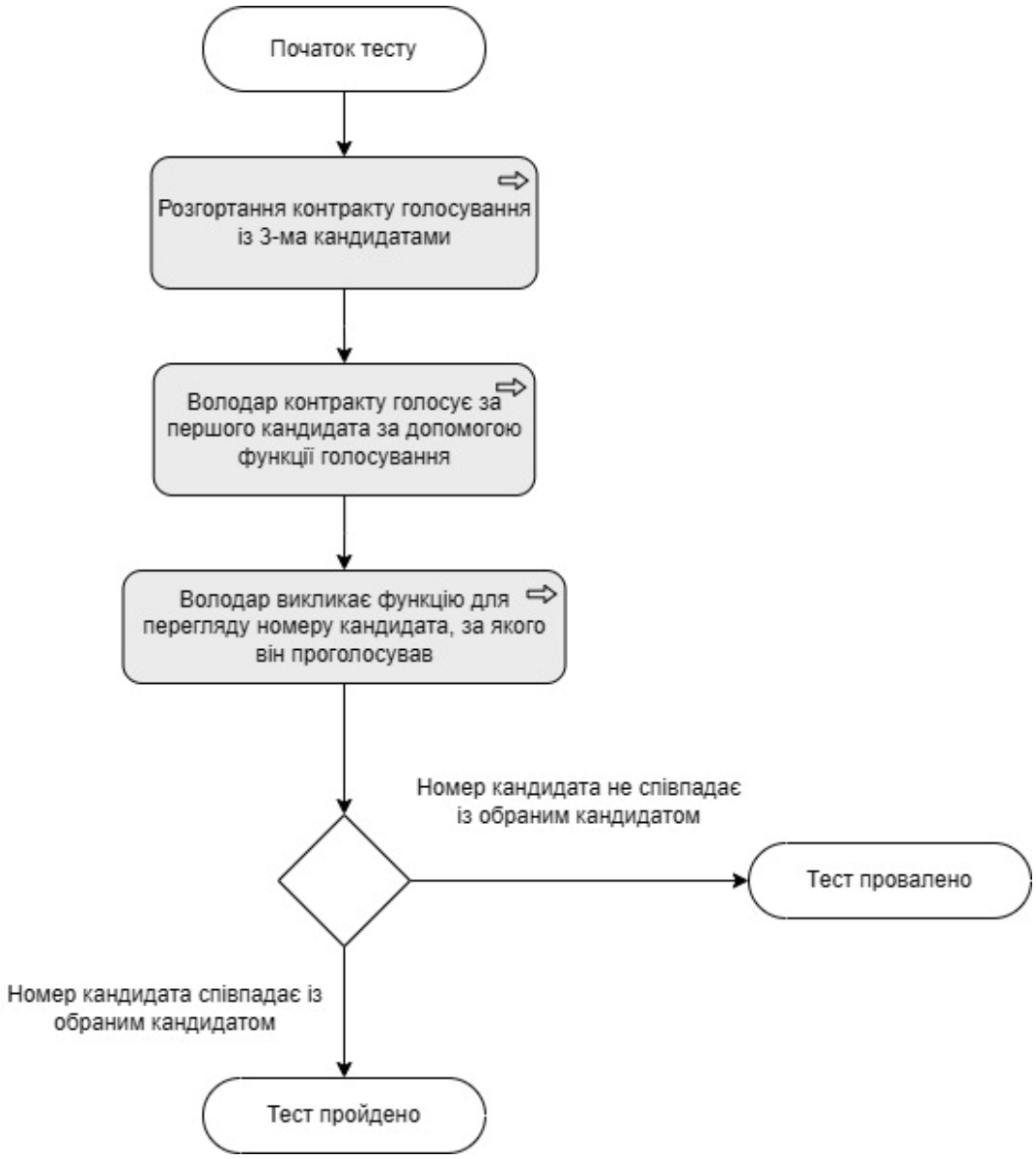

Рисунок 3.8 – UML-діаграма послідовності тестування для перевірки правильності голосування

Перший тест перевіряє, чи правильно нараховуються голоси. Він створює контракт з трьома кандидатами й перевіряє, що при віддачі голосу першому, його кількість голосів успішно збільшується на одиницю.

Другий тест перевіряє, чи не може один користувач голосувати декілька разів.

Для цього створюється контракт з трьома кандидати, користувач голосує за першого, після чого намагається проголосувати за нього ще раз. В цьому разі транзакція повинна провалитися з помилкою "*Already voted*".

Останній третій тест цього блока перевіряє, чи не може користувач голосувати, якщо голосування вже завершилося. Для цього створюється голосування, далі за допомогою інструментів hardhat час блокчейну збільшується до часу закінчення голосування. Далі йде спроба проголосувати, яка має видати помилку "*Voting is not active*".

3.3.2 Тести контракту краудфандингу

Спочатку йде тест на правильність створення контракту. Створюється контракт, після чого перевіряються всі його змінні.

Далі йде блок тестів, що перевіряє можливості володаря контракту.

Перший тест викликає функцію setVotingThreshold, та перевіряє що значення відповідної змінної встановлюється коректно.

Другий тест перевіряє, що володар контракту, має змогу створювати нове голосування. Для цього після створення контракту, з аккаунту його володаря викликається функція createVoting з необхідними параметрами. Далі переглядається адреса створеного голосування, вона не повинна дорівнювати нульовій адресі.

Далі йде блок тестів для перевірки функціональності виведення коштів з контракту. Перший тест перевіряє, що володар не може вивести кошти, якщо кампанія ще не завершилася. Для цього створюється голосування, після чого з аккаунту володаря одразу викликається функція withdraw, що повинно викликати помилку "*CrowdFunding is not finished yet*".

