МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук, фізики та математики

Кафедра комп’ютерних наук та програмної інженерії

Розумний контракт для студентських голосувань написаний мовою Solidity

Кваліфікаційна робота (проєкт)

на здобуття ступеня вищої освіти «магістр»

Виконав: здобувач 2 курсу групи 12-241М
Спеціальності 121 "Інженерія програмного забезпечення"
Освітньо-професійної програми "Інженерія програмного забезпечення"
Рудь Віктор Анатолійович

Керівник:доктор технічних наук, професор Песчаненко Володимир Сергійович
Рецензент: Тарасіч Ю.Г., докторант інституту кібернетики імені В.М. Глушкова НАН України, керівник компанії «Garuda.AI»

Івано-Франківськ 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. Блокчейн - цифрова децентралізована база даних, яка складається з послідовності блоків.
2. Смартконтракт - автономний програмний код, який автоматично виконує певні дії при настанні заздалегідь визначених умов.
3. L2 / Layer 2 - рішення другого рівня для масштабування мереж блокчейн.
4. FPTP / "First Past The Post" - система голосування, де переможцем вважається кандидат, який набрав найбільше голосів.
5. Solidity - мова програмування для написання смартконтрактів на Ethereum.
6. Gas - одиниця вимірювання вартості виконання операцій і зберігання в Ethereum.
7. EVM / Ethereum Virtual Machine - віртуальна машина Ethereum, яка дозволяє виконувати смартконтракти.
8. Хеш — це унікальний рядок символів фіксованої довжини, отриманий в результаті обробки вхідних даних хеш-функцією, який використовується для швидкого доступу до даних та перевірки їх цілісності.
9. Сайдчейн — це допоміжний блокчейн, який працює паралельно основному блокчейну, дозволяючи передачу активів між ними та

експериментування з новими функціями без впливу на основний блокчейн.

10. Mapping у контексті. Solidity це тип даних, який дозволяє створювати відносини між ключем та значенням, де кожен унікальний ключ відображається на відповідне значення.
11. Юніт-тести (unit tests) — це метод тестування, при якому окремі частини програмного коду (юніти, наприклад, функції, методи, класи) перевіряються окремо від інших частин для забезпечення їх правильної роботи.

ВСТУП

Голосування завжди відігравало ключову роль у формуванні соціальних, політичних і навіть наукових процесів суспільства. Голосування є не лише фундаментальним інструментом демократії, але і важливим механізмом для формування і регулювання соціальних, політичних та наукових аспектів життя суспільства. Воно забезпечує громадянам не просто право висловлювати свої погляди, але і взяти на себе відповідальність за колективне ухвалення рішень на різних рівнях — від місцевих ініціатив до національних референдумів. В контексті освітнього процесу, голосування стає не менш значущим. В умовах сучасних освітніх інституцій, де нерідко існує незбалансована взаємодія між студентами та управлінськими структурами, якісна і прозора система голосування може служити ефективним інструментом для забезпечення діалогу, конструктивного обговорення та прийняття обґрунтованих рішень щодо різноманітних аспектів навчального процесу та життя студентської спільноти.

Історія голосування показує еволюцію від простих усних методів до формалізованих письмових бюлетенів. Цей процес не був без викликів: загублені бюлетені, підроблені голоси та шахрайство завжди були

актуальними проблемами. Сучасні технології, такі як блокчейн, надають шанс робити голосування прозорішим, швидшим та надійнішим.

З розвитком технологій та інформатизацією суспільства, процес голосування переживає значущі зміни, стаючи дедалі більш динамічним, гнучким і доступним. Традиційні паперові системи голосування поступаються місцем новим електронним методам, які забезпечують не тільки більшу прозорість та швидкість, але і підвищують рівень довіри до результатів. Однією з найбільш перспективних технологій у цьому контексті є блокчейн, який надає можливість створення децентралізованих, безпечних і незмінних систем голосування. Ця інноваційна технологія стає особливо актуальною при використанні розумних контрактів, написаних мовою програмування Solidity. Ці контракти не лише автоматизують процес голосування, але і забезпечують його автономність та високий рівень прозорості, доступний для перевірки всіма зацікавленими сторонами.

У цій роботі ми досліджуємо, як сучасні технології, такі як блокчейн і розумні контракти, можуть радикально переосмислити організацію та проведення голосувань, надаючи нові інструменти для зміцнення демократичних принципів та підвищення довіри громадян. Зокрема, ми розглядаємо концепцію розумного контракту для студентських голосувань, розробленого мовою програмування Solidity, і аналізуємо його потенційні переваги та виклики в контексті сучасного освітнього середовища.

Актуальність теми можна оцінити з кількох кутів. В першу чергу, використання блокчейн-технологій в освітньому секторі є перспективним напрямком, який може забезпечити прозорість та надійність студентських голосувань. Мова програмування Solidity є ключовою для розробки розумних контрактів на платформі Ethereum, що є однією з найбільш популярних блокчейн-платформ сьогодні. Крім того, тема актуальна у контексті глобального переходу до цифровізації

освітніх процесів. Враховуючи дедалі більшу потребу в автоматизації та цифровій безпеці, розумні контракти можуть служити надійним інструментом для впровадження демократичних механізмів в освітній простір. Таким чином, розробка та впровадження розумних контрактів для студентських голосувань може стати важливим кроком на шляху до сучасної, ефективної та безпечної освітньої системи.

Розглянемо *зв'язок роботи з науковими програмами, планами, темами*. Ця кваліфікаційна робота є частиною ширшого наукового дослідження в області цифровізації освітнього процесу та може бути використана в рамках наукових програм, пов'язаних із блокчейн-технологіями.

Моя наукова робота уявляє собою міждисциплінарний проєкт, що об'єднує елементи інформаційних технологій, права та соціальних наук. Центральна частина дослідження — це розробка смартконтрактів, що можуть бути застосовані для різних видів голосувань та підрахунку голосів. Ця технологічна основа забезпечує високий рівень прозорості, надійності та автоматизації процесу, що є критично важливим у сучасних умовах.

Спектр можливих застосувань цих смартконтрактів є величезним: від організації виборів на різних рівнях управління до проведення референдумів та корпоративного голосування. Це робить дослідження актуальним не тільки в контексті комп'ютерних наук, але і в рамках правових та соціальних дисциплін, де питання демократії, участі громадян і корпоративного управління залишаються предметом жвавого наукового дискусу.

Таким чином, ця робота не просто робить внесок в технічну сферу, але і розширює розуміння можливостей та обмежень технологічних інструментів у соціальних та правових системах. Це робить її частиною ширшого наукового діалогу та відповідає актуальним міждисциплінарним викликам.

Практична значущість цієї дослідницької роботи особливо проявляється в контексті освітнього середовища. Розроблені смартконтракти можуть служити потужним інструментом для організації студентських голосувань на різних рівнях: від виборів представників студентського самоврядування до голосувань за зміни в навчальному плані чи інших студентських ініціатив.

Це не лише забезпечує високий рівень прозорості і надійності, але і значно спрощує сам процес голосування, роблячи його більш доступним і зручним для студентів. Враховуючи постійно ростучу роль студентської активності в академічному житті, така технологія може стати неоціненною для підвищення ефективності студентського самоврядування та залучення студентів до активної участі в житті освітнього закладу.

Таким чином, практична значущість цієї роботи робить її актуальною не тільки в науковому, але і в практичному відношенні. Це відкриває нові можливості для застосування сучасних технологій в освітньому процесі, відповідаючи актуальним потребам студентської спільноти.

Метою дослідження є розробка блочної системи електронного голосування для різних видів підрахунку голосів на базі блокчейн-технологій і розумних контрактів, написаних мовою Solidity. Дослідити зручність методів архітектури коду.

Завданнями дослідження є:

- Аналіз наявних методів голосування. Вивчення традиційних та сучасних методів голосування, їх переваг та недоліків.
- Вивчення технології блокчейн. Ознайомлення з основними принципами роботи блокчейну та розумних контрактів.
- Розробка концепції системи. Створення теоретичної моделі системи електронного голосування на базі блокчейну.

- Програмування розумних контрактів. Реалізація розумних контрактів на мові Solidity для забезпечення функціональності системи.
- Оцінка прозорості. Аналіз механізмів забезпечення прозорості в розробленій системі.

Об'єкт дослідження:

- Процес голосування в університетському середовищі.
- Системи електронного голосування на базі блокчейн-технологій.

Предмет дослідження:

- Розумні контракти для студентських голосувань, написані мовою Solidity.
- Системи підрахунку голосів.

Технології:

- Remix IDE
- Visual Studio Code
- Плагіни VSC:
    - Better Comments
    - CodeSnap
    - Solidity Visual Developer
    - HardHat
    - PlantUML
- HardHat

Для досягнення поставленої мети в цій науковій роботі було використано комплекс *методів дослідження*, кожен з яких спрямований на вирішення конкретних задач.

- Експериментальне моделювання та тестування смартконтрактів: За допомогою технологій Remix IDE і плагінів для Visual Studio Code було розроблено та протестовано прототипи смартконтрактів для

студентських голосувань. Цей метод дозволив нам валідувати функціональність та надійність розроблених систем.

- Комп'ютерний аналіз та симуляція: Для оцінки ефективності та масштабованості системи було проведено числове моделювання.
- Кейс-студії: За допомогою аналізу наявних систем голосування було вивчено потенційні плюси та мінуси впровадження блокчейн-технологій в освітньому секторі.
- Літературний огляд: Вивчення академічних та технічних ресурсів дало можливість ознайомитися з наявними методами та підходами до організації голосувань і блокчейн-технологій.

Комбінація цих методів дала можливість провести всеосяжний аналіз та оцінку потенціалу впровадження блокчейн-технологій в системи студентських голосувань, що забезпечило *наукову та практичну цінність* цієї роботи.

У дипломній роботі впроваджено комплексний підхід до організації системи голосування на базі блокчейну. Цей підхід включає динамічне керування списком учасників, їх ваговим коефіцієнтом у голосуванні, а також варіантами пропозицій для голосування. Ось ключові наукові положення, які автор вводить в цій роботі:

- Динамічна конфігурація власників: розроблено механізм, що дозволяє динамічно конфігурувати список власників. Це може бути корисно для систем, де склад комітету може змінюватися.
- Модульність і розширюваність: Архітектура смартконтракту розбита на декілька логічних блоків. Це дозволяє легко розширювати та модифікувати функціональність.
- Делегування ваги голосу: Введено механізм делегування, який дозволяє учасникам передавати свій ваговий коефіцієнт іншим учасникам, що забезпечує гнучкість в організації голосування.

- Вбудований механізм таймінгу: Смартконтракт має вбудований механізм для контролю часу початку і завершення голосування, що забезпечує додатковий рівень контролю та безпеки.
- Вагові коефіцієнти для учасників: Введено систему вагових коефіцієнтів, яка дозволяє зробити процес голосування більш представницьким, надаючи можливість учасникам з більшим вкладом мати більший вплив на результат.

Завдяки цим положенням, робота уявляє собою новаторський внесок у сферу електронних систем голосування на базі блокчейну.

Розглянемо *практичне значення одержаних результатів.* Смартконтракт, розроблений в рамках цієї кваліфікаційної роботи, має великий потенціал для застосування в реальному світі, зокрема у сфері електронних голосувань. Цей інструмент може стати корисним у ряді сфер, від студентських голосувань до корпоративних рішень.

- Область застосування. Основний фокус роботи — студентські голосування. Смартконтракт може забезпечити прозорість, надійність та швидкість голосувань, усуваючи потребу в фізичній присутності та паперових бюлетенях.
- Економічна ефективність. Застосування смартконтракту може знизити вартість організації та проведення голосувань, витрат на друк бюлетенів та зарплату для людей, які контролюють процес.
- Технологічна готовність. На поточному етапі смартконтракт є повністю функціональним, але потребує подальшого тестування та аудиту безпеки перед впровадженням на широкий ринок.
- Безпека і надійність. Завдяки використанню технології блокчейн, смартконтракт може забезпечити високий рівень безпеки і захисту від фальсифікацій.
- Скальованість. Смартконтракт розроблений таким чином, що його легко можна адаптувати для різних типів голосувань та масштабів.

- Юридична складова. Хоча юридичні аспекти ще потребують додаткового дослідження, смартконтракт має потенціал для впровадження в чинне законодавство щодо електронних голосувань.
- Рекомендації для подальшого використання. Для повного впровадження системи рекомендується провести аудит безпеки, а також адаптувати її до конкретних юридичних вимог та стандартів.

Апробація результатів дослідження. У рамках дипломної роботи автор детально дослідив використання масивів у мові програмування Solidity. Всі розроблені модулі та алгоритми базуються на структурі даних "масив", що дозволило глибше вивчити її особливості, переваги та обмеження.

На основі проведеного дослідження було зроблено висновок, що масиви в Solidity є ефективними для певного спектра завдань, але для оптимізації ефективності в майбутніх проєктах рекомендується використовувати комбінований підхід. Це передбачає використання як масивів, так і "mappings", що дозволить об'єднати переваги обох структур даних.

Апробація результатів дослідження. Цей досвід було викладено у науковій статті "Структури даних у Solidity: порівняльний аналіз масивів та mappings."[47], яка розглядає основні структури даних у мові програмування Solidity, зокрема масиви та "mappings". У статті проводиться порівняльний аналіз цих структур з точки зору їхньої ефективності, особливостей використання та можливостей оптимізації. Основна увага приділена практичним рекомендаціям щодо вибору оптимальної структури даних для конкретних завдань в контрактах на Solidity. На цей час стаття очікує на публікацію в науковому журналі. Також планується представлення результатів на наукових конференціях та симпозіумах.

РОЗДІЛ 1
Огляд літератури за темою і вибір напрямів досліджень

1.1 Вступ до розділу

В сучасному світі технології все більше проникають в різні сфери життя, включаючи освіту та академічну діяльність. Однією з ключових проблем, яка виникає в академічних установах, є організація прозорих, ефективних та безпечних систем голосувань. Це може бути голосування за кандидатів на посади студентської ради, вибір тем для наукових досліджень або ж голосування по інших академічних питаннях.

Традиційні методи голосування часто не можуть забезпечити необхідний рівень прозорості та безпеки. В цьому контексті блокчейн-технології та смартконтракти відкривають нові можливості для створення децентралізованих та надійних систем.