Другий тест перевіряє, що володар не може вивести кошти, якщо краудфандинг провалився. Після створення контракту часова мітка блокчейну збільшується до значення часу його завершення. Викликається функція withdraw, що повинно видати помилку "*CrowdFunding failed*" (рис. 3.9).

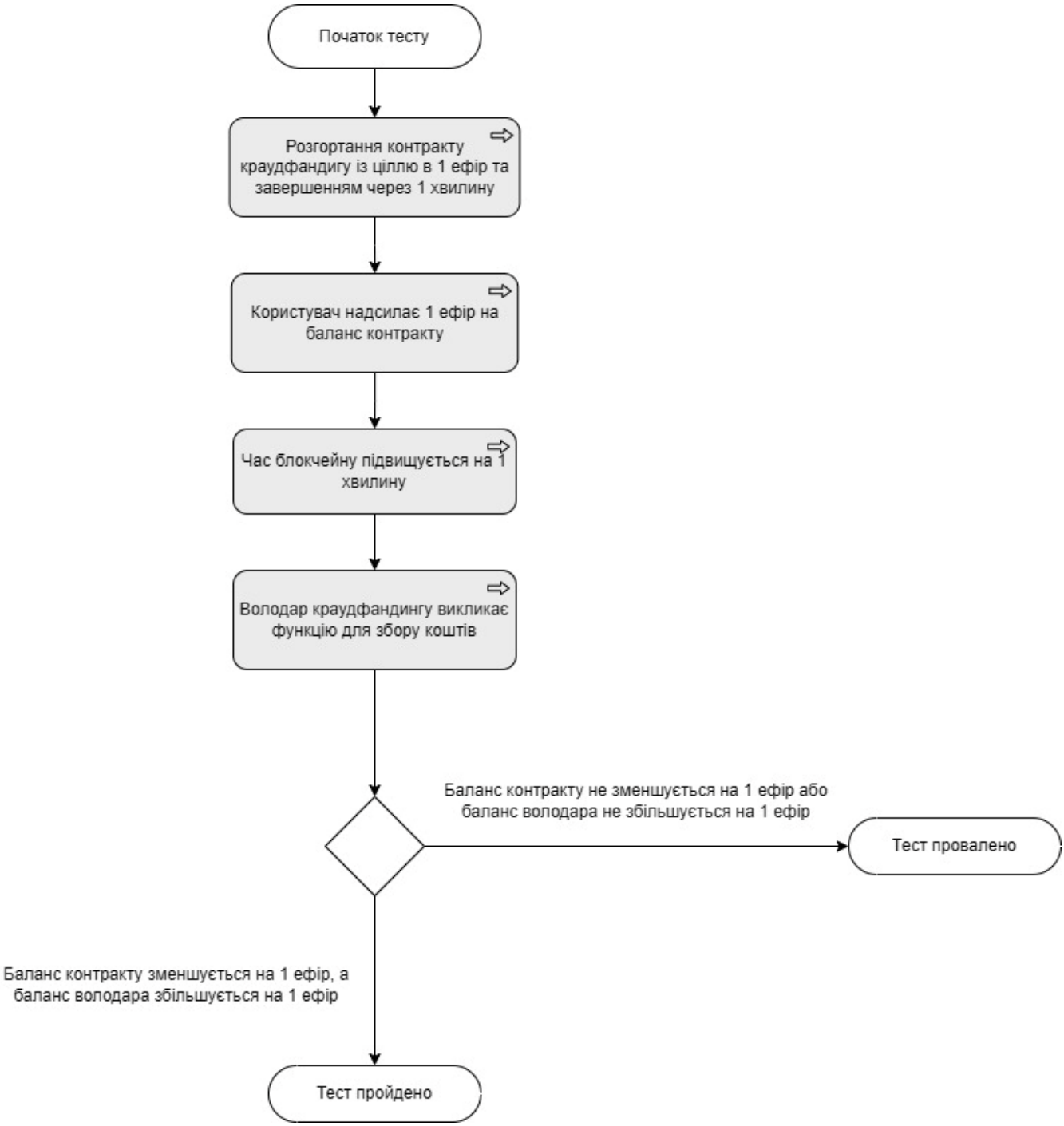

Рисунок 3.9 – UML-діаграма послідовності третього тесту

Третій тест перевіряє, що володар може вивести кошти у разі успішного завершення кампанії. Створюється новий контракт з ціллю в 1 ефір, на його адресу

надсилаються кошти в розмірі 1 ефір, час блокчейну підвищується до завершення краудфандингу. Далі з аккаунту володаря викликається withdraw, що повинно призвести переведення всіх коштів з контракту на адресу його володаря (рис. 3.7).Далі йде блок тестів, що перевіряють можливості звичайних користувачів.

Перший тест перевіряє, що користувач не може редагувати мінімальну необхідну суму для голосування. Створюється контракт, користувач викликає функцію setVotingThreshold. Це повинно призвести до помилки "*OwnableUnauthorizedAccount*".

Другий тест перевіряє, що користувач не може створювати нові голосування. Створюється контракт, користувач викликає функцію createVoting з необхідними параметрами. Це має призвести до помилки "*OwnableUnauthorizedAccount*".

Третій тест перевіряє, що користувач не може вивести кошти з контракту. Створюється контракт, користувач викликає функцію withdraw. Це має призвести до помилки "*OwnableUnauthorizedAccount*".

Наступний блок тестів перевіряє основний функціонал застосунку.

Перший тест перевіряє, що контракт зберігає інформацію про внесені кошти. Створюється контракт, на його адресу користувач надсилає кошти в розмірі 0.5 ефіру. Це має правильно зберегти в змінну contribution за адресою користувача кількість отриманих коштів.

Другий тест перевіряє, що контракт не повинен отримувати кошти, якщо кампанія провалилася. Створюється контракт, час блокчейну підвищується до закінчення контракту, на адресу краудфандингу надсилається 0.5 ефіру. Це повинно призвести до помилки "*CrowdFunding failed*".

Далі йде тест, який перевіряє, що користувач не може голосувати, якщо він не вніс достатньо коштів. Створюється контракт, де мінімальна сума для голосування дорівнює 0.00027 ефіру, після чого користувач намагається викликати функцію vote з необхідними параметрами. Це має призвести до помилки "*Voting threshold not met*".

Наступний блок тестів перевіряє можливість повернення коштів з краудфандингу.

Перший тест перевіряє, що користувач без жодних внесків не може повернути собі кошти. Створюється контракт, користувач намагається викликати функцію refund. Це повинно призвести до помилки "*No contributions*".

Другий тест перевіряє, що користувач не може повернути собі кошти, якщо кампанія не провалилася. Створюється контракт, користувач надсилає на його адресу 1 ефір, після чого намагається викликати функцію refund. Це повинно призвести до помилки "*CrowdFunding not failed*".

Третій тест перевіряє, що користувач може повернути собі кошти, якщо кампанія провалилася. Створюється контракт з ціллю в 1 ефір, користувач надсилає на його адресу 0.9 ефіру, час блокчейну підвищується до закінчення контракту, після чого він викликає функцію refund. Це повинно правильно змінити баланси контракту та користувача на суму внеску.

3.4 Переваги створеної системи електронного голосування

У рамках цього дослідження було розглянуто розробку і впровадження системи електронного голосування на основі блокчейн-технологій, яка вирішує численні виклики, що стосуються безпеки, прозорості та надійності виборчих процесів. Реалізація такої системи має ряд значних переваг, які відображають важливість та ефективність запроваджених технологічних рішень.

Забезпечення високого рівня безпеки. Використання блокчейн-технології дозволяє забезпечити незмінність голосів та їх захист від зовнішніх та внутрішніх спроб маніпуляцій. Смарт-контракти створюють безпечне середовище, де кожен голос реєструється, верифікується та зберігається без можливості його зміни.

Прозорість виборчого процесу. Прозорість забезпечується завдяки публічному характеру блокчейну, що дозволяє всім учасникам виборів перевіряти хід голосування

та підрахунку голосів в реальному часі. Це значно підвищує довіру до системи голосування, зменшуючи можливості для сумнівів чи звинувачень у фальсифікаціях.

Надійність. Система проєктується з урахуванням високих стандартів надійності. Автоматизовані процеси, що керуються смарт-контрактами, мінімізують людські помилки та забезпечують стабільність та ефективність виборчих процесів.

Доступність. Система дозволяє залучати ширший круг виборців, зокрема людей, які через фізичні обмеження або віддаленість від виборчих дільниць раніше не могли взяти участь у голосуванні. Інтерфейс системи розробляється таким чином, щоб бути зрозумілим і зручним для користувачів будь-якого віку та рівня технологічної освіченості.

Розробка і впровадження системи електронного голосування на базі блокчейн не тільки вирішує багато наявних проблем, але й відкриває нові можливості для зміцнення демократичних процесів. Це стає особливо актуальним у контексті сучасних викликів, пов'язаних із забезпеченням цілісності та надійності виборів.

3.5 Висновки до розділу 3

Цей розділ детально описує процес реалізації децентралізованого застосунку для системи голосування, який включає в себе розробку смарт-контрактів та фронтенду. Здійснена інтеграція краудфандингових і голосувальних модулів забезпечує динамічну взаємодію між користувачами та контрактами, що є критичною для успішної роботи системи.

Два основні смарт-контракти (Voting і CrowdFunding) забезпечують весь необхідний функціонал для проведення краудфандингу та голосувань. Використання Solidity для написання контрактів дозволило точно визначити умови та правила роботи системи, що забезпечує прозорість та справедливість процесів. Контракт Voting керує процесом голосування, забезпечуючи реєстрацію голосів і збереження інформації про голосування. Контракт CrowdFunding контролює умови участі в

голосуванні, включаючи збір коштів та встановлення порогів для участі, що створює мотиваційний механізм для залучення фінансування.

Реалізовані механізми блокування та перевірки стану кампаній забезпечують захист від вразливостей, таких як reentrancy атаки, і гарантують правильність функціонування контрактів навіть у випадку непередбачених ситуацій.

Інтенсивне тестування обох модулів на платформі Hardhat з використанням TypeScript забезпечило перевірку всіх аспектів роботи контрактів, від базових функцій до складних сценаріїв взаємодії. Це підтверджує стабільність системи та її готовність до впровадження у реальних умовах.

Фронтенд, розроблений з використанням React і TypeScript, надає зручний та інтуїтивно зрозумілий користувацький інтерфейс, який забезпечує легке взаємодія з системою для кінцевих користувачів. Інтеграція з бібліотекою Bootstrap додала візуальну привабливість та адаптивність інтерфейсу.

Реалізація децентралізованої системи голосування на основі блокчейну відкриває нові можливості для проведення прозорих, справедливих і безпечних виборів та голосувань у різних сферах. Описана система не тільки сприяє підвищенню довіри користувачів до електронних виборів, але й демонструє можливість її масштабування і адаптації під різні умови та вимоги.

ВИСНОВКИ

У рамках дослідження було розроблено систему електронного голосування на базі блокчейн-мережі Ethereum, яка підходить для застосування в різних сферах, включаючи муніципальні та національні вибори. Система забезпечує високий рівень безпеки, прозорість результатів та можливість перевірки даних будь-яким учасником голосування, що робить процес максимально відкритим і чесним.

Розроблена система голосування має два основних компоненти: 1) смарт-контракти для обробки голосів і забезпечення відповідних умов участі у голосуванні; 2) користувацький інтерфейс, що дозволяє виборцям з легкістю взаємодіяти з системою через веббраузер або мобільний застосунок.