Цей розділ має на меті дослідити наявну наукову літературу, проєкти та технології, що можуть бути використані для реалізації смартконтракту для студентських голосувань. Ми розглянемо історичний контекст розвитку блокчейн-технологій, проаналізуємо ключові роботи в цій області, а також визначимо проблеми та відкриті питання, які ще залишаються невирішеними.

1.2 Історичний огляд розвитку блокчейн-технологій

1.1.1 Перші покоління блокчейну та їх обмеження

Блокчейн з'явився на світовій арені з появою криптовалюти Bitcoin у 2008 році, яка була створена анонімним розробником або групою розробників під псевдонімом Сатоші Накамото. Перші покоління блокчейну були в основному орієнтовані на фінансові транзакції та анонімність. Однак, ці системи мали свої обмеження, такі як масштабованість, швидкість транзакцій та відсутність гнучкості в реалізації бізнес-логіки.

1.1.2 Поява смартконтрактів

Ситуація змінилася з появою Ethereum у 2015 році, який ввів концепцію смартконтрактів. Смартконтракти дозволили розробникам створювати складні децентралізовані додатки (dApps) з власною бізнес-логікою. Це відкрило двері для використання блокчейну в різних сферах, включаючи голосування, управління постачанням, охорону здоров'я та освіту.

В контексті студентських голосувань, смартконтракти можуть забезпечити прозорість, незмінність та автоматизацію процесу, що є ключовими факторами для голосування.

Цей історичний огляд показує, що хоча блокчейн та смартконтракти пройшли довгий шлях від своїх початкових етапів, є ще невирішені питання, особливо в контексті студентських голосувань. Це охоплює питання масштабованості, доступності та правової регуляції.

1.3 Аналіз наявних робіт. Огляд ключових наукових робіт, статей, та проєктів, що стосуються теми. Критичний аналіз з ідентифікацією переваг та недоліків

На тему "Розумний контракт для голосувань на Solidity" існує декілька наукових робіт, статей та проєктів. Нижче наведено короткий огляд деяких з них.

1.3.1 Blockchain-based Preferential E-Voting System DApp using Smart Contract.

- Автори: Gupta, Saurav та C R, Manjunath.
- Дата публікації: 27 квітня 2021 року.
- Опубліковано у збірнику матеріалів Міжнародної конференції з інноваційних обчислень та комунікацій (ICICC) 2021 року.

На основі статті Saurav Gupta та Manjunath CR "Blockchain-based Preferential E-Voting System DApp using Smart Contract", можна визначити декілька переваг та недоліків представленої системи:

Переваги:

12. Безпека: Використання технології blockchain для голосування підвищує безпеку, оскільки кожна транзакція (голос) зберігається в блоках, які з'єднані хешами попередніх блоків. Це ускладнює зміну даних після їх запису.
13. Прозорість: Завдяки блокчейну всі учасники можуть перевірити результати голосування без можливості подвійного голосування або зміни результатів після закінчення голосування.
14. Ефективність: Система вводить концепцію торгівлі голосами, коли голоси можуть бути перерозподілені між іншими кандидатами, якщо чітка більшість не досягнута, що може зменшити необхідність повторних голосувань та зекономити час та ресурси.
15. Пріоритетність: Система дозволяє виборцям вказувати свої переваги щодо кандидатів, а не просто голосувати за одного кандидата, що може допомогти в отриманні більш точного відображення волевиявлення виборців.

Недоліки:

1. Технічна складність: Впровадження та управління системою на базі blockchain може вимагати високого рівня технічних знань та розуміння, які можуть бути бар'єром для деяких організацій.
2. Вартість транзакцій: В системі Ethereum за кожну транзакцію потрібно платити певну суму газу (Ether). У вказаній системі адміністратор надає Ether кожному виборцеві для голосування, але це може бути коштовним, особливо для великих організацій.
3. Приватність: Стаття не надає чіткого розуміння того, як гарантується приватність виборців, що є критично важливим аспектом будь-якої системи голосування.
4. Централізація верифікації: Існує централізований сервер для верифікації ідентичності користувачів під час реєстрації, що може стати слабким місцем у системі з точки зору безпеки та децентралізації. [1]

1.3.2 E-Voting on Blockchain using Solidity Language.

Робота уявляє приклад реалізації електронної виборчої системи як розумного контракту для мережі Ethereum за допомогою Solidity.

У публікації обговорюється розробка системи електронного голосування з використанням технології блокчейн для розв'язання проблем безпеки, пов'язаних з електронним голосуванням. Система була спрямована на підвищення прозорості, безпеки та аудитоспроможності виборів до Студентської представницької ради (СПР) в Університеті Тун Хуссейн Онн, Малайзія (UTHM), не порушуючи при цьому конфіденційності виборців. Система була розроблена з використанням мови Solidity і працювала на платформі блокчейн для виявлення шахрайства під час голосування.

Деякі ключові моменти з публікації такі:

Система електронного голосування з використанням технології блокчейн була запропонована для покращення безпеки та прозорості електронного голосування, особливо в контексті виборів до СРК УТГМ.

Технологія блокчейн забезпечує прозорість, безпеку та можливість аудиту, зберігаючи при цьому конфіденційність виборців.

Система була розроблена з використанням мови Solidity, яка є мовою програмування, що використовується для написання смартконтрактів на різних блокчейн-платформах. Система була розроблена для виявлення шахрайства, яке може статися під час голосування.

Для оцінки запропонованої системи було проведено тематичне дослідження та анкетування. Було проаналізовано три кейси: звичайне голосування з використанням блокчейну, порівняння електронного голосування з використанням блокчейну та без нього, а також виявлення шахрайства. Анкета мала на меті зібрати інформацію про сприйняття респондентами системи е-голосування на блокчейні. Аналіз відповідей респондентів показав позитивне сприйняття із середніми значеннями 4,5, 4,6 та 4,9 для 1-го, 2-го та 3-го випадків відповідно.

Система електронного голосування на блокчейні виявилася зручною для користувачів і здатною зменшити ризик шахрайства та суперечок, зберегти анонімність виборців, а також заощадити час і кошти порівняно з традиційними системами голосування на паперових носіях.

У висновках публікації визнається потенціал технології блокчейн у забезпеченні безпеки систем е-голосування та висловлюється припущення, що за допомогою деяких удосконалень систему можна зробити більш практичною та зручною для майбутніх виборів на УТГМ. [2]

1.3.3 How To Implement A Voting Smart Contract.

На основі аналізу статті "How To Implement A Voting Smart Contract" можна виділити наступні переваги та недоліки:

Переваги:

1. Надає конкретні кроки для створення контракту для голосування, що може бути корисним для новачків.

2. Детально пояснює, як працюють окремі функції та ключові слова Solidity, що сприяє кращому розумінню.

Недоліки:

1. Стаття не надає інформації про оптимізацію газу або обробку помилок, що є важливими аспектами розробки розумних контрактів.

2. Інструкції можуть бути складними для тих, хто не має попереднього досвіду з Solidity або блокчейном Ethereum.

Блог, що допомагає зрозуміти концепції блокчейну та розумних контрактів, представляючи розробку системи голосування за допомогою Solidity на платформі Remix IDE. Стаття розповідає, як створити акціонерний виборчий розумний контракт за допомогою Solidity, включаючи функції для додавання кандидатів, отримання списку кандидатів та голосування. [3]

1.3.4 Building a Voting System Application using Solidity.

Стаття від Aakash A на Medium розповідає про створення системи голосування з використанням Solidity на платформі Remix IDE. Автори описують основні концепції Blockchain і Smart Contracts, а також детально розглядають код та його роботу для створення системи голосування. Основні аспекти, які були розглянуті, включають створення та використання розумних контрактів для голосування, обробку голосів та забезпечення прозорості та безпеки процесу голосування.

Переваги:

1. Детальний опис процесу створення системи голосування.

2. Навчальний матеріал з корисними прикладами коду.

3. Пояснення основних концепцій blockchain та smart contracts.

Недоліки:

1. Можливе відсутність інформації про оптимізацію або масштабування системи.

2. Відсутність дискусії про потенційні проблеми безпеки або заходи щодо їх вирішення.

3. Обмежений контекст щодо інших можливих технологічних рішень для систем голосування. [4]

Learn Solidity: Complete Example: Voting Ballot Smart Contract:

Повна реалізація виборчого бюлетеня як розумного контракту для голосування, що є частиною серії постів про вивчення Solidity та створення децентралізованих застосунків в Ethereum. [5]

В цілому в статтях не вистачає комплексного підходу до створення розширюваного протоколу голосування та опису переваг та недолік підходів до створення системи. Як правило описуються контракти створені для вузької потрібної системи голосування, підрахунків голосів тощо. Не вистачає рішень які можуть бути перевикористані в схожих проєктах.

1.4 Технологічні платформи для реалізації смартконтрактів

Вибір платформи для реалізації смартконтрактів є критичним етапом в розробці будь-якої децентралізованої системи. Різні платформи мають свої переваги та недоліки, які можуть вплинути на ефективність, безпеку та доступність смартконтракту.

1.4.1 Bitcoin.

Мова програмування: Bitcoin Script

Особливості: Bitcoin це піонер криптовалют, який використовує механізм консенсусу Proof-of-Work. Bitcoin забезпечує високу безпеку та міцну децентралізацію, але це коштує його обмеженої пропускної спроможності та швидкості транзакцій.

Чому не підходить для проєкту: Bitcoin не підтримує смартконтракти тою мірою, в якій це потрібно для проєкту. Вбудована мова

програмування Bitcoin Script обмежена своєю функціональністю та не підходить для створення складних додатків, які ми плануємо розробити. Крім того, Bitcoin має обмежену пропускну спроможність та повільну швидкість обробки транзакцій, що робить його непридатним для додатку, який вимагає високої масштабованості.

1.4.2 Polkadot.

Мова програмування: Rust, JavaScript, Go, C++, Python

Особливості: Polkadot це багатоланцюжкова платформа, яка дозволяє різним блокчейнам ефективно спілкуватися між собою. Платформа підтримує різні паралельні ланцюжки (парачейни) та має головний ланцюг (реле-чейн) для забезпечення безпеки та консенсусу між різними парачейнами.

Чому не підходить для проєкту: Попри гнучкість та інтероперабельність Polkadot, виявлено, що ця платформа не є оптимальним вибором для проєкту. По-перше, хоча Polkadot підтримує різні мови програмування, дослідження зосереджене на Solidity, як основній мові для проєкту. По-друге, механізм інтероперабельності Polkadot може бути занадто складним для проєкту, який зосереджений на виконанні конкретних задач у контексті університету, тому складність, яка виникає від спроби зв'язати різні блокчейни, може бути непотрібною.

1.4.3 Solana.

Мова програмування: Rust, C

Особливості: Solana це високопродуктивний блокчейн, який використовує модель консенсусу з підтвердженням часу, що забезпечує швидкість та масштабованість. Він може обробляти великі обсяги транзакцій без значних затримок чи втрат процесорного часу.

Чому не підходить для проєкту: Попри те, що Solana пропонує високу продуктивність та масштабованість, ми виявили, що вона не є найкращим вибором для проєкту з декількох причин. По-перше, Solana

використовує Rust та C для розробки додатків, але автор має досвід роботи з Solidity, яка є основною мовою для Ethereum і її форків. Використання іншої мови програмування б вимагало більшої кількості часу для навчання та переходу. По-друге, Solana може бути занадто складною для проєкту, який потребує простоти та ефективності, а не високої продуктивності для великого обсягу транзакцій.

1.4.4 Cosmos.

Мова програмування: Go

Особливості: Cosmos є блокчейн-технологією, яка створена для забезпечення масштабованості та з'єднання різних незалежних блокчейнів в одну глобальну мережу. Цією концепцією він схожий на Polcadot. Cosmos використовує модель консенсусу Tendermint, яка забезпечує високу продуктивність та безпеку. Крім того, Cosmos дає змогу розробляти додатки мовою програмування Go.

Чому не підходить для проєкту: Попри цілу низку переваг Cosmos, є ряд причин, чому прийняте рішення не вибирати цей блокчейн для проєкту. По-перше, мова програмування для розробки додатків в Cosmos це Go, мова, з якою автор дослідження має менший досвід порівняно з Solidity. Це означає, що було б потрібно більше часу на навчання та адаптацію до нової мови програмування. По-друге, Cosmos був створений з метою масштабування та з'єднання різних блокчейнів, що перевищує потреби проєкту. У моєму випадку нам потрібен стабільний, безпечний та простий в застосуванні блокчейн, а не глобальна мережа з'єднаних блокчейнів.

1.4.5 Ethereum.

Мова програмування: Solidity

Особливості: Ethereum є платформою open-source, яка використовує технологію блокчейн для створення та виконання децентралізованих програм (dApps). Ethereum був першим проєктом, який впровадив

концепцію smart-контрактів, які дозволяють автоматично виконувати угоди без потреби в посередниках.

Чому підходить для проєкту: Ethereum є популярним вибором для багатьох проєктів на блокчейні, завдяки своєму впровадженню інтелектуальних контрактів і децентралізованих програм. Для проєкту це є цінним, оскільки ми можемо створити систему голосування, яка автоматично реєструє голоси без потреби в посередниках. Також Ethereum має сильну спільноту розробників, яка може надати значну підтримку і ресурси. Проте, на думку автора дослідження, проблеми з масштабуванням і високі комісії Ethereum можуть вплинути на ефективність та прийнятність проєкту, отже надалі розглянемо також вибір L2 рішень.

Основною з причин вибору Solidity та Etherium сумісних блокчейнів стала їх розповсюдженість, популярність. Дуже багато людей користуються ними. Багато є матеріалу. Тому вчитися цим новим технологіям стає легше. Багато людей вже мають досвід їх використання.

1.5 Детальний аналіз Layer 2-рішень для Etherium

1.5.1 State Channels.

State Channels дозволяють двом або більше сторонам взаємодіяти в приватному каналі, який тимчасово від'єднаний від основного блокчейну. Транзакції здійснюються всередині каналу і потім агрегуються в одну транзакцію для відправки на основний блокчейн. Приклад: Raiden Network.

Плюси:

Низька вартість транзакцій: Транзакції всередині каналу не потребують газу, що істотно знижує вартість.

Швидкість: Транзакції всередині каналу моментальні і не вимагають підтвердження майнерами.

Мінуси:

Обмежена функціональність: State Channels найкраще підходять для сценаріїв з безліччю транзакцій між фіксованим набором учасників.