Інтерфейс, розроблений на основі React, забезпечує зручність та інтуїтивну простоту використання, що є критично важливим для залучення широкої аудиторії користувачів.

Для забезпечення безпеки голосування в створюваній системі організовано кілька рівнів захисту: криптографічне шифрування транзакцій, механізми відновлення та верифікації голосів, а також передові методи запобігання зловмисним атакам на мережу. Система має механізми для аудиту та перевірки результатів голосування, які можуть бути використані спостерігачами від різних політичних партій та незалежними організаціями для забезпечення чесності виборчого процесу. Такий підхід значно підвищує довіру до системи голосування та є фундаментальним принципом для підтримки демократичних ідей.

У першому розділі роботи розглянуто розробку і впровадження системи електронного голосування на основі блокчейн-технологій, яка вирішує численні виклики, що стосуються безпеки, прозорості та надійності виборчих процесів. Реалізація такої системи має ряд значних переваг, які відображають важливість та ефективність запроваджених технологічних рішень. З’ясовано, що розробка і впровадження системи електронного голосування на базі блокчейн не тільки вирішує

багато наявних проблем, але й відкриває нові можливості для зміцнення демократичних процесів. Це стає особливо актуальним у контексті сучасних викликів, пов'язаних із забезпеченням цілісності та надійності виборів.

Другий розділ роботи присвячений детальному аналізу та оцінці використаних технологічних рішень у контексті блокчейну Ethereum, мови програмування Solidity та розробки децентралізованих застосунків (dApps). В цьому розділі розглянуто ключові аспекти, що визначають вибір блокчейну Ethereum як платформи для реалізації системи електронного голосування, включаючи його здатність до виконання Тьюринг-повних смарт-контрактів, що дозволяє створювати високоадаптивні та надійні програмні рішення. Основна увага у розділі приділяється технічному забезпеченню безпеки, масштабування та інтеграції з іншими системами і технологіями. Проаналізовано переваги та потенційні недоліки використання блокчейну та шляхи їх нейтралізації. Значну увагу в розділі приділено оптимізації виконання смарт-контрактів, що є критично важливим для забезпечення швидкості обробки голосів та низької вартості взаємодії користувачів з системою. Розглянуто різні підходи до оптимізації, а також техніки шаблонів проєктування для розробки смарт-контрактів.

У третьому розділі роботи описано специфіку реалізації системи електронного голосування через розробку і впровадження смарт-контрактів і фронтенд-інтерфейсу, що дозволяє користувачам взаємодіяти з системою в інтуїтивно зрозумілий та зручний спосіб. Описано структуру і логіку роботи основних компонентів системи, їх взаємодію та методи забезпечення безпеки при обробці голосів. Детально проаналізовано роботу двох основних смарт-контрактів – Voting і CrowdFunding. Розглянуто механізми голосування, збору коштів, а також умови, за якими користувачі можуть брати участь у голосуванні. Також описано засоби захисту від потенційних атак на прикладі атак повторного введення (reentrancy). Значну частину розділу займає опис процесу тестування реалізованих контрактів. Використання тестового середовища Hardhat і написання тестів у TypeScript дозволили перевірити різні аспекти роботи системи: від функціональності окремих функцій до взаємодії

користувачів з контрактами на різних етапах виборчого процесу. За результатами тестування зроблено висновки про високу надійність і безпеку системи. Окремо в цьому розділі висвітлено розробку користувацького інтерфейсу на базі React і TypeScript, що забезпечує динамічне і зручне відображення інформації про хід голосування, результати та управління особистими даними. Інтерфейс розроблено так, щоб користувач міг легко здійснювати всі необхідні дії від внесення коштів на баланс краудфандингу до участі у голосуванні та перегляду результатів. Загалом цей розділ розглядає практичну реалізацію та функціонування системи електронного голосування, що становить основу для її подальшого впровадження і масштабування.

Створена у роботі програмна система дозволяє забезпечити високу безпеку та прозорість голосування, завдяки використанню блокчейн-технологій. Через інтеграцію смарт-контрактів з вебінтерфейсом, система здатна оперативно обробляти дані виборців, реєструвати кожен голос як незмінний запис у блокчейні, що неможливо підробити або змінити без відома інших учасників мережі. Система здатна обслуговувати велику кількість користувачів одночасно, при цьому зберігаючи низькі витрати на транзакції і обмін даними завдяки оптимізації смарт-контрактів і використанню ефективних методів голосування, таких як голосування з мінімальними витратами газу. Подальший розвиток системи може включати інтеграцію з іншими блокчейн-платформами, що підтримують смарт-контракти, для забезпечення ще більшої універсальності і безпеки.

Результати дослідження показали, що застосування блокчейн-технологій у сфері електронного голосування має значний потенціал для розвитку, зокрема, за рахунок подальшого вдосконалення технологій розподіленого реєстру та їх інтеграції з сучасними інформаційними системами. Подальше впровадження розробленої системи сприятиме збільшенню прозорості та зменшенню випадків виборчих фальсифікацій. Є широкі перспективи для масштабування та адаптації системи до різних культурних та політичних умов, надаючи можливість для глобального використання електронного голосування як надійного та безпечного інструмента демократії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Berenjestanaki M., Barzegar H., Ioini N., Pahl C. Blockchain-Based E-Voting Systems: A Technology Review. *Electronics*. 2024. Vol. 13(1), No 17. P. 1-38. DOI: 10.3390/electronics13010017. URL: https://www.mdpi.com/2079-9292/13/1/17.

2. Narayanan A., Bonneau J., Felten E. Bitcoin and cryptocurrency technologies: a comprehensive introduction. Princeton: Princeton University Press, 2016. 336 p. ISBN 978-0-691-17169-2.

3. Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. 9 с. URL: https://bitcoin.org/bitcoin.pdf.