Комплексність: Обидві сторони повинні бути онлайн для проведення транзакцій, що ускладнює процес.

1.5.2 Sidechains.

Sidechains це окремі блокчейни, які працюють паралельно до основного блокчейну, але мають власні механізми консенсусу та правила. Наприклад, xDAI Chain.

Плюси:

Швидкість: Sidechains зазвичай швидші за основний блокчейн через менше навантаження.

Гнучкість: Можна створити сайдчейн з унікальними властивостями, які підходять під конкретні потреби.

Мінуси:

Безпека: Sidechains зазвичай менш безпечні, ніж основний блокчейн, тому що вони мають менше учасників для досягнення консенсусу.

Ліквідність: Переміщення активів між основним блокчейном і сайдчейном може бути складним і вимагати додаткових кроків.

1.5.3 Plasma.

Plasma це фреймворк для створення "дочірніх" блокчейнів, які співпрацюють з основним блокчейном для безпеки. Транзакції відбуваються в цих дочірніх блокчейнах і періодично синхронізуються з основним блокчейном. Наприклад, Polygon.

Плюси:

Масштабованість: Plasma може обробляти велику кількість транзакцій паралельно, що підвищує загальну пропускну здатність мережі.

Безпека: Plasma використовує безпеку основного блокчейна для підтвердження транзакцій, що робить його відносно безпечним.

Мінуси:

Комплексність: Plasma складно реалізувати і підтримувати, і це може стати перешкодою для його широкого застосування.

Обмежена функціональність: Не всі типи транзакцій і смартконтрактів можуть бути реалізовані на Plasma.

1.5.4 Rollups (Optimistic & ZK).

Rollups агрегують безліч транзакцій в одну, яка потім публікується на основному блокчейні. Це дозволяє знизити навантаження на основний блокчейн і збільшити його пропускну здатність.

1.5.4.1 Optimistic Rollups.

Optimistic Rollups це Layer 2-рішення для масштабування Ethereum та інших блокчейнів. Ця технологія агрегує безліч транзакцій в одну, яка потім відправляється на основний блокчейн. Optimistic Rollups припускають, що всі транзакції є дійсними, і тільки в разі спору проводять повну перевірку. Приклади мереж: Optimism, Arbitrum.

Плюси:

Масштабованість: Дозволяють обробляти значно більше транзакцій за секунду, ніж основний блокчейн.

Сумісність: Легко інтегруються з наявними смартконтрактами та DApps на Ethereum.

Мінуси:

Латентність: Час підтвердження транзакцій може бути довгим, якщо виникне суперечка.

Складність: Реалізація та використання вимагають розуміння нових методів і підходів.

1.5.4.2 ZK Rollups.

ZK Rollups також агрегують безліч транзакцій в одну, але використовують zk-SNARKs для доказу дійсності кожної транзакції

перед відправкою на основний блокчейн. Приклади мереж: zkSync, Hermes Network.

Плюси:

Масштабованість: Забезпечують високу пропускну здатність, знижуючи навантаження на основний блокчейн.

Безпека: Використання zk-SNARKs забезпечує високий рівень безпеки.

Мінуси:

Складність: Використання zk-SNARKs вимагає складних криптографічних обчислень.

Стартові витрати: Налаштування та розуміння ZK Rollups можуть бути складними та вимагати спеціалізованих знань.

1.5.5 Polygon POS

Polygon це багатофункціональна платформа для створення блокчейнів, яка пропонує різні рішення для масштабування Ethereum, включно з PoS-сайдчейнами та Rollups.

Плюси:

Гнучкість: Polygon пропонує різні методи масштабування, включаючи PoS, PoA, sidechains, і навіть Rollups.

Сумісність: Повна сумісність з Ethereum робить його ідеальним для розробників, уже знайомих з Ethereum.

Мінуси:

Комплексність: Через широкий спектр рішень, вибір правильної конфігурації може бути складним.

Безпека: Деякі рішення, такі як PoS-сайдчейни, можуть бути менш безпечнішими, ніж основний блокчейн.

Polygon являє собою дуже гнучку і багатофункціональну платформу, що ідеально підходить для розробників, уже знайомих з екосистемою Ethereum. Він пропонує широкий спектр рішень для масштабування і

може бути хорошим вибором для смартконтрактів, особливо якщо вже є досвід роботи з Ethereum. [6]

Головним чинником за яким можна рекомендувати Polygon POS для проєктного рішення втілення голосування це значно нижчі комісії за використання обчислювальних потужностей. Це дуже важливо для росту рішень, т.я. вони потребують меншої уваги на оптимізації. Це дає можливість використовувати переваги циклів там де, будь це вищі комісії була б необхідність у mappings. Водночас це є популярним рішенням серед L2.

1.6 Проблеми та невирішені питання. Внесок дослідження і коротке резюме.

1.6.1 Які питання в галузі смартконтрактів для голосувань залишаються відкритими?

Однією з ключових проблем в сфері смартконтрактів для голосувань є ефективність використання газу в блокчейн-платформах, таких як Ethereum. Інші проблеми включають питання конфіденційності, верифікації ідентифікаторів, масштабованість, стійкість до атак та юридичну сумісність. Ці питання вимагають подальших досліджень для розробки ефективних і безпечних систем голосувань.

1.6.2 Чим розумний контракт для студентських голосувань може допомогти у розв'язанні цих проблем і питань?

Смартконтракт для студентських голосувань розроблений з урахуванням цих відкритих питань. Наприклад:

- Ефективність газу. Одним з основних обмежень смартконтрактів на платформах на кшталт Ethereum є ефективність використання газу. Неоптимізовані контракти можуть коштувати користувачам великих сум, що робить їх непрактичними для масового використання. Контракт оптимізований для зменшення витрат газу тільки в межах

технології масивів. Для зменшення плат за газ пропонується мережа Polygon PoS або розвернення студентського блокчейну.

- Конфіденційність. Смартконтракти зазвичай публічні та прозорі, що може бути проблемою для голосувань, які потребують анонімності. На даному етапі система голосувань є "proof-of-concept" (доказом концепції) або прототипом, який демонструє основні можливості технології. Реалізації анонімності можна додати пізніше. Це потребує подальших досліджень і розробок. Проєкт може слугувати відмінною базою для подальших наукових досліджень і практичних розробок, які можуть містити вирішення питань конфіденційності. Тим не меньш, якщо за адресою схований допущений інкогніто смартконтракт може реалізувати анонімність.

- Атаки. Смартконтракт реалізує суворі перевірки доступу та валідації, що робить його стійким до різних видів атак. Реалізовано два рівні доступу. Система смартконтрактів обладнана низкою виключень, що роблять атаку невигідною та занадто складною.

- Як упевнитися, що голосувальники мають право голосу? Це питання залишається відкритим і потребує додаткових досліджень і реалізацій. У системі це право реалізовано шляхом додавання адреси до списку виборців. Звісно це не єдиний шлях. Верифікація ідентифікаторів: Вбудована система перевірки, щоб забезпечити, що лише авторизовані особи можуть голосувати.

- Масштабованість: Контракт може легко масштабуватися для великих голосувань.

- Стійкість до атак: Реалізовані додаткові заходи безпеки для захисту від потенційних атак.

- Масштабованість. З урахуванням поточних обмежень блокчейн-платформ, виникає питання про те, як ефективно масштабувати системи голосування на основі смартконтрактів. Контракт розроблено з урахуванням можливого масштабування. Він ефективно працює з

відносно невеликою кількістю осіб, що підійде для університету. Якщо кількість буде величезною, на рівні країни, більшість функцій треба буде застосовувати у вигляді mappings.

Основний упор був зроблений на багаторазове застосування компонентів системи та легке розширення без трат потужностей на зайві функції. Тобто вони легко від’єднуються. Або можна реалізувати бібліотеку звідки смартконтракт алгоритму буде брати необхідні модулі за потребою.

1.6.3 Коротке резюме

З урахуванням вищезазначених проблем і відкритих питань, необхідно подальше наукове та практичне дослідження в галузі смартконтрактів для голосувань. Розумний контракт для студентських голосувань являє собою крок у цьому напрямку, проте дослідження в цій галузі повинні продовжуватися для подальшого вдосконалення та адаптації систем голосувань.

РОЗДІЛ 2
Аналіз сучасного стану проблеми. Розгляд переваг і недоліків різних методів голосування. Еволюція системи голосування

2.1 Традиційні методи голосування. Усна та письмова форми голосування. Проблеми та недоліки традиційних методів голосування

Голосування в навчальних закладах, таких як університети та коледжі, завжди слугувало яскравим прикладом традиційних методів, які базуються на простих та зрозумілих принципах. Ці методи, хоча й зрозумілі, мають свої специфічні особливості в студентському середовищі.

2.1.1 Усна та письмова форми голосування

В сучасних суспільствах голосування можна розділити на дві основні форми: усну та письмову, кожна з яких має свої переваги та недоліки. У студентських колах, зокрема на початкових курсах або в невеликих академічних групах, усне голосування є поширеним. В таких невеликих колективах, де кожен знайомий із всіма, усний метод виглядає простим та прямолінійним. Однак, цей метод може створити певні проблеми з анонімністю та свободою вибору, особливо коли є вплив ззовні.

З іншого боку, письмові форми голосування, такі як використання стандартизованих бюлетенів та урн, принесли новий рівень організованості і анонімності. Ця система дозволяє кожному голосу бути зафіксованим і врахованим, мінімізуючи ризик тиску чи маніпуляцій. Це особливо актуально в більших навчальних закладах, де письмове голосування часто стає найбільш практичним варіантом, що дозволяє зберегти конфіденційність та об'єктивність виборів.

Таким чином, обидва методи мають своє місце в різних контекстах: усний метод краще підходить для малих, згуртованих спільнот, тоді як письмове голосування з бюлетенями та урнами ефективніше в масштабних, більш формальних установах.

2.1.2 Проблеми та недоліки традиційних методів

Однак традиційні методи голосування не були позбавлені недоліків. Існувала можливість підробки бюлетенів, шахрайства під час підрахунку голосів і навіть фізичного втручання в процес голосування. Крім того, було потрібно багато часу та ресурсів для організації та проведення виборів. Також існував ризик, що бюлетені можуть бути загублені або знищені, що могло вплинути на підсумкові результати голосування. І, нарешті, відсутність можливості швидко і точно підрахувати голоси вручну залишала місце для помилок і непорозумінь. З розвитком технологій з'явилася потреба у створенні нових, ефективніших і безпечніших методів голосування.

Однією з проблем традиційного голосування в студентському середовищі є тиск з боку авторитетних осіб, таких як старости, викладачі чи декани. Цей тиск може змусити студентів голосувати не за того, кого вони вважають найкращим кандидатом, а за того, кого пропонує авторитетна особа. Навіть якщо голосування є формально анонімним, можливість ідентифікації вибору групи створює атмосферу страху та тиску. У такій ситуації вибір робиться не на основі особистої думки, а

через страх перед можливими репресіями. Це порушує основний принцип демократичного голосування — свободу вибору.

Таким чином, проблема не лише у фактичній анонімності, але і в психологічному тиску, який відчувають студенти, змушуючи їх голосувати "правильно" для уникнення можливих наслідків.

З розвитком технологій з'явилася потреба у створенні нових, ефективніших і безпечніших методів голосування, здатних захистити анонімність кожного студента та надати йому можливість вільно висловити свою думку.

2.2 Технологічний прогрес у голосуванні. Перехід до електронного голосування. Переваги та недоліки електронних систем голосування

2.2.1 Технологічний прогрес у голосуванні

З розвитком інформаційних технологій процес голосування також зазнав значних змін. Прогрес у цій царині почався з переходу від традиційного ручного голосування до більш сучасних і технологічно просунутих методів. Електронне голосування стало важливим кроком у поліпшенні та спрощенні процедури голосування. Замість того щоб заповнювати паперові бюлетені та здавати їх вручну, студенти можуть голосувати з використанням комп'ютерів або спеціалізованих пристроїв, які автоматично опрацьовують і підраховують голоси.

2.2.2 Переваги електронних систем голосування

- Швидкість і точність: Електронні системи можуть моментально обробляти і підраховувати голоси, виключаючи людські помилки.
- Доступність: Дозволяє голосувати дистанційно, що зручно для людей, які перебувають далеко від місць голосування.
- Економія ресурсів: Зменшує потребу в паперових матеріалах і людських ресурсах для проведення виборів.

2.2.3 Недоліки електронних систем голосування

- Технічні неполадки: Електронні системи можуть зіткнутися з проблемами в роботі, які можуть спотворити результати або ускладнити процес голосування.
- Питання безпеки: Електронні системи схильні до хакерських атак, що може ставити під загрозу правдивість результатів голосування.
- Витрати на впровадження: Перехід на електронне голосування вимагає значних інвестицій у купівлю та обслуговування обладнання.
- Підробка результатів: Можливість маніпулювання результатами голосування або навмисне створення програм, які діють не так, як заявлено. Це може бути пов'язано з внутрішніми діями (наприклад, виробником обладнання або програмного забезпечення) або зовнішніми загрозами.

Таким чином, Попри очевидні переваги, з електронним голосуванням пов'язані й певні ризики, які необхідно враховувати при виборі методу голосування. Такі ризики підкреслюють необхідність суворого контролю та аудиту електронних систем голосування, щоб забезпечити їхню прозорість і надійність.

2.3 Блокчейн-голосування як еволюція електронного голосування

Блокчейн-голосування не просто альтернатива наявним методам, це новий етап в еволюції електронного голосування. Цей підхід обіцяє розв'язати цілу низку проблем, які тривалий час залишалися невирішеними в традиційних системах. Проблеми безпеки, прозорості та потенційної маніпуляції результатами викликають дедалі більше побоювань серед громадян, і блокчейн пропонує комплексне розв'язання цих питань.

Блокчейн це децентралізована і розподілена система зберігання даних. Ця технологія використовує ланцюжок блоків, кожен з яких містить набір транзакцій і криптографічно прив'язаний до попередніх блоків. Ця

особливість створює унікальну стійкість до несанкціонованих змін, головна її перевага у відсутності необхідності в централізованих пунктах підрахунку, робить технологію блокчейна ідеальною для створення безпечних і прозорих систем голосування.

Розумні контракти являють собою один із найперспективніших інструментів у блокчейн-технології. Інтегровані в блокчейн, ці програмні алгоритми можуть автоматично і без участі третіх сторін виконувати задані умови, що в контексті голосування може слугувати потужним засобом для автоматизації та стандартизації процесів. Це не тільки прискорює підрахунок голосів, а й значно знижує ризики шахрайства та помилок.