4. Що таке блокчейн? Пояснюємо простими словами. URL: https://blog.whitebit.com/uk/what-is-blockchain-technology/

5. What are the benefits of blockchain? URL: https://www.ibm.com/topics/benefits-of-blockchain

6. Roose K. What are DAOs? URL: https://www.nytimes.com/interactive/2022/03/18/technology/what-are-daos.html

7. Fan Y., Zhang L., Wang R., Imran M. Insight into Voting in DAOs: Conceptual Analysis and A Proposal for Evaluation Framework. *IEEE Network*. 05 June 2023. DOI: 10.1109/MNET.137.2200561. URL: https://ieeexplore.ieee.org/document/10144510

8. Verma G. A Secure Framework for E-Voting Using Blockchain. *Second International Conference on Computer Science, Engineering and Applications (ICCSEA)*. 8 September 2022. DOI: 10.1109/ICCSEA54677.2022.9936073. URL: https://ieeexplore.ieee.org/document/9936073

9. Nnebe S., Okafor C., Tochukwu O. Design and Implementation of Decentralized *Voting System on the Ethereum Blockchain. International Journal of Computer (IJC)*. 2022. Vol. 45(1). P. 95-104. URL: https://ijcjournal.org/index.php/InternationalJournalOfComputer/article/view/1970.

10. Khoury D., Kfoury E., Harb H. Decentralized Voting Platform Based on Ethereum Blockchain. *IEEE International Multidisciplinary Conference on Engineering Technology (IMCET'18).* 4-16 November 2018. P. 224-229. DOI: 10.1109/IMCET.2018.8603050. URL: https://ieeexplore.ieee.org/document/8603050.

11. Youssef F., Javier A., Samer H. A comparative analysis of the platforms for decentralized autonomous organizations in the Ethereum blockchain. *Journal of Internet Services and Applications*. 2021. Vol. 12. P. 1-20. DOI: 10.1186/s13174-021-00139-6. URL: https://jisajournal.springeropen.com/articles/10.1186/s13174-021-00139-6

12. Faqir-Rhazoui Y., Gallardo J., Hassan S. A Comparative Analysis of the Platforms for Decentralized Autonomous Organizations in the Ethereum Blockchain. 2021. P. 1-21. URL: https://www.researchgate.net/publication/349034784_A_Comparative_Analysis_of_the_Platforms_for_Decentralized_Autonomous_Organizations_in_the_Ethereum_Blockchain/citations

13. DAOstack end-of-road update, and the Common Open Development program announcement. URL: https://daostack.notion.site/DAOstack-end-of-road-update-and-the-Common-Open-Development-program-announcement-b808c3bf9e3a4b47bd42b7a67da707b4

14. Jyotirmoy B. Understanding the advantages and disadvantages of DAO voting schemes. URL: https://jyotirmoy.dev/understanding-the-advantages-and-disadvantages-of-dao-voting-schemes

15. Solidity. URL: https://docs.soliditylang.org/en/v0.8.25/

16. What is Solidity Syntax? URL: https://docs.alchemy.com/docs/what-is-solidity-syntax

17. Shelar V. Evolving with Solidity: A Journey Through Major Version Updates. URL: https://www.linkedin.com/pulse/evolving-solidity-journey-through-major-version-updates-viraj-shelar-5zjxf/

18. Basiuk V., Adams A. Role of Solidity in Blockchain Technology. URL: https://evacodes.com/blog/role-of-solidity-in-blockchain

19. What are smart contracts on blockchain? URL: https://www.ibm.com/topics/smart-contracts

20. What do “State” and “State Change” Mean in Blockchain? URL: https://www.nervos.org/knowledge-base/state_and_state_change_(explainCKBot)

21. Gas (Ethereum): How Gas Fees Work on the Ethereum Blockchain. URL: https://www.investopedia.com/terms/g/gas-ethereum.asp

22. Shubhani A., Neeraj K. Chapter Three – Hashes. Cryptographic primitives used in blockchain. *Elsevier*. 2021. Vol. 121. P. 83-93. URL: https://doi.org/10.1016/bs.adcom.2020.08.003.

23. Reiff N. Decentralized Autonomous Organization (DAO): Definition, Purpose, and Example. URL: https://www.investopedia.com/tech/what-dao/

24. What Is a DEX (Decentralized Exchange)?. URL: https://chain.link/education-hub/what-is-decentralized-exchange-dex

25. Submitting your Smart Contract to Etherscan. URL: https://docs.alchemy.com/docs/submitting-your-smart-contract-to-etherscan

26. React. URL: https://react.dev/learn

ДОДАТКИ

Додаток А. Фрагменти програмного коду

contracts > Voting.sol

```
1   // SPDX-License-Identifier: MIT
2   pragma solidity ^0.8.19;
3
4   import "@openzeppelin/contracts/access/Ownable.sol";
5
6   contract Voting is Ownable {
7       struct Candidate {
8           string description;
9           string imageLink;
10          uint32 votes;
11      }
12
13      string public description;
14      uint public startTimestamp;
15      uint public endTimestamp;
16      uint public createdTimestamp;
17      mapping(address => uint) public votedNumber;
18      Candidate[] public candidates;
19
20      constructor(
21          string memory _description,
22          uint _startTimestamp,
23          uint _endTimestamp,
24          string[] memory candidatesDescriptions,
25          string[] memory candidatesImages
26      ) Ownable(msg.sender) {
27          description = _description;
28          startTimestamp = _startTimestamp;
29          endTimestamp = _endTimestamp;
30          createdTimestamp = block.timestamp;
31          for (uint i = 0; i < candidatesDescriptions.length; i++) {
32              candidates.push(
33                  Candidate({
34                      description: candidatesDescriptions[i],
35                      imageLink: candidatesImages[i],
36                      votes: 0
37                  })
38              );
39          }
40      }
```