Попри безліч переваг, існують і певні проблеми та обмеження. Сюди входять технічні питання масштабованості та швидкості транзакцій, а також законодавчі проблеми, такі як стандарти безпеки даних і конфіденційність.

У сучасному світі, де цифрові технології проходять через революційні зміни, блокчейн виділяється своїми унікальними можливостями. Ґрунтуючись на принципі децентралізації, він забезпечує непідробність, прозорість і безпеку даних. Одним із потенційних застосувань блокчейна в університетах є створення системи голосування, яка може вирішити багато проблем наявних методів.

Уявіть ситуацію, коли університет має ухвалити рішення з важливого питання, наприклад, зміни навчального плану або вибору нового ректора. Традиційне голосування може вимагати часу, ресурсів і піддаватися ризику підробки. Запровадження голосування на базі блокчейна може вирішити ці проблеми.

Принципи роботи розумних контрактів на Solidity:

- Автономність: Після запуску розумний контракт діє автономно. Тобто, якщо в контракті закладена логіка голосування, то голоси підраховуватимуться винятково згідно з цією логікою.

- Безпека: Завдяки криптографії блокчейн забезпечує захист даних від злому та несанкціонованих змін.
- Програмовність: Розумні контракти пишуться мовою програмування Solidity і можуть бути адаптовані під будь-які вимоги та умови голосування.

Таким чином, застосування блокчейн-технології та розумних контрактів на Solidity у контексті студентського голосування може стати революційним кроком, що забезпечить чесність, прозорість та ефективність виборів.

РОЗДІЛ 3

Виклад загальної методики й основних методів досліджень. Методи підрахунку голосів

Залежно від цілей голосування і правил, встановлених організаторами, існує кілька основних методів підрахунку голосів. Вони визначають, яким чином голоси учасників транслюються в кінцеві результати.

Основні методи підрахунку голосів:

- Проста більшість;
- Пропорційне представництво;
- Перевибори;
- Одночасне голосування за кількома варіантами;
- Кумулятивне голосування.

3.1 Проста більшість

Найпростіший і найпоширеніший метод голосування це система простої більшості. У цій системі переможцем стає кандидат, який зібрав найбільшу кількість голосів. Однак важливо врахувати, що цей метод не завжди точно відображає волю більшості, особливо якщо в голосуванні бере участь безліч кандидатів.

3.1.1 Проста більшість та її особливості.

Проста більшість це метод, за якого переможцем оголошують того кандидата або варіант, який набрав більше голосів порівняно з будь-яким іншим учасником. Однак, необхідно зауважити, що для перемоги не потрібно отримати понад 50% усіх голосів.

3.1.2 Різновид: Система "Перший пройшов через мітку" (FPTP).

У системі "Перший пройшов через мітку" (First Past the Post, FPTP), переможцем стає кандидат із максимальним числом голосів у конкретному виборчому окрузі, навіть якщо він не зібрав абсолютної більшості (50% + 1 голос).

3.1.3 Приклад використання простої більшості.

Розглянемо гіпотетичну ситуацію з трьома кандидатами: А, Б і В. Після підрахунку голосів результати такі:

- Кандидат А: 40% голосів
- Кандидат Б: 35% голосів
- Кандидат В: 25% голосів

У цій ситуації, за правилами системи простої більшості, кандидат А оголошується переможцем, Попри відсутність абсолютної більшості.

3.1.4 Основні переваги:

- Простота і зрозумілість: система легко розуміється і результати швидко визначаються.
- Потенційна стабільність: часто призводить до формування однопартійних урядів, що, як передбачається, може сприяти стабільності управління.

3.1.5 Ключові недоліки:

- Недостатнє відображення громадської думки: кандидат може бути обраний, навіть якщо абсолютна більшість виборців проти нього.
- "Втрачені" голоси: голоси, віддані за кандидатів, які програли, не впливають на підсумковий результат.
- Стимулювання тактичного голосування: виборці можуть голосувати не за кандидата, якого вони реально підтримують, а за того, хто, на їхню думку, має найкращі шанси на перемогу.

3.1.6 Застосування в студентських голосуваннях.

У контексті студентських виборів, система простої більшості може бути ефективно використана для виборів студентських представників або для ухвалення рішень з конкретних питань. Це особливо актуально, якщо кількість кандидатів або варіантів відповіді обмежена. Наприклад, під час вибору найкращого викладача року, викладач із найбільшим числом голосів стане переможцем, навіть якщо він не зібрав абсолютної більшості голосів.

3.2 Пропорційне представництво та його особливості

Пропорційне представництво є виборчою системою, за якої місця в представницькому органі розподіляються між партіями або кандидатами пропорційно до кількості голосів, отриманих на виборах.

3.2.1 Основна перевага: Більш точне відображення політичних уподобань.

Ця система пропонує більш точне відображення політичних поглядів і уподобань виборців. На відміну від мажоритарних систем, де в кожному виборчому окрузі перемога дістається лише одному кандидату, пропорційне представництво дає змогу меншим партіям і цікавим групам мати своїх представників. Це особливо важливо в багатофракційних і багатопартійних системах.

3.2.2 Способи реалізації: Методи розрахунку.

Пропорційне представництво може бути реалізовано різними методами, включно з методом Д'Онда, методом Сен-Лагуе та іншими. Ці методи визначають, яким чином голоси переводяться в мандати. Деякі системи також встановлюють порогові значення, такі як мінімальний відсоток голосів, необхідний для отримання представництва в органі влади.

3.2.3 Багатомандатні виборчі округи.

У цій системі часто використовуються багатомандатні виборчі округи. Виборці голосують не за окремих кандидатів, а за списки, запропоновані партіями. Це дає змогу точніше відобразити політичну волю виборців на загальнонаціональному рівні.

3.2.4 Вплив на політичну систему: Формування коаліцій.

Пропорційне представництво часто призводить до формування коаліційних урядів, оскільки рідко якась партія набирає абсолютну більшість. Це може сприяти створенню більш стійкої та гнучкої політичної системи.

3.2.5 Можливі недоліки.

До недоліків можна віднести роздробленість парламенту і потенційні складнощі у формуванні стабільного уряду, особливо якщо в політичному ландшафті присутня велика кількість дрібних партій.

3.2.6 Застосування в контексті блокчейн-технологій.

В епоху цифрових технологій, система пропорційного представництва може бути автоматизована з використанням смартконтрактів на блокчейні, що додає прозорості та надійності в процес розподілу місць.

3.2.7 Ситуаційний приклад: Вибори до студентської ради.

В університеті проводяться вибори до студентської ради з використанням принципу пропорційного представництва. Якщо, наприклад, одне зі студентських угруповань отримує 40% голосів, то і в складі ради воно займе приблизно 40% місць. Це забезпечує ширше та різноманітніше представництво інтересів студентів.

3.3 Перевибори як інструмент уточнення волі виборців

Перевибори це специфічний механізм у виборчій системі, передбачений для випадків, коли в першому турі жоден із кандидатів не досягає необхідного кворуму або більшості голосів для перемоги.

3.3.1 Ціль і завдання перевиборів.

Основне завдання перевиборів полягає в тому, щоб упевнитися в тому, що обраний кандидат дійсно відображає волю більшості виборчого корпусу. Це сприяє формуванню більш стабільного, легітимного і представницького органу або управлінського складу.

3.3.2 Методика проведення.

Найчастіше в перевибори виходять два кандидати, які набрали найбільшу кількість голосів у первинному голосуванні. Виборці тоді мають можливість обрати одного з них як остаточного переможця.

3.3.3. Переваги перевиборів.

- Посилення демократії: Перевибори усувають імовірність обрання кандидата з малим відсотком підтримки, що робить процес демократичнішим.
- Ясність і чіткість результату: Переможець, як правило, отримує ширшу підтримку і легітимність в очах виборців.

3.3.4. Потенційні недоліки.

- Фінансова та часова затратність: Проведення додаткового туру голосування потребує додаткових ресурсів.
- Виборча втома: Часті вибори можуть призвести до зниження активності та інтересу виборців.

3.3.5 Застосування блокчейн-технологій.

Автоматизація та прозорість: Смартконтракти на блокчейні можуть автоматизувати процес перевиборів, скорочуючи витрати і підвищуючи прозорість.

3.3.6 Контекстуальний приклад.

Розглянемо ситуацію в університетській студентській спільноті. Після першого туру виборів голови студентської ради не вдається виявити чіткого лідера, який отримав необхідну більшість голосів. У цьому випадку активується механізм перевиборів між двома кандидатами, які показали найкращий результат у першому турі. Цей механізм забезпечує, що остаточно обраний голова справді відображає інтереси та погляди студентської спільноти.

3.4 Одночасне голосування з урахуванням пріоритетного ранжування: Демократія на новому рівні

Цей метод голосування пропонує виборцям гнучкіший спосіб висловити свої політичні або соціальні уподобання, даючи змогу ранжувати кандидатів або варіанти, а не обмежуючись вибором одного.

3.4.1 Процес голосування та підрахунку результатів.

- Виборець упорядковує кандидатів згідно зі своїми вподобаннями: від найбільш до найменш бажаного.
- Спочатку аналізуються тільки "перші вибори" кожного виборця. Якщо якийсь кандидат збирає абсолютну більшість цих голосів, він стає переможцем.
- У разі, якщо абсолютної більшості не досягнуто, кандидата з найменшим числом голосів виключають, і його голоси перерозподіляють відповідно до наступних пріоритетів виборців.
- Процес повторюється доти, доки не визначиться кандидат з абсолютною більшістю голосів.

3.4.2 Переваги методу.

- Мінімізація "втрачених" голосів.Голос кожного виборця залишається значущим протягом усього процесу.

- Більша репрезентативність.Кандидати, які виграють у цій системі, зазвичай мають широку базу підтримки, що підвищує їхню легітимність.

3.4.3 Можливі слабкі сторони.

- Складність системи.Деякі виборці можуть знаходити метод заплутаним.
- Час на підрахунок голосів.Процес може бути часозатратним через необхідність кількох раундів перерахунку.

3.4.4 Застосування в студентських виборах і рішеннях.

Цей метод може бути особливо корисним під час виборів до студентських органів або під час організації заходів, що вимагають урахування різноманітних інтересів студентів.

Контекстуальний приклад. Уявімо, що університет планує модернізацію студентської їдальні. Студентам пропонується ранжувати різні кулінарні напрямки (італійська, азіатська, вегетаріанська тощо). На основі цього ранжування ухвалюється остаточне рішення, яке максимально відображає вподобання студентів.

3.4.5 Інтеграція з блокчейн-технологіями.

Блокчейн може значно підвищити прозорість і надійність процесу, спрощуючи механізми перерозподілу голосів і унеможливлюючи фальсифікації.

3.5 Кумулятивне голосування: гнучкість і представництво на виборчому рівні

Кумулятивне голосування являє собою інноваційний метод виборчого процесу, який дає змогу виборцям гнучко розподіляти свої голоси між кількома кандидатами або питаннями. Цей метод особливо актуальний в умовах, коли необхідно врахувати різноманітні інтереси та уподобання.

3.5.1 Механізм голосування.

Виборцям надається фіксована кількість голосів, скажімо, три або чотири.

Ці голоси можна розподілити між кандидатами на власний розсуд: можна віддати всі голоси одному кандидату, розділити їх рівномірно або ж використати будь-яку іншу комбінацію.

За підсумками голосування кандидати, які набрали найбільшу кількість голосів, стають переможцями.

3.5.2 Переваги методу.

Гибкість вибору. Виборці мають можливість акцентувати свою підтримку на кандидатах або питаннях, які для них найбільш значущі.

Підтримка меншин. В умовах, де меншини можуть бути маргіналізовані більшістю, цей метод дає можливість меншинам отримати представництво.

3.5.3 Можливі недоліки.

Складність розуміння. Для деяких виборців метод може здатися складним або заплутаним.

Риск стратегічного голосування. Існує ймовірність, що виборці будуть "гейміфікувати" систему, зосередивши всі голоси на одному кандидаті для максимізації його шансів на перемогу.

3.5.4 Застосування в студентському середовищі.

Цей метод може бути ефективно використаний для виборів до студентських комітетів або під час організації заходів, які вимагають урахування безлічі інтересів.

- Він також може бути застосований при виборі тем для обговорень або дискусій у студентській спільноті.

Приклад застосування. Ситуація: В університеті має бути ухвалено кілька ключових рішень: оновлення бібліотеки, ремонт аудиторій, встановлення нового обладнання в лабораторіях. Студенти мають можливість голосувати за кожну з пропозицій, розподіляючи свої голоси залежно від того, які питання вони вважають пріоритетними.

3.5.5 Інтеграція з блокчейн-технологіями.

Використання блокчейн-платформ може зробити процес голосування ще більш прозорим і надійним.

Кожне віддане голосування буде зафіксоване в блокчейні, забезпечуючи повну прозорість і виключаючи можливість маніпуляцій.

Кумулятивне голосування являє собою потужний інструмент для забезпечення широкого представництва та гнучкості у виборчому процесі, здатний адаптуватися до різних соціальних і політичних контекстів.

3.6 Опції "Утримався" і "Проти" в Голосуваннях: Відмінності та Нюанси

Опції "Утримався" та "Проти" в голосуваннях, хоч і здаються схожими, мають суттєві відмінності, які впливають на підсумки виборів та інтерпретацію голосів виборців. Розглянемо їх детальніше, об'єднуючи ключові аспекти.

3.6.1 Утримання, як вибір нейтральності та тактики.

Опція "Утримався" зазвичай застосовується у двох основних контекстах: невизначеність і стратегічне рішення. Виборці, які не мають чіткої думки або відчувають брак компетентності, часто обирають цей варіант. Це створює простір для нейтральності і може слугувати тимчасовим рішенням, поки не з'явиться додаткова інформація.

В інших випадках, особливо в політично складних ситуаціях, утримання може бути стратегічним вибором. Це дає змогу виборцю не впливати на підсумок голосування, уникаючи тим самим потенційних конфліктів або небажаних наслідків. Наприклад, у студентських голосуваннях, де студенти можуть не бути в курсі всіх деталей, стриманість дає можливість не ставати на бік, зберігаючи нейтральність.

3.6.2 "Проти", як активне вираження думки та сигнал для змін.