contracts > CrowdFunding.sol

```
1   // SPDX-License-Identifier: MIT
2   pragma solidity ^0.8.19;
3
4   import "@openzeppelin/contracts/access/Ownable.sol";
5   import "./Voting.sol";
6
7   contract CrowdFunding is Ownable {
8       uint public goal;
9       uint public startTimestamp;
10      uint public endTimestamp;
11      uint public votingThreshold;
12      bool private locked;
13      Voting[] public votings;
14      mapping(address => uint) public contributions;
15
16      constructor(
17          uint _startTimestamp,
18          uint _endTimestamp,
19          uint _votingThreshold,
20          uint _goal
21      ) Ownable(msg.sender) {
22          startTimestamp = _startTimestamp;
23          endTimestamp = _endTimestamp;
24          votingThreshold = _votingThreshold;
25          goal = _goal;
26      }
27
28      function setVotingThreshold(uint amount) external onlyOwner {
29          votingThreshold = amount;
30      }
31
32      function createVoting(
33          string memory _description,
34          uint _startTimestamp,
35          uint _endTimestamp,
36          string[] memory candidatesDescriptions,
37          string[] memory candidateImages
38      ) external onlyOwner {
39          votings.push(
40              new Voting(_description, _startTimestamp, _endTimestamp, candidatesDescriptions, candidateImages)
41          );
42      }
43
44      function withdraw() external payable onlyOwner {
45          require(
46              block.timestamp >= endTimestamp,
```

contracts > CrowdFunding.sol

```
7   contract CrowdFunding is Ownable {
38      ) external onlyOwner {
42      }
43
44      function withdraw() external payable onlyOwner {
45          require(
46              block.timestamp >= endTimestamp,
47              "CrowdFunding is not finished yet"
48          );
49          require(address(this).balance >= goal, "CrowdFunding failed");
50          address ownerAddress = payable(owner());
51          (bool sent, ) = ownerAddress.call{value: address(this).balance}("");
52          require(sent, "Funds wasn't withdrawn");
53      }
54
55      function getVotingsCount() public view returns (uint) {
56          return votings.length;
57      }
58
59      function vote(uint votingIndex, uint candidateIndex) external {
60          require(
61              contributions[msg.sender] >= votingThreshold,
62              "Voting threshold not met"
63          );
64          votings[votingIndex].vote(msg.sender, candidateIndex);
65      }
66
67      function refund() external payable {
68          require(!locked, "Reentrancy guard");
69          require(contributions[msg.sender] > 0, "No contributions");
70          require(
71              checkFailed(),
72              "CrowdFunding not failed"
73          );
74          locked = true;
75          (bool sent, ) = msg.sender.call{value: contributions[msg.sender]}("");
76          if (sent) contributions[msg.sender] = 0;
77          locked = false;
78      }
79
80      function checkFailed() public view returns (bool) {
81          return block.timestamp >= endTimestamp && address(this).balance < goal;
82      }
83
```

test > Voting.ts > ...

```ts
import {
  loadFixture,
  time,
} from "@nomicfoundation/hardhat-toolbox/network-helpers";
import { expect } from "chai";
import { ethers } from "hardhat";

describe("Voting", function () {
  async function deploy(
    description: string,
    startTimestamp: bigint,
    endTimestamp: bigint,
    candidatesDescriptions: string[],
    candidatesImages: string[]
  ) {
    const [owner, account2] = await ethers.getSigners();

    const Voting = await ethers.getContractFactory("Voting");
    const contract = await Voting.deploy(
      description,
      startTimestamp,
      endTimestamp,
      candidatesDescriptions,
      candidatesImages
    );
    await contract.waitForDeployment();

    return { contract, owner, account2 };
  }

  async function deployVotingFixture() {
    const description = "Description";
    const startTimestamp = BigInt(await time.latest());
    const endTimestamp = startTimestamp + 60n;
    const candidatesDescriptions = ["Candidate1", "Candidate2", "Candidate3"];
    const candidatesImages = ["link1", "link2", "link3"];

    const contractConstructor = {
      description,
      startTimestamp,
      endTimestamp,
      candidatesDescriptions,
      candidatesImages,
    };
    const deployed = await deploy(
      description,
```

test > Voting.ts > ...

```
8   describe("Voting", function () {
31    async function deployVotingFixture() {
38      const contractConstructor = {
43        candidatesImages,
44      };
45      const deployed = await deploy(
46        description,
47        startTimestamp,
48        endTimestamp,
49        candidatesDescriptions,
50        candidatesImages
51      );
52      return { contractConstructor, ...deployed };
53    }
54
55    describe("Deployment", async () => {
56      it("Should correctly start voting on deployment", async () => {
57        const { contractConstructor, contract, owner, account2 } =
58          await loadFixture(deployVotingFixture);
59        const contractStartTimestamp = await contract.startTimestamp();
60        const contractEndTimestamp = await contract.endTimestamp();
61        const contractCandidates = await Promise.all(
62          [0, 1, 2].map((ind) => contract.candidates(ind))
63        );
64
65        expect(contractStartTimestamp).to.eq(contractConstructor.startTimestamp);
66        expect(contractEndTimestamp).to.eq(contractConstructor.endTimestamp);
67        expect(contractStartTimestamp).to.eq(contractConstructor.startTimestamp);
68        expect(contractCandidates.map((c) => c.description)).all.members(
69          contractConstructor.candidatesDescriptions
70        );
71        expect(contractCandidates.map((c) => c.imageLink)).all.members(
72          contractConstructor.candidatesImages
73        );
74      });
75    });
76
77    describe("Owner", async () => {
78      it("Should be able to vote", async () => {
79        const { contract, owner } = await loadFixture(deployVotingFixture);
80        const ownerAddress = await owner.getAddress();
81        await contract.connect(owner).vote(ownerAddress, 0);
82        const votedFor = await contract.votedNumber(ownerAddress);
83        expect(votedFor).to.be.eq(1);
84      });
```