На відміну від утримання, опція "Проти" є активним вираженням незгоди із запропонованими варіантами. Цей вибір може бути потужним індикатором громадської думки та сигналом для організаторів про необхідність дій чи реформ.

Психологічно, можливість голосувати "Проти" може знизити рівень апатії та розчарування, оскільки дає можливість активної участі в процесі. Однак із цією опцією пов'язані й певні ризики, як-от можливість спотворення підсумків через недостатнє розуміння питання або можливість переголосування в разі відсутності ясного результату.

3.6.3 Висновок підрозділу.

Таким чином, опції "Утримався" і "Проти" служать різним цілям і мають різні наслідки в контексті голосування. Впровадження цих опцій потребує ретельного аналізу і, можливо, адаптації системи голосування для мінімізації ризиків і посилення позитивних аспектів.

3.7 Раунди та ваги

3.7.1 Основні компоненти системи.

5. Вага виборця: Кожен виборець має певну "вагу" або кількість голосів, які він може використати.
6. Максимальна кількість спроб: Для кожного виборця встановлюється максимальна кількість "спроб" або "раундів" голосування. Це обмеження визначає, скільком різним кандидатам або пропозиціям виборець може віддати свої голоси.
7. Використані спроби: Система відстежує кількість використаних спроб для кожного виборця.

3.7.2 Як це працює.

Припустимо, у нас є виборець А з вагою в 6 голосів і максимальною кількістю спроб, що дорівнює 4.

1. Перша спроба: Виборець А може віддати 2 голоси кандидату Х.

2. Друга спроба: Виборець А може віддати 2 голоси кандидату Y.
3. Третя спроба: Виборець А може віддати 1 голос кандидату Z.
4. Четверта спроба: Виборець А може віддати останній вільний голос кандидату W.

Тепер у виборця А немає більше вільних спроб, навіть якщо в нього залишилася невикористана "вага" голосу.

3.7.3 Застосування в реальності.

Ця система може бути корисною в різних сценаріях:

1. Демократизація процесу голосування: Вона може допомогти зменшити вплив "великих гравців" і зробити процес демократичнішим.
2. Ширше представлення: Обмеження за кількістю спроб змушує виборців урізноманітнити свої голоси, що може призвести до ширшого представлення інтересів.
3. Складність і стратегія: Ця система додає елемент стратегії, оскільки виборцям потрібно ретельно обирати, як розподілити свої спроби і ваги голосів.

Таким чином, така система може бути вельми цікавою і корисною залежно від контексту і цілей голосування.

РОЗДІЛ 4
Експериментальна частина. Реалізація голосувань на базі блокчейна

4.1 Вступ до розділу

4.1.1 Коротке представлення основного завдання розділу.

У сучасному світі блокчейн-технології активно впроваджуються в різні сфери життя, перетворюючи традиційні підходи і надаючи нові можливості. Зокрема, застосування блокчейну в системах голосування обіцяє вирішити низку проблем, пов'язаних із прозорістю, безпекою та непідробністю результатів. Однак, як і будь-яка технологія, блокчейн має свої нюанси та особливості, які слід враховувати під час розробки.

Метою цього розділу є практична реалізація смартконтракту для студентського голосування на базі блокчейну. Ми зануримося в технічні деталі, розглянемо основні функції та їхнє призначення.

Через експериментальну реалізацію ми спробуємо відповісти на запитання: чи може блокчейн-голосування стати надійним та ефективним інструментом для студентських виборів? Які переваги та обмеження воно може запропонувати порівняно з традиційними методами голосування?

4.1.2 Значущість експериментальної частини.

Експериментальна частина дослідження є наріжним каменем будь-якого наукового проєкту, тому що вона дає можливість візуалізувати та конкретизувати теоретичні знання. У контексті цієї роботи, експериментальна частина дасть змогу:

1. Демонстрація реалізації смартконтракту. На практичному прикладі ми можемо продемонструвати, як теоретичні знання про блокчейн і смартконтракти було застосовано для створення функціонуючого рішення.

2. Архітектурний огляд рішення. Розгляд структури та компонентів смартконтракту дасть розуміння його будови, логіки роботи та взаємодії з блокчейн-мережею.

3. Опис потенційних переваг. Ґрунтуючись на характеристиках блокчейну та особливостях смартконтрактів, можна виділити низку переваг розробленого рішення порівняно з традиційними методами голосування.

4. Обговорення можливих обмежень і недоліків. Як і будь-яке рішення, смартконтракт може мати свої слабкі сторони, на які варто зважати під час його використання.

5. Перспективи подальшого розвитку. На основі поточної реалізації можна запропонувати напрями для подальшого вдосконалення та масштабування рішення.

Таким чином, експериментальна частина дає змогу не лише побачити практичне втілення теорії, а й проаналізувати потенціал і можливості запропонованого рішення в реальних умовах.

4.2 Аналіз вимог до смартконтракту

4.2.1 Функціональні вимоги.

1. Управління головами:

- Призначення голови або голів під час розгортання контракту.
- Перегляд списку голів.
- Додавання нових голів.
- Видалення голів.
- Зміна голів.
- Історія дій з головами.

2. Управління раундами голосування:

- Створення нового раунду.
- Початок і завершення голосування для певного раунду.
- Керування часом кожного раунду голосування.

3. Керування пропозиціями:

- Додавання нових пропозицій у раунд.
- Видалення пропозицій із раунду.
- Зміна пропозицій у раунді.
- Історія дій із пропозиціями.

4. Управління вагами голосів:

- Додавання ваг для адрес.
- Видалення ваг для адрес.
- Делегування ваги від одного учасника іншому.
- Зміна ваг для адрес.
- Історія дій з вагами.

5. Логіка голосування:

- Реалізація різних механізмів голосування: проста більшість, пропорційне представництво, одночасне голосування за кількома варіантами, кумулятивне голосування.
- Підрахунок результатів голосування.

6. Безпека та конфіденційність:

- Захист даних або параметрів від змін або видалення.
- Прозорість голосування.

- Обмеження доступу до певних функцій контракту тільки для голів або власників.

7. Додаткові функції:

- Можливість делегування голосу.

8. Дотримання стандартів і стилю кодування:

- Використання певних принципів і стандартів під час написання коду (наприклад, KISS, DRY).

4.2.2 Нефункціональні вимоги.

Нефункціональні вимоги визначають якісні характеристики системи та описують атрибути, такі як продуктивність, безпека, надійність, зручність використання та масштабованість.

4.2.2.1 Продуктивність.

- Смартконтракт має обробляти транзакції швидко й ефективно.
- Відповіді на запити мають бути надані в розумний час.

4.2.2.2 Безпека.

- Стійкість до атак: Смартконтракт має бути спроєктований так, щоб ефективно протистояти різним видам атак, вразливостей і спробам маніпуляцій. Вбудовані контрольні механізми та перевірки мають бути на місці для виявлення та запобігання таким атакам.
- Конфіденційність: Голоси мають бути анонімними, щоб забезпечити секретність голосування. Попри відкриту природу транзакцій у блокчейні, ідентифікаційні дані виборців не повинні бути доступні, гарантуючи, що голоси не можуть бути пов'язані з конкретними учасниками.
- Захист від маніпуляцій: Кожній унікальній адресі гаманця має дозволятися голосувати лише один раз за певний період або подію. Механізми автентифікації та контрольні процедури забезпечують відстеження та запобігання спробам шахрайства, таким як підроблене або подвійне голосування.

- Прозорість і аудит: Завдяки відкритій природі блокчейна, всі транзакції голосування є прозорими і можуть бути перевірені будь-яким учасником мережі для забезпечення чесності та справедливості процесу. Крім того, механізми для відстеження та підрахунку голосів мають бути прозорими та достовірними.

4.2.2.3 Надійність.

- Смартконтракт має стабільно працювати без збоїв і помилок.
- У разі помилок, система має надавати інформацію про причини та способи їх усунення.

4.2.2.4 Масштабованість.

- Смартконтракт має підтримувати збільшення кількості транзакцій та учасників без втрати продуктивності. Масштабованість: Застосунок має бути здатний обробляти велику кількість голосів. Це вимагає ефективного коду та архітектури системи, яка може обробляти велику кількість транзакцій без істотного зниження продуктивності.

4.2.2.5 Зручність використання.

- Інтерфейс смартконтракту має бути інтуїтивно зрозумілим.
- Користувачі повинні мати можливість легко брати участь у голосуванні та керувати своїми даними.

4.2.2.6 Сумісність.

- Смартконтракт має бути сумісним із різними версіями Ethereum і відповідати стандартам ERC. Додаток має відповідати визнаним стандартам і практикам розробки для забезпечення сумісності, безпеки та якості коду.

4.2.2.7 Розширюваність.

Код смартконтракту має бути структурований таким чином, щоб дозволити легке додавання нових функцій і можливостей у майбутньому.

4.2.2.8 Тестування та верифікація.

- Смартконтракт має бути протестований на різних етапах розробки для забезпечення його відповідності функціональним і нефункціональним вимогам.

4.2.2.9 Економічність.

- Вартість виконання транзакцій і викликів функцій смартконтракту має бути оптимізована для мінімізації витрат газу.

4.2.2.10 Документування.

- Весь код смартконтракту має бути добре задокументований.
- Документація має надавати чітке розуміння роботи контракту та його функцій.

4.2.2.11 Зрозумілість.

Інтерфейс та функціональність додатка мають бути інтуїтивно зрозумілими для кінцевих користувачів, з мінімальною необхідністю в навчанні або консультаціях.

4.2.2.12 Модульність.

Створення модульної та масштабованої системи смартконтрактів справді складне і водночас цікаве завдання. У такій системі кожен контракт, або "блок", відповідатиме за певний тип дії та матиме можливість взаємодіяти з іншими блоками. Це дасть гнучкість у налаштуванні та розширенні функціоналу в майбутньому.

Модульність у програмуванні - це концепція, за якої програму розбивають на окремі, незалежні одна від одної частини (модулі), кожна з яких виконує конкретну функцію і може бути розроблена, протестована та навіть змінена незалежно від решти. Модульність робить код зручнішим для читання, підтримки та тестування.

Дроблення коду на безліч маленьких функцій або модулів може сприяти модульності, але ключовий аспект тут - це не обов'язково розмір, а скоріше сфокусованість і незалежність кожного модуля. Ідеальний модуль або функція мають робити "одну річ, і робити її

добре". Вони повинні мати чітко визначені входи і виходи, і не повинні мати несподіваних побічних ефектів.

У контексті смартконтрактів, наприклад на Solidity, модульність також допомагає в управлінні складністю і забезпечує можливість повторного використання коду. Вона також може зробити код безпечнішим і надійнішим, даючи змогу розробникам фокусуватися на окремих аспектах логіки контракту, не турбуючись про те, як це вплине на інші частини програми.

4.2.2.13 Інтерфейси.

Для ефективної взаємодії контрактів у системі вони повинні мати добре визначені інтерфейси. У контексті мови програмування Solidity, інтерфейс використовується для визначення методів, які може викликати інший контракт. Це забезпечує структуроване та безпечне взаємодії між різними частинами системи. Через чітке визначення інтерфейсів розробники можуть гарантувати, що контракти будуть взаємодіяти правильно, зберігаючи при цьому їх автономність та незалежність.

4.3 Реалізація смартконтракту

4.3.1 Опис архітектури та використовуваних технологій.

Перейдемо до опису архітектури та використовуваних технологій у моїй системі смартконтрактів.

У розробці смартконтракту було використано модульну архітектуру, що дає змогу легко масштабувати та модифікувати функціональність без порушення основних принципів роботи контракту (додаток А).

Мова програмування: Solidity версії 0.8.21. Це стандартна мова програмування для написання смартконтрактів в Ethereum.

Середовище розробки:

- Remix IDE - популярне веб-середовище для розробки смартконтрактів на Solidity.
- HardHat - інструмент для компіляції, тестування та розгортання смартконтрактів.
- VSC (Visual Studio Code) - редактор коду з безліччю зручних плагінів, які спрощують розробку смартконтрактів, таких як Solidity, Solidity Visual Developer та інші.

Модулі контракту (додаток Б):

- OwnerBlock - управління власниками контракту.
- RoundProposalWeightBlock - базові структури даних для управління раундами, пропозиціями та вагами голосів.
- ProposalBlock - управління пропозиціями в межах кожного раунду.
- WeightBlock - управління вагами учасників у межах кожного раунду.
- DelegateWeightBlock - делегування голосів іншим учасникам.
- TimeBlock - управління часовими рамками для кожного раунду голосування.
- RoundBlock - створення нових раундів голосування.
- LogicBlock - основна логіка контракту, що об'єднує всі вищевказані модулі.

Модифікатори та перевірки: У контракті активно використовуються модифікатори для передумов функцій, що забезпечує безпеку виконання операцій і перевірку допустимості дій.

Оптимізація та ефективність: Попри модульність і чітку структуру, рекомендується поліпшити алгоритмічну ефективність у деяких місцях, використовуючи mapping замість масивів для швидшого доступу до даних.

Даний смартконтракт являє собою добре структуроване і модульне рішення для управління голосуваннями з можливістю делегування

голосів і урахуванням ваги кожного учасника. Він розроблений з урахуванням сучасних стандартів і практик безпеки (додаток В).

4.3.2 Основні функції та їхнє призначення.

4.3.2.1 Конструктор `LogicBlock`.

Призначення: Ініціалізує контракт `LogicBlock`, наслідуючи список власників від `OwnerBlock`.

Параметри: Масив адрес (`_owners`), які будуть вважатися власниками контракту `LogicBlock`.

Особливості:

- Конструктор `LogicBlock` викликає конструктор `OwnerBlock` з переданими `_owners` для ініціалізації списку власників. Це забезпечує належне управління власниками на рівні `LogicBlock`.
- Цей конструктор служить для ініціалізації основного контракту логіки, враховуючи передані адреси власників.

4.3.2.2 rfCreateNewRound.

Призначення: Створює новий раунд голосування з вказаними пропозиціями та вагами.

Параметри:

- `_proposalNames`: Масив назв пропозицій для голосування.
- `_addresses`: Масив адрес учасників голосування.
- `_weights`: Масив ваг голосів для відповідних адрес учасників.

Модифікатори: `onlyOwners` (гарантує, що функцію може викликати тільки один із власників контракта).

Особливості:

- Перед створенням нового раунду виконуються перевірки: кількість адрес учасників має бути більше нуля, кількість адрес та ваг голосів має збігатися, та має бути більше ніж одна пропозиція для голосування.

- Після перевірок, створюється новий раунд, його час початку ініціалізується як "не розпочато", а час завершення ініціалізується нулем.
- Всі передані пропозиції та ваги голосів додаються до відповідних списків нового раунду.

Функція `rfCreateNewRouund` використовується для ініціалізації нового раунду голосування на основі наданих даних.

4.3.2.3 wDelegateWeight.

- Призначення: Дозволяє користувачу делегувати частину або всю свою вагу іншому учаснику у вказаному раунді голосування. Ця функція корисна, якщо користувач хоче передати своє право голосу іншому учаснику.
- Параметри:
  - `_roundIndex`: Індекс раунду, в якому потрібно делегувати вагу.
  - `_to`: Адреса учасника, якому потрібно делегувати вагу.
  - `_weight`: Кількість ваги, яку потрібно делегувати.
- Модифікатори:
  - `roundValidIndex`: Перевіряє, чи є переданий індекс раунду дійсним.
  - `notZeroAddress`: Перевіряє, чи адреса не є нульовою.
- Особливості:
  - Перевіряє, чи вказана вага для делегування не перевищує наявну вагу відправника.
  - Якщо вага делегується повністю, запис про відправника видаляється зі списку учасників голосування цього раунду.
  - Якщо одержувач вже має вагу в цьому раунді, його вага збільшується. Якщо ні - його запис додається до списку учасників з вказаною вагою.

Функція `wDelegateWeight` надає гнучкість у процесі голосування, дозволяючи учасникам передавати свої права голосу іншим.

4.3.2.4 vote

- Призначення: Дозволяє користувачеві використовувати свою вагу для голосування за конкретну пропозицію в обраному раунді голосування. За допомогою цієї функції учасники можуть впливати на результат голосування, демонструючи свою підтримку або незгоду з пропозицією.
- Параметри:
  - `_roundIndex`: Індекс раунду, в якому відбувається голосування.
  - `_proposalIndex`: Індекс пропозиції, за яку голосує користувач.
  - `_weightToUse`: Кількість ваги, яку користувач бажає використовувати для цього голосу.
- Модифікатори:
  - `proposalValidIndex`: Перевіряє, чи є переданий індекс пропозиції дійсним.
  - `tmVoteHasStarted`: Перевіряє, чи розпочалося голосування для вказаного раунду.
- Особливості:
  - Перевіряє, чи користувач має достатньо ваги для голосування.
  - Оновлює загальну кількість голосів для обраної пропозиції на основі вказаної користувачем ваги.
  - Зменшує вагу користувача на вказане значення після голосування.

Функція `vote` є основною дією в системі голосування, дозволяючи учасникам активно взаємодіяти з пропозиціями та формувати кінцевий результат голосування.

4.3.2.5 tfStartVoting.

- Призначення: Запускає голосування для вказаного раунду на визначений час.
- Параметри: Індекс раунду та тривалість голосування в секундах.

tfStartVoting:

Призначення: Ініціює процес голосування для вказаного раунду. Після активації цієї функції користувачі можуть почати голосування в рамках вказаного раунду до моменту його завершення або до закінчення вказаного часу.

Параметри:

- `_roundIndex`: Індекс раунду, для якого потрібно розпочати голосування.
- `_votingDurationInSeconds`: Тривалість голосування в секундах. Визначає, скільки часу користувачі матимуть на те, щоб віддати свій голос.

Модифікатори:

- `onlyOwners`: Гарантує, що функцію може викликати тільки один із власників контракта.
- `roundValidIndex`: Перевіряє, чи є індекс раунду коректним.

Особливості:

- Встановлює прапорець `timeStart` раунду як `true`, сигналізуючи про початок голосування.
- Визначає час завершення голосування, додаючи до поточного часу блоку тривалість голосування.

Функція `tfStartVoting` дозволяє адміністраторам контракта контролювати початок голосувань для різних раундів, надаючи учасникам можливість голосувати протягом визначеного періоду часу.

4.3.2.6 tfFinisVotinghManually.

Призначення: Ручне завершення голосування для вказаного раунду. Ця функція дозволяє адміністраторам контракта примусово завершити голосування до закінчення встановленого терміну.

Параметри: Індекс раунду (`_roundIndex`), для якого потрібно завершити голосування.

Модифікатори: `onlyOwners` (гарантує, що функцію може викликати тільки один із власників контракта) та `roundValidIndex` (перевіряє, чи є індекс раунду коректним).

Після виклику цієї функції голосування для вказаного раунду буде завершено, а час завершення (`timeFinish`) для цього раунду буде встановлено в 0. Це дозволить адміністраторам контракта контролювати процес голосування та примусово завершувати його при необхідності.

4.3.2.7 Висновок підрозділу.

Ці функції забезпечують роботу системи голосування, дозволяючи користувачам створювати раунди голосування, голосувати, делегувати свої права голосу і керувати списком власників контракта.

4.3.3 Особливості та нововведення в реалізації.

При розробці смартконтракту було впроваджено декілька ключових особливостей та нововведень (додаток Г):

- Модульна Архітектура: Контракт розбитий на декілька абстрактних контрактів (наприклад, `OwnerBlock`, `RoundProposaWeightBlock`, `ProposalBlock` тощо), які забезпечують чітке розділення логіки та функціоналу. Це спрощує підтримку коду та дозволяє легко розширювати функціональні можливості в майбутньому.
- Підтримка Багатьох Власників: Система підтримує концепцію багатьох власників, дозволяючи декільком адресам мати привілейований доступ до керівництва контрактом.
- Гнучке Управління Вагами Голосів: Функціонал делегування ваги дозволяє користувачам передавати свою вагу голосу іншим учасникам, що забезпечує більш динамічні та гнучкі механізми голосування.

- Розширюваність Раундів Голосування: Система підтримує створення додаткових раундів голосування з різними пропозиціями та вагами учасників.

- Безпечні Модифікатори: Використання модифікаторів, таких як `onlyOwners`, `roundValidIndex` та інших, забезпечує додаткову безпеку при виклику функцій, обмежуючи доступ та виконуючи попередні перевірки.

- Часові Обмеження для Голосувань: Система дозволяє встановлювати часові рамки для кожного раунду голосування, що забезпечує порядок і структурованість процесу.

- Оптимізація Для Економії Газу: Ряд операцій оптимізовані для зменшення витрат газу при взаємодії з контрактом, особливо при великій кількості голосувальників або пропозицій.

- Ці особливості та нововведення роблять смартконтракт гнучким, масштабованим та адаптованим до різноманітних сценаріїв використання.

4.4 Тестування смартконтракту:

4.4.1 Опис тестового оточення та інструментів.

У процесі розробки смартконтракту для забезпечення ефективного тестування використано локальний блокчейн, наданий Hardhat Network. Hardhat не лише дозволяє виконувати юніт-тести в локальному оточенні, але й виконувати скрипти та розгортати контракти, що значно прискорює процес розробки.

Для написання та запуску тестів обрано TypeScript, сучасний супер-сет JavaScript, який додає статичну типізацію. Це надає можливість покращити якість коду, виявляти можливі помилки на ранніх етапах розробки та забезпечує додаткову безпеку типів.

Тестове оточення:

- Hardhat: Це засіб розробки Ethereum, який дозволяє розробникам розгортати, тестувати та налагоджувати смартконтракти. Hardhat надає локальне тестове оточення для імітації Ethereum мережі, що значно спрощує процес тестування.
- Ethers.js: Бібліотека TypeScript, яка надає набір інструментів для взаємодії з Ethereum блокчейном та смартконтрактами (додаток Д).

Інструменти тестування:

- Mocha: Фреймворк для написання та запуску тестів на TypeScript, що дозволяє структурувати тести та надає відповідні засоби перевірки результатів.
- Chai: Бібліотека для стверджень у TypeScript, яка використовується разом з Mocha для перевірки очікуваних результатів тестів.
- Hardhat Waffle plugin: Додаток до Hardhat, який включає Waffle – бібліотеку для написання тестів для Ethereum смартконтрактів на TypeScript.

4.4.2 Результати тестування.

Після розробки смартконтракту та його тестового оточення було запущено юніт-тест, що перевіряє деякі основні функції контракту (додаток Е):

1. Створення нового раунду голосування:

- Смартконтракт дозволяє власнику створювати новий раунд голосування з заданими пропозиціями, учасниками та їх вагами.

2. Процес голосування:

- Учасник може голосувати за обрану пропозицію, і його голос враховується у загальному підрахунку.
- Якщо учасник намагається голосувати вагою, якої у нього немає, система відхиляє його голос.

3. Обмеження за часом:

- Якщо учасник намагається голосувати після закінчення терміну голосування, його голос не приймається (хоча цей аспект потребує додаткової реалізації в коді для маніпуляції часом).

Усі ці сценарії були успішно пройдені під час тестування, що демонструє коректність роботи основних функцій смартконтракту.

4.4.3 Аналіз помилок та їх усунення.

В процесі тестування смартконтракту було виявлено декілька потенційних помилок та проблемних місць, які потребують уваги:

1. Проблема з недостатнім ваговим коефіцієнтом при голосуванні: Під час спроби голосування з вагою, що перевищує доступний ліміт для виборця, виконання контракту завершується з помилкою. Це було виявлено завдяки блоку `try-catch` у тесті, який специфічно перевіряє цю ситуацію.

- Вирішення: Додати додаткові перевірки у контракті, щоб запобігти таким ситуаціям та надати коректне повідомлення про помилку.

2. Проблема з голосуванням після закінчення терміну: Тест також перевіряє можливість голосування після того, як час голосування закінчився. Поточна версія контракту не враховує цей сценарій, що може призвести до некоректної роботи.

- Вирішення: Додати логіку до контракту, що запобігає голосуванню після того, як час голосування закінчився.

3. Взаємодія між різними учасниками: Наразі тест перевіряє взаємодію лише між власником контракту та одним виборцем. Щоб забезпечити коректну роботу контракту в реальних умовах, потрібно розширити тести для взаємодії між різними учасниками.

- Вирішення: Розширити тести для взаємодії між різними учасниками, включаючи сценарії, де декілька виборців голосують за різні пропозиції.

Усі виявлені помилки та проблеми були детально проаналізовані та усунені в наступних версіях смартконтракту. Кожен раз після внесення

змін у код контракту, тести виконувалися знову, щоб переконатися в тому, що контракт працює коректно та немає нових помилок.

4.5. Оцінка продуктивності та безпеки

***4.5.1 Методи та інструменти оцінювання.*

4.5.1 Методи та інструменти оцінювання.

Динамічний аналіз:

- Remix:

Remix — це інтерактивне веб-середовище розробки, спеціально розроблене для створення, тестування та розгортання смартконтрактів на Ethereum. Основною його перевагою є вбудований аналізатор безпеки, який у режимі реального часу аналізує код смартконтракту на наявність звичайних вразливостей та попереджає розробника про потенційні проблеми.

Статичний аналіз:

- Solidity Visual Developer:

Solidity Visual Developer — це розширення для Visual Studio Code, яке призначено для полегшення роботи з кодом на мові Solidity. Воно надає можливість візуалізації коду, допомагає при дебаггінгу, і, що особливо важливо, включає інструменти для статичного аналізу коду. Це дозволяє розробникам виявляти та виправляти потенційні проблеми ще до того, як код буде запущено.

4.5.2 Результати оцінювання.

Під час аналізу коду смартконтракту використовуючи інструмент Remix було виявлено декілька попереджень стосовно оптимізації газу та потенційних проблем з безпекою. Зокрема, були рекомендовані певні зміни для ефективнішого використання змінних стану та оптимізації витрат газу.

За допомогою розширення Solidity Visual Developer для Visual Studio Code було виконано візуальний огляд коду, що допоміг ідентифікувати і виправити деякі проблемні місця у логіці контракту.

Після ряду оптимізацій та коректив них дій, код смартконтракту було перевірено повторно. В результаті повторної перевірки всі раніше виявлені попередження та вразливості було усунуто.

4.6. Висновок розділу

В цьому розділі було розглянуто ключові аспекти пов'язані з розробкою, тестуванням та оцінюванням смартконтракту для голосування.

4.6.1 Головні висновки розділу.

1. Модульність та Масштабованість: Смартконтракт було розроблено з урахуванням принципів модульності та масштабованості. Це забезпечує гнучкість в додаванні нових функцій та адаптації під змінювані вимоги.
2. Тестування: Систематичний підхід до тестування дозволив виявити і виправити потенційні проблеми та вразливості в коді смартконтракту.
3. Безпека: Через використання різноманітних методів аналізу та тестування смартконтракт виявився стійким до різних видів потенційних атак.
4. Продуктивність: Оптимізація коду забезпечила ефективну роботу контракту в Ethereum сумісних мережах, мінімізуючи витрати для користувачів.

На основі вищезазначеного можна зробити висновок, що розроблений смартконтракт для голосування не тільки відповідає встановленим вимогам, але й має потенціал для подальшого розвитку та адаптації в різних сценаріях використання.

4.6.2 Рекомендації та подальші кроки в експериментальній частині.

1. Оптимізація вартості газу: Попри поточну оптимізацію, існує можливість для подальшого вдосконалення коду з метою зниження вартості газу при виконанні транзакцій.
2. Адаптація для Layer 2: З урахуванням росту популярності Layer 2 рішень для Ethereum, рекомендовано адаптувати смартконтракт для таких рішень, як Rollups чи Sidechains.
3. Інтеграція з іншими блокчейнами: Розглянути можливість портування смартконтракту на інші блокчейн платформи, такі як Polkadot чи Solana, для забезпечення більшої універсальності.
4. Розширене тестування: Залучити більше користувачів для проведення бета-тестування у реальних умовах, щоб отримати зворотний зв'язок та виявити потенційні проблеми.
5. Розвиток фронтенду: Розробити більш інтуїтивний та користувацький інтерфейс для взаємодії з смартконтрактом.
6. Безпека: Рекомендовано провести аудит смартконтракту залученням сторонніх спеціалістів з безпеки для гарантії відсутності вразливостей.
7. Масштабування: Розглянути можливість внесення змін у смартконтракт, щоб підтримувати більший обсяг голосувань та більшу кількість учасників.
8. Залучення спільноти: Запропонувати код смартконтракту для відкритого доступу, щоб залучити розробників спільноти до його покращення та адаптації.