test > Voting.ts > ...

```
8   describe("Voting", function () {
86
87    describe("Non owner", async () => {
88      it("Should not be able to vote", async () => {
89        const { contract, account2 } = await loadFixture(deployVotingFixture);
90        const nonOwnerAddress = await account2.getAddress();
91        await expect(
92          contract.connect(account2).vote(nonOwnerAddress, 0)
93        ).to.be.revertedWithCustomError(contract, "OwnableUnauthorizedAccount");
94      });
95    });
96
97    describe("Voting", async () => {
98      it("Should correctly score votes", async () => {
99        const { contract, owner } = await loadFixture(deployVotingFixture);
100       const ownerAddress = await owner.getAddress();
101       await contract.connect(owner).vote(ownerAddress, 0);
102       const candidate = await contract.candidates(0);
103       expect(candidate.votes).to.be.eq(1);
104     });
105
106     it("Should not allow to vote multiple times", async () => {
107       const { contract, owner } = await loadFixture(deployVotingFixture);
108       const ownerAddress = await owner.getAddress();
109       await contract.connect(owner).vote(ownerAddress, 0);
110       await expect(
111         contract.connect(owner).vote(ownerAddress, 0)
112       ).to.be.revertedWith("Already voted");
113     });
114
115     it("Should not allow to vote if voting is finished", async () => {
116       const { contract, owner, contractConstructor } = await loadFixture(
117         deployVotingFixture
118       );
119       const ownerAddress = await owner.getAddress();
120       await time.increaseTo(contractConstructor.endTimestamp);
121       await expect(
122         contract.connect(owner).vote(ownerAddress, 0)
123       ).to.be.revertedWith("Voting is not active");
124     });
125   });
126 });
127
```

test > CrowdFunding.ts > ...

```
import {
  loadFixture,
  time,
} from "@nomicfoundation/hardhat-toolbox/network-helpers";
import { expect } from "chai";
import { ethers } from "hardhat";
import { ZeroAddress, parseEther } from "ethers";

describe("CrowdFunding", () => {
  async function deploy(
    startTimestamp: bigint,
    endTimestamp: bigint,
    votingThreshold: bigint,
    goal: bigint
  ) {
    const [owner, account2] = await ethers.getSigners();

    const CrowdFunding = await ethers.getContractFactory("CrowdFunding");
    const contract = await CrowdFunding.deploy(
      startTimestamp,
      endTimestamp,
      votingThreshold,
      goal
    );
    await contract.waitForDeployment();
    const contractAddress = await contract.getAddress();

    return { contractAddress, contract, owner, account2 };
  }

  async function deployCrowdfundingFixture() {
    const startTimestamp = BigInt(await time.latest());
    const endTimestamp = startTimestamp + 60n * 1_000n;
    const votingThreshold = parseEther("0.00027"); // approximately 1 usd
    const goal = parseEther("1");
    const deployed = await deploy(
      startTimestamp,
      endTimestamp,
      votingThreshold,
      goal
    );
    const contractConstructor = {
      startTimestamp,
      endTimestamp,
      votingThreshold,
      goal,
```

test > CrowdFunding.ts > ...

```
9   describe("CrowdFunding", () => {
31    async function deployCrowdfundingFixture() {
42      const contractConstructor = {
43        startTimestamp,
44        endTimestamp,
45        votingThreshold,
46        goal,
47      };
48      return { contractConstructor, ...deployed };
49    }
50
51    async function getVotingConstructor() {
52      const description = 'Description';
53      const startTimestamp = BigInt(await time.latest());
54      const endTimestamp = startTimestamp + 60n;
55      const candidatesDescriptions = ["Candidate1"]
56      const candidateImages = ["image.link"]
57      return { startTimestamp, endTimestamp, candidatesDescriptions, candidateImages, description};
58    }
59
60    describe("Deployment", async () => {
61      it("Should correctly start crowdfunding on deployment", async () => {
62        const { contractConstructor, contract } = await loadFixture(
63          deployCrowdfundingFixture
64        );
65        const contractStartTimestamp = await contract.startTimestamp();
66        const contractEndTimestamp = await contract.endTimestamp();
67        const contractVotingThreshold = await contract.votingThreshold();
68        const contractGoal = await contract.goal();
69
70        expect(contractStartTimestamp).to.eq(contractConstructor.startTimestamp);
71        expect(contractEndTimestamp).to.eq(contractConstructor.endTimestamp);
72        expect(contractVotingThreshold).to.eq(
73          contractConstructor.votingThreshold
74        );
75        expect(contractGoal).to.eq(contractConstructor.goal);
76      });
77    });
78
79    describe("Owner", async () => {
80      it("Should be able to set votingThreshold", async () => {
81        const { contract, owner } = await loadFixture(deployCrowdfundingFixture);
82        const newVotingThreshold = parseEther("0.00055"); // approximately 2 usd
83        await contract.connect(owner).setVotingThreshold(newVotingThreshold);
84        const contractVotingThreshold = await contract.votingThreshold();
```