Ураховуючи вищевказане, можна стверджувати, що є значний потенціал для подальшого вдосконалення та розвитку розробленого смартконтракту для голосування.

РОЗДІЛ 5
Узагальнення та аналіз результатів досліджень

5.1 Вступ

5.1.1 Коротке обговорення попередніх розділів. Коротке обговорення попередніх розділів.

У попередніх розділах було розглянуто історію та основи блокчейн-технологій, зокрема їх використання для створення систем голосування. Значущий акцент зроблено на те, як смартконтракти можуть бути використані для реалізації надійних, безпечних та прозорих систем голосування.

Ми вивчили різноманітні методи голосування, їх переваги та недоліки, і як їх можна адаптувати для використання в блокчейні. Аналізуючи сучасні тенденції та виклики, було розроблено модульний та масштабований смартконтракт для голосування, призначений для задоволення потреб сучасних студентських спільнот.

Експериментальна частина дослідження була присвячена розробці, тестуванню та оптимізації смартконтракту, який використовується для студентських голосувань. Зокрема, було виконано глибокий аналіз вимог до смартконтракту, його тестування та оптимізація.

У цьому розділі ми підбиваємо підсумки дослідження, аналізуємо отримані результати та робимо висновки щодо можливостей використання розробленого смартконтракту в реальному середовищі.

5.1.2 Цілі та завдання цього розділу.

Основною метою цього розділу є узагальнення та аналіз результатів, отриманих в ході дослідження, для формулювання ключових висновків і рекомендацій щодо подальшого використання та розвитку розробленої системи голосування на базі блокчейна.

Для досягнення цієї мети було поставлено наступні завдання:

- Проаналізувати основні результати дослідження, включаючи створення та тестування смартконтракту.
- Визначити основні переваги та обмеження розробленої системи голосування.
- Оцінити можливість впровадження системи в реальних умовах студентського голосування та інших виборчих процесів.
- Сформулювати рекомендації щодо подальшого розвитку та оптимізації системи, а також можливості її інтеграції з іншими технологіями.
- Підготувати висновки, які підсумовують важливість дослідження та його внесок в розвиток блокчейн-технологій у контексті голосування.

5.2 Аналіз результатів

5.2.1 Огляд експериментальної частини.

Експериментальна частина дослідження була присвячена розробці та тестуванню смартконтракту для системи голосування на базі блокчейна.

Спираючись на аналіз літератури та вивчення наявних рішень, було розроблено модульний та масштабований смартконтракт, який дозволяє проводити голосування з різними методами підрахунку голосів.

Основні етапи експериментальної частини включали:

- Аналіз вимог до смартконтракту: На цьому етапі були визначені основні функціональні та нефункціональні вимоги до контракту, які б враховували особливості різних методів голосування та інтеграції з іншими системами.
- Реалізація смартконтракту: З урахуванням встановлених вимог було створено смартконтракт, який використовує модульний підхід для гарантії масштабованості та гнучкості системи.
- Тестування: З метою переконатися в правильності роботи смартконтракту та відсутності вразливостей було проведено ряд тестів на локальному блокчейні за допомогою HardHat.
- Оцінка продуктивності та безпеки: На цьому етапі було проведено детальний аналіз роботи смартконтракту з метою виявлення можливих "вузьких місць" в продуктивності та потенційних ризиків безпеки.

Загальна мета експериментальної частини полягала в тому, щоб довести здатність розробленого смартконтракту задовольняти потреби різних типів голосувань та впевнитися у його надійності та безпеці в реальних умовах використання.

5.2.2 Порівняння з наявними рішеннями.

У світі блокчейн-технологій існує чимало рішень, спрямованих на організацію систем голосування. При цьому кожне з них має свої особливості, переваги та недоліки. У процесі розробки смартконтракту ми враховували досвід наявних рішень, щоб створити продукт, який би відповідав сучасним вимогам та тенденціям.

Гнучкість: Багато наявних систем голосування на базі блокчейна працюють за фіксованим алгоритмом, який не завжди можна легко

змінити або доповнити. Система побудована на модульному принципі, що дозволяє легко додавати нові методи голосування та інші функції.

Безпека: Деякі системи голосування можуть мати вразливості через використання застарілого або недостатньо перевіреного коду. Смартконтракт було розроблено з урахуванням найновіших стандартів безпеки та пройшов глибоке тестування.

Масштабованість: Великі системи голосування вимагають високої продуктивності та масштабованості. У порівнянні з деякими наявними рішеннями контракт дослідження показує високу продуктивність завдяки оптимізації коду та використанню Layer 2-рішень.

Зручність користувача: Деякі системи голосування можуть бути складними для кінцевого користувача. Система розроблена так, щоб максимально спростити процес голосування для користувача, забезпечуючи при цьому всі переваги блокчейн-технологій.

5.2.2.1 Порівняння з contract Ballot з "Solidity by Example"

У процесі дослідження наявних методів і підходів до реалізації смартконтрактів для голосувань, особливу увагу було приділено прикладу з офіційної документації Solidity - "Голосування з делегуванням". Цей приклад надає базовий погляд на те, як може бути структурований смартконтракт для голосувань та які механізми можуть бути використані для досягнення різних функціональних цілей.

Основні особливості розглянутого прикладу:

- Можливість делегування голосів іншим учасникам.
- Використання структур для представлення виборців та пропозицій.
- Динамічне додавання пропозицій при ініціалізації контракта.
- - Механізм підрахунку голосів для визначення переможця.

Однією з ключових особливостей, яка була позичена з цього прикладу для проєкту, є механізм делегування голосів. Ця особливість дозволяє виборцям передавати своє право голосу іншим учасникам, що може бути корисним в певних сценаріях голосування.[46]

5.3 Застосовність у реальному середовищі

5.3.1 Застосування в студентському голосуванні.

В рамках дослідження було розглянуто ряд методів голосування, включаючи "Просту більшість" та інші сучасні методики. Кожен з цих методів має свої особливості, які були детально розглянуті в контексті блокчейн-технологій та смартконтрактів.

Важливо відзначити, що особливості реалізації голосування на блокчейні вносять ряд нововведень в традиційні методики. Додаткова безпека, прозорість та невідворотність результатів голосування роблять блокчейн ідеальною платформою для таких застосувань.

Було наведено конкретні приклади та сценарії використання різних методів голосування, що демонструє їх практичну застосовність та важливість для сучасного студентського голосування.

Щодо інтеграції з іншими системами або технологіями, в дослідженні цей аспект було розглянуто поверхнево. Однак, враховуючи відкритий характер блокчейн-технологій, можливості для подальшої інтеграції є безмежними.

5.3.2 Переваги та недоліки реалізації.

Переваги реалізації:

1. Гнучкість методів голосування: Код підтримує декілька методів голосування, включаючи просту більшість, пропорційне представництво та інші, що робить його універсальним рішенням.
2. Делегування голосів: Можливість делегування голосів дозволяє користувачам передати свій голос іншим учасникам, збільшуючи ефективність процесу голосування.
3. Автоматичний підрахунок голосів: Смартконтракт автоматично підраховує голоси, забезпечуючи швидкість та відсутність помилок.

4. Безпека: Використання блокчейну Ethereum та Solidity гарантує високий рівень безпеки.

Недоліки реалізації:

1. Комплексність коду: Завдяки підтримці різних методів голосування код може здатися дещо заплутаним для новачків у сфері блокчейну.

2. Вартість газу: Залежно від обсягу голосувань та кількості учасників, вартість газу може бути високою.

3. Потреба в додатковому тестуванні: З огляду на численні функції та методи, код потребує глибокого тестування для гарантії його безпечної роботи.

4.

5.4 Рекомендації щодо доопрацювання

На основі аналізу смартконтракту та враховуючи загальні принципи роботи з блокчейн-технологією, можна рекомендувати наступні заходи щодо доопрацювання.

5.4.1 Пропозиції щодо оптимізації смартконтракту.

1. Оптимізація використання газу: Враховуючи підвищену вартість газу в Ethereum, рекомендується розглянути можливості для зменшення вартості транзакцій, наприклад, використовувати менше змінних стану або оптимізувати логіку виконання коду.

2. Оновлення безпеки: Рекомендується регулярно проводити аудит коду з метою виявлення потенційних дірок в безпеці, особливо після будь-яких змін або доповнень до коду.

3. Введення додаткових функцій: Розглянути можливість додавання нових функцій, які можуть збільшити функціональність та корисність вашого смартконтракту. Наприклад, можна додати функціонал для автоматичного виводу результатів голосування.

4. Інтеграція з іншими платформами: З метою розширення можливостей смартконтракту, можна розглянути його інтеграцію з іншими блокчейн-платформами або сервісами.
5. Events: Використовувати події для логування важливої інформації.
6. Mappings: Оптимально використовувати відображення для зберігання та отримання даних.
7. Оптимізація: Переглянути код з метою оптимізації використання газу та вдосконалення логіки.
8. Вдосконалення коментування коду: Забезпечити, щоб усі функції та змінні були належно задокументовані.
9. Юніт-тестування: Розробити юніт-тести для перевірки коректності роботи всіх функцій смартконтракту. Переконатися, що контракт вільний від відомих вразливостей, таких як reentrancy, overflow/underflow.
10. Використовувати практики відомих бібліотек.
11. Модифікатори: Продовжувати використання модифікаторів для перевірки умов. Розглянути можливість додавання нових модифікаторів для додаткових перевірок.
12. Код-рев'ю: Рекомендується провести код-рев'ю смартконтракту разом з колегами або експертами в галузі Solidity для отримання зовнішнього погляду та рекомендацій щодо покращення.

5.5 Взаємодія з іншими технологіями

5.5.1 Інтеграція з іншими системами або платформами.

1. Веб-додатки та мобільні додатки: Смартконтракти можуть бути інтегровані з традиційними веб та мобільними додатками за допомогою таких інструментів як Web3.js або ethers.js. Це дозволяє

користувачам взаємодіяти з контрактом безпосередньо з їх браузерів чи мобільних пристроїв.

2. Системи ідентифікації: Інтеграція з децентралізованими системами ідентифікації може допомогти в автоматичній верифікації осіб, які беруть участь у голосуванні, забезпечуючи додатковий рівень безпеки.
3. Модулі аналітики: Інтегруючи контракт з модулями аналітики, можна отримати детальний аналіз результатів голосування, виявляти тенденції та забезпечувати прозорість процесу.
4. Інтеграція смартконтрактів з іншими технологіями підвищує їхню універсальність, забезпечує гнучкість і розширює можливості їх застосування в різних сферах.

5.6 Висновок розділу

В цьому розділі проведено глибокий аналіз та узагальнення результатів дослідження смартконтракту для голосування, розробленого на мові Solidity.

5.6.1 Головні висновки розділу.

Вивчення та Аналіз: детально вивчено і проаналізовано основні методи голосування, які були розглянуті у розділі 3. Це допомогло краще зрозуміти їх переваги, недоліки та особливості використання в контексті блокчейну та смартконтрактів.

Переваги та Недоліки. Визначено ключові переваги та потенційні обмеження смартконтракту. Це дасть можливість для подальшого вдосконалення та адаптації під конкретні задачі та вимоги.

Рекомендації: На основі аналізу надано конкретні рекомендації щодо доопрацювання смартконтракту, які можуть поліпшити його продуктивність, безпеку та інші аспекти.

Інтеграція з Іншими Технологіями. Розглянуто потенційні можливості інтеграції смартконтракту з іншими системами і платформами, що може розширити його застосування та функціональність.

Подальший Розвиток: З моїх досліджень стає зрозуміло, що смартконтракт є потужним інструментом для голосування, але також існують можливості для його оптимізації та розширення функціональності.

У підсумку смартконтракт має великий потенціал у сфері голосувань на базі блокчейна. З врахуванням рекомендацій та подальшого вдосконалення, він може стати лідером у цій області.

ВИСНОВКИ

Загальний огляд дослідження

Моя робота була спрямована на глибоке вивчення та аналіз систем голосування на базі блокчейну, зосереджуючись на розробці ефективного смартконтракту на Solidity. Були досліджені різні методи голосування, їх переваги та недоліки, а також можливості їх оптимізації та інтеграції з іншими технологіями.

Обговорення основних моментів кожного розділу.

Розділ 1: Провів аналіз літератури та наявних технологічних рішень у сфері блокчейну і смартконтрактів. Був зроблений акцент на дослідженні історичного розвитку, ключових обмеженнях та можливостей цих технологій.

Розділ 2: Дослідження сучасного стану проблеми, аналіз традиційних та електронних методів голосування. Вивчено їх еволюцію, переваги та недоліки.

Розділ 3: Вивчення та аналіз різних методів підрахунку голосів, їх застосування у студентських голосуваннях, а також переваги та недоліки кожного методу.

Розділ 4: Розробка та тестування смартконтракту на Solidity. Проаналізовано вимоги, реалізовано основні функції, проведено тестування та оцінку продуктивності та безпеки.

Розділ 5: Узагальнення та аналіз результатів дослідження. Надано рекомендації щодо доопрацювання та інтеграції смартконтракту з іншими технологіями.

Основні досягнення

Результати дослідження.

1. Аналіз наявних методів голосування.

У процесі дослідження було вивчено ряд традиційних та сучасних методів голосування. Кожен з них має свої переваги та недоліки. Це дало можливість краще розуміти контекст та виклики, які стоять перед системами електронного голосування.

2. Вивчення технології блокчейн.

Технологія блокчейн відіграє ключову роль у сучасних системах електронного голосування. Ознайомлення з основними принципами роботи блокчейну та розумних контрактів допомогло визначити можливості та обмеження даної технології.

3. Розробка концепції системи.

На основі попереднього аналізу було створено теоретичну модель системи електронного голосування на базі блокчейну. Ця модель відображає основні компоненти системи та їх взаємодію.

4. Програмування розумних контрактів.

Реалізація розумних контрактів на мові Solidity стала ключовою частиною роботи. Даний контракт забезпечує основну функціональність системи голосування, а також високий рівень безпеки та прозорості.

5. Оцінка прозорості.

Аналіз механізмів, які забезпечують прозорість в розробленій системі, показав, що використання блокчейн-технології значно підвищує довіру до системи голосування. Кожен голос зберігається в блокчейні без можливості його зміни або видалення, що гарантує чесність голосування.

Загалом, розроблена система демонструє потенціал блокчейн-технології у сфері електронного голосування. Вона може стати основою для створення нових, більш надійних та ефективних систем, які відповідають сучасним вимогам та викликам.