test > CrowdFunding.ts > ...

```
9   describe("CrowdFunding", () => {
79    describe("Owner", async () => {
80      it("Should be able to set votingThreshold", async () => {
84        const contractVotingThreshold = await contract.votingThreshold();
85
86        expect(contractVotingThreshold).to.eq(newVotingThreshold);
87      });
88
89      it("Should be able to create voting", async () => {
90        const { contract, owner } = await loadFixture(deployCrowdfundingFixture);
91        const votingConstructor = await getVotingConstructor();
92        await contract
93          .connect(owner)
94          .createVoting(
95            votingConstructor.description,
96            votingConstructor.startTimestamp,
97            votingConstructor.endTimestamp,
98            votingConstructor.candidatesDescriptions,
99            votingConstructor.candidateImages,
100         );
101       const voting = await contract.votings(0);
102
103       expect(voting).to.not.eq(ZeroAddress);
104     });
105
106     describe("Withdraw", async () => {
107       it("Should not be able to withdraw funds if crowdfunding is not finished yet", async () => {
108         const { contract, owner } = await loadFixture(
109           deployCrowdfundingFixture
110         );
111
112         await expect(contract.connect(owner).withdraw()).to.be.revertedWith(
113           "CrowdFunding is not finished yet"
114         );
115       });
116
117       it("Should not be able to withdraw funds if crowdfunding failed", async () => {
118         const { contractConstructor, contract, owner } = await loadFixture(
119           deployCrowdfundingFixture
120         );
121         await time.increaseTo(contractConstructor.endTimestamp);
122
123         await expect(contract.connect(owner).withdraw()).to.be.revertedWith(
124           "CrowdFunding failed"
125         );
126       });
```

test > CrowdFunding.ts > ...

```
  9  describe("CrowdFunding", () => {
148    describe("Non owner", async () => {
160      it("Should not be able to create voting", async () => {
165        await expect(
166          contract
167            .connect(account2)
168            .createVoting(
169              votingConstructor.description,
170              votingConstructor.startTimestamp,
171              votingConstructor.endTimestamp,
172              votingConstructor.candidatesDescriptions,
173              votingConstructor.candidateImages,
174            )
175        ).to.be.revertedWithCustomError(contract, "OwnableUnauthorizedAccount");
176      });
177
178      it("Should not be able to withdraw funds", async () => {
179        const { contract, account2 } = await loadFixture(
180          deployCrowdfundingFixture
181        );
182
183        await expect(
184          contract.connect(account2).withdraw()
185        ).to.be.revertedWithCustomError(contract, "OwnableUnauthorizedAccount");
186      });
187    });
188
189    describe("Crowdfunding", async () => {
190      it("Should receive contributions", async () => {
191        const { contractAddress, contract, account2 } = await loadFixture(
192          deployCrowdfundingFixture
193        );
194        const FUNDS_AMOUNT = parseEther("0.5");
195        await account2.sendTransaction({
196          value: FUNDS_AMOUNT,
197          to: contractAddress,
198        });
199        const contribution = await contract.contributions(account2.address);
200
201        expect(contribution).to.eq(FUNDS_AMOUNT);
202      });
203
204      it("Should not receive contributions if company failed", async () => {
205        const { contractAddress, contract, account2 } = await loadFixture(
206          deployCrowdfundingFixture
```

test > CrowdFunding.ts > describe("CrowdFunding") callback > describe("Crowdfunding") callback

```
  9  describe("CrowdFunding", () => {
189    describe("Crowdfunding", async () => {
204      it("Should not receive contributions if company failed", async () => {
206          deployCrowdfundingFixture
207        );
208        const end = await contract.endTimestamp()
209        await time.increaseTo(end + BigInt(1_000))
210        const FUNDS_AMOUNT = parseEther("0.5");
211        await expect(
212          account2.sendTransaction({
213            value: FUNDS_AMOUNT,
214            to: contractAddress,
215          })
216        ).to.be.revertedWith('CrowdFunding failed')
217      });
218
219      describe("Vote", async () => {
220        it("Should not allow to vote when contribution threshold not met", async () => {
221          const { contract, account2 } = await loadFixture(
222            deployCrowdfundingFixture
223          );
224
225          await expect(contract.connect(account2).vote(0, 0)).to.be.revertedWith(
226            "Voting threshold not met"
227          );
228        });
229      });
230
231      describe("Refund", async () => {
232        it("Should not allow to refund without contributions", async () => {
233          const { contract, account2 } = await loadFixture(
234            deployCrowdfundingFixture
235          );
236
237          await expect(contract.connect(account2).refund()).to.be.revertedWith(
238            "No contributions"
239          );
240        });
241
242        it("Should not allow to refund when crowdfunding not failed", async () => {
243          const { contractAddress, contract, account2 } = await loadFixture(
244            deployCrowdfundingFixture
245          );
246          const FUNDS_AMOUNT = ethers.parseEther("1");
247          await account2.sendTransaction({
248            value: FUNDS_AMOUNT,
```

test > CrowdFunding.ts > ...

```
  9  describe("CrowdFunding", () => {
189    describe("Crowdfunding", async () => {
231      describe("Refund", async () => {
242        it("Should not allow to refund when crowdfunding not failed", async () => {
243          const { contractAddress, contract, account2 } = await loadFixture(
244            deployCrowdfundingFixture
245          );
246          const FUNDS_AMOUNT = ethers.parseEther("1");
247          await account2.sendTransaction({
248            value: FUNDS_AMOUNT,
249            to: contractAddress,
250          });
251
252          await expect(contract.connect(account2).refund()).to.be.revertedWith(
253            "CrowdFunding not failed"
254          );
255        });
256
257        it("Should allow to refund when crowdfunding failed", async () => {
258          const { contractConstructor, contractAddress, contract, account2 } =
259            await loadFixture(deployCrowdfundingFixture);
260          const FUNDS_AMOUNT = ethers.parseEther("0.9");
261          await account2.sendTransaction({
262            value: FUNDS_AMOUNT,
263            to: contractAddress,
264          });
265          await time.increaseTo(contractConstructor.endTimestamp);
266
267          await expect(
268            contract.connect(account2).refund()
269          ).to.changeEtherBalances(
270            [account2.address, contractAddress],
271            [FUNDS_AMOUNT, -FUNDS_AMOUNT]
272          );
273        });
274      });
275    });
276  });
277
```