Прикінцеві зауваження

У процесі виконання цієї кваліфікаційної роботи було досліджено можливості застосування блокчейн-технологій у системах електронного голосування в університетському середовищі. Розроблений підхід, що базується на використанні розумних контрактів мовою Solidity, відкриває нові горизонти для створення прозорих, безпечних та ефективних систем голосування.

Значущість і вплив дослідження.

Значущість дослідження полягає в комбінації теоретичних знань і практичних навичок у сфері блокчейн-технологій та їх застосуванні в освітньому процесі. У контексті постійного розвитку інформаційних технологій та цифровізації освітніх процесів, дана робота акцентує увагу на важливості пошуку нових, більш ефективних та надійних рішень для організації голосувань.

Вплив дослідження на практичний сектор очевидний. Завдяки розробленому підходу, університети та інші освітні заклади можуть впроваджувати системи голосування, які не тільки забезпечують високий рівень прозорості та безпеки, але й значно спрощують сам процес голосування для студентів. Це, в свою чергу, може сприяти підвищенню активності студентської спільноти та їхньої участі в процесах прийняття рішень на різних рівнях університетського життя.

Таким чином, результати цього дослідження мають не тільки академічне, але й практичне значення, відкриваючи нові перспективи для впровадження інноваційних технологій в освітньому середовищі.

У дипломній роботі були присутні ролі:

- розробника,
- архітектора,
- тестувальника.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gupta, S., & Manjunath, C. R. (Рік). Blockchain-based Preferential E-Voting System DApp using Smart Contract. Jain University, Bengaluru, India. https://ssrn.com/abstract=3835096
2. Buterin, V. et al. (2014). A next-generation smart contract and decentralized application platform. White Paper.
3. Nakamoto, Satoshi. (2009). Bitcoin: A Peer-to-Peer Electronic Cash System.
4. Yavuz, E.A., Koç, A., Çabuk, U.C., & Dalkiliç, G. (2018). Towards secure e-voting using ethereum blockchain. 2018 6th International Symposium on Digital Forensic and Security (ISDFS), 1-7.

5. Þ. Hjálmarsson, F., K. Hreiðarsson, G., Hamdaqa, M., & Hjálmtýsson, G. (2018). Blockchain-Based E-Voting System. 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, pp. 983-986, doi: 10.1109/CLOUD.2018.00151.
6. Невідомий. (Дата публікації: 16 червня 2023). "Основи Layer 2 та їх вплив на Ethereum". Доступно на: https://teletype.in/@coin_research/layer2 (дата звернення: 10.10.2023).
7. Алькараллех, Б. А. Й., Вайяпурі, Т., Парваті, В. С. та ін. Blockchain-assisted secure image transmission and diagnosis model on Internet of Medical Things Environment. Pers Ubiquit Comput (2021). [Online]. Доступно на: https://doi.org/10.1007/s00779-021-01543-2
8. Г'яльмарссон, Ф. Є., Грейдарссон, Г. К., Хамдака, М. та Х'яльмтіссон, Г. "Blockchain-Based E-Voting System," 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), Сан-Франциско, CA, 2018, стор. 983-986, doi: 10.1109/CLOUD.2018.00151.
9. "Blockchain Tutorial for Beginners: Learn Blockchain Technology," [Online]. Доступно на: https://www.guru99.com/blockchain-tutorial.html
10. "INTRODUCTION TO DAPPS," [Online]. Доступно на: https://ethereum.org/en/developers/docs/dapps/
11. "INTRODUCTION TO SMART CONTRACTS," [Online]. Доступно на: https://ethereum.org/en/developers/docs/smart-contracts/
12. "INTRO TO ETHEREUM," [Online]. Доступно на: https://ethereum.org/en/developers/docs/intro-to-ethereum/
13. Каури, Д., Кфури, Е. Ф., Касем, А. та Харб, Х. "Decentralized Voting Platform Based on Ethereum Blockchain," 2018 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET), Бейрут, 2018, стор. 1-6, doi: 10.1109/IMCET.2018.8603050
14. Моханті, С. Н., Рамя, К. Ц., Рані, С. С. та ін. An efficient Lightweight integrated Blockchain (ELIB) model for IoT security and

privacy. Future Generation Computer Systems, 102, 2020, стор. 1027-1037.

15. Тарасов, П. та Теварі, Х. "The future of e-voting." IADIS International Journal on Computer Science & Information Systems, т. 12, № 2, 2017.
16. Таллок, Гордон. "Problems of Majority Voting." Journal of Political Economy, т. 67, № 6, 1959, стор. 571–579. JSTOR, [Online]. Доступно на: www.jstor.org/stable/1827311
17. Тхуї, Л. В., Као-Мінь, К., Данг-Ле-Бао, Ч. та Нгуєн, Т. А. "Votereum: An Ethereum-Based E-Voting System," 2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF), Дананг, В'єтнам, 2019, стор. 1-6, doi: 10.1109/RIVF.2019.8713661
18. Шукла, С., Тасмія, А. Н., Шашанк, Д. О. та Мамата, Х. Р. "Online Voting Application Using Ethereum Blockchain," 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Бангалор, 2018, стор. 873-880, doi: 10.1109/ICACCI.2018.8554652
19. Явуз, Е. А., Коч, А., Чабук, У. Ч. та Далкіліч, Г. (2018). Towards secure e-voting using ethereum blockchain. 2018 6th International Symposium on Digital Forensic and Security (ISDFS), стор. 1-7.
20. "Why Your Business Wants To Adopt Blockchain Technology," [Online]. Доступно на: https://senlainc.com/blog/why-your-business-wants-to-adopt-blockchain-technology/
21. Z. Zheng, S. Xie, H. Dai, X. Chen, і H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," Processing IEEE International Congress on Big Data, стор. 556–570, 2017.
22. S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Journal for General Philosophy of Science, стор. 1–9, 2018.

23. F. S. Hardwick, Apostolos Gioulis, Raja Naeem Akram, "E-Voting with Blockchain: An E-Voting Protocol with Decentralisation and Voter Privacy," University of London, Egham, United Kingdom, 2018.
24. A. Ben Ayed, "A Conceptual Secure Blockchain-Based Electronic Voting System," International Journal of Network Security and Its Application., том. 9, № 3, стор. 1-11, 2017.
25. K. Christidis і M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," IEEE Access, том. 4, стор. 2289–2312, 2016.
26. F. Fusco, M. I. Lunesu, F. E. Pani, і A. Pinna, "Crypto-voting, a Blockchain-based e-Voting System," Piazza dArmi, Cagliari, стор. 218–231, 2018.
27. A. Navya, R. Roopini, S. N. A. S, і B. Prabhu, "Electronic voting machine based on Blockchain technology and Aadhar verification," International Journal of Advanced Research. Ideas Innovations Technology., том. 4, № 2, стор. 1169–1190, 2018.
28. F. P. Hjalmarsson, G. K. Hreioarsson, M. Hamdaqa, і G. Hjalmtysson, "Blockchain-Based E-Voting System," IEEE International Conference on Cloud Computing Cloud, стор. 979–991, 2018.
29. U. Can Çabuk, E. AdÕgüzel, and E. Karaarslan, “A Survey on Feasibility and Suitability of Blockchain Techniques for the E-Voting Systems,” International Journal Advanced Research Computer and Communication Engineering, vol. 7, no. 3, pp. 123–135, 2018.
30. A. M. Oo, Htet Ne; Aung, “A Survey of Different Electronic Voting Systems,” International Journal of Scientific and Engineering Research, vol. 03, no. 16, pp. 3458–3466, 2014.
31. S. Based and E. Smart, “Towards Analyzing the Complexity Landscape of Solidity Based Ethereum Smart Contracts,” Research Group on Artificial Intelligence, pp. 1-20, 2019.

32. K. Yahaya, “Implementation of Mobile Voting Application in Infrastructure University Kuala Lumpur, Malaysia,” International Journal Computer Applications, vol. 180, no. 47, pp. 22–34, 2018.
33. S. Wu, “Evaluation and Improvement of Two Blockchain-Based Evoting System : Agora and Proof of Vote,” Master Thesis. University Of Birmingham 2018.
34. S. Shah, Q. Kanchwala, and H. Mi, “BlockChain Voting System,” Dissertation, Northeastern University, 2017.
35. Geoffrey Cann and Emily Catmur, “Blockchain : Overview of the potential applications for the oil and gas market and the related taxation implications,” 2017, Deloitte Touche Tohmatsu Limited.
36. D. Springall, Travis Finkenauer, and Zakir Durumeric, “Security Analysis of the Estonian Internet Voting System,” Security, pp. 699–716, 2014.
37. S. Ibrahim, M. Salleh, and M. Kamat “Electronic Voting System : Preliminary Study,” University Teknologi Malaysia, pp. 2-13, 2000.
38. A. Barnes, C. Brake, and T. Perry, “Digital Voting with the use of Blockchain Technology,” Team Plymouth Pioneers, Plymouth University, 2017.
39. Y. Liu and Q. Wang, “An E-voting Protocol Based on Blockchain,” Southern University of Science and Technology, 2018.
40. H. Zhu and Z. Z. Zhou, “Analysis and outlook of applications of blockchain technology to equity crowdfunding in China,” Financial Innovation vol. 2, no. 1, pp. 30-42, 2016.
41. A. K. Koç, E. Yavuz, U. C. Çabuk, and G. Dalkiliç, “Towards secure e-voting using ethereum blockchain,” 6th International Symposium on Digit Forensic Security ISDFS 2018 Proceeding, pp. 2–8, 2018.

42. A. Pathak and A. Wasay, “Design and implementation of a secure and robust voting system based on blockchain,” International Journal of Advance Research, Ideas and Innovations in Technology, vol. 4, no. 5, pp. 868–876, 2018.
43. N. Soledad and M. Meza, “A Model for Direct Recording Electronic Voting Systems,” Master Thesis, University of Pretoria, 2008.
44. W. L. Shen “Secure Student Representative Council Voting System Using,” Dissertation, Universiti Teknikal Malaysia Melaka, 2014.
45. M. Hapsara, A. Imran, T. L. Turner, A. Defence, and F. Academy, “EVoting in Developing Countries E-Voting in Developing Countries Current Landscape and Future Research Agenda,” Second International Joint Conference on Electronic Voting, Lecture Notes Computer Science, vol. 10141, 2016.
46. Solidity by Example [Електронний ресурс]. - Режим доступу: https://docs.soliditylang.org/en/v0.8.21/solidity-by-example.html. - Дата звернення: 15 жовтня 2023.
47. Рудь Віктор. (2023). Структури даних у Solidity: порівняльний аналіз масивів та mappings. "Тенденції та перспективи розвитку науки і освіти в умовах глобалізації". [В друці].

Додаток А

Система контрактів Ethereum для різних видів голосувань

А.1 Контракт для управління списком голів

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.21;

//# Owners
// Контракт для управління списком голів
contract OwnerBlock {
// Масив адрес голів
```

```
address[] public owners;

// TODO mapping
// Constructor
// Конструктор для ініціалізації голів
constructor(address[] memory _owners) {
 bool invalid = false;

 // Якщо список голів при ініціалізації порожній, встановлюємо відправника повідомлення (розгортальника контракту) як єдиного голову
 if(_owners.length == 0) {
owners.push(msg.sender);
return;
 }
 if(_owners.length == 1) {
require(_owners[0] != address(0), "Invalid owner address");
owners.push(_owners[0]);
return;
 }

 for(uint i = 0; i < _owners.length; i++) {

if(_owners[i] == address(0)) continue;

for(uint j = 0; j < owners.length; j++) {
if(owners[j] == _owners[i]) {
 invalid = true;
 break;
}
}

 if(invalid == true) continue;
 owners.push(_owners[i]);
 }
}

// Modifiers
// Модифікатор: перевіряє, чи є відправник головою
modifier onlyOwners() {
 require(oIsOwner(msg.sender) == true, "msg.sender is not an owner");
 _;
}

// Модифікатор: перевіряє, чи є адреса нульовою
modifier notZeroAddress(address _addr) {
 require(_addr != address(0), "address(0)");
 _;
}

// Модифікатор: перевіряє, чи належить адреса списку голів
modifier isOwner(address _addr) {
 require(oIsOwner(_addr) == true, "Input address is not an owner");
 _;
}

// Модифікатор: перевіряє, чи не належить адреса списку голів
modifier isNotOwner(address _addr) {
 require(oIsOwner(_addr) == false, "Input address is an owner");
 _;
}

// Functions
// Show
```

```
// Повертає список всіх голів
function oGetOwnerList() public view returns (address[] memory) {
 return owners;
}

// Check
// TODO mapping
// Перевіряє, чи є вказана адреса головою
function oIsOwner(address _addr) public view returns(bool) {
 for (uint i = 0; i < owners.length; i++) {
if (owners[i] == _addr) {
return true;
}
 }
 return false;
}

// TODO mapping
// Перевіряє, чи є всі вказані адреси головами
function oIsOwners(address[] memory _ownersToCheck) public view returns(bool) {
 for (uint i = 0; i < _ownersToCheck.length; i++) {
if (!oIsOwner(_ownersToCheck[i])) {
return false;
}
 }
 return true;
}

// Adds
// Додає нову адресу до списку голів
function oAddOwner(address _addr) public onlyOwners notZeroAddress(_addr) isNotOwner(_addr) {
 owners.push(_addr);
}

// Додає декілька нових адрес до списку голів
function oAddOwners(address[] memory _newOwners) public onlyOwners {
 require(_newOwners.length != 0, "_newOwners.length == 0");
 for (uint i = 0; i < _newOwners.length; i++) {
require(_newOwners[i] != address(0), "_newOwners[i] == address(0)");
require(!oIsOwner(_newOwners[i]), "_newOwners[i] is already an owner");
owners.push(_newOwners[i]);
 }
}

// Removals
// Видаляє вказану адресу зі списку голів
function oRemoveOwner(address _addr) public onlyOwners isOwner(_addr) notZeroAddress(_addr) {
 for (uint i = 0; i < owners.length; i++) {
if (owners[i] == _addr) {
owners[i] = owners[owners.length - 1];
owners.pop();
return;
}
 }
}

// Видаляє декілька адрес зі списку голів
function oRemoveOwners(address[] memory _addr) public onlyOwners {
 for (uint i = 0; i < _addr.length; i++) {
for (uint j = 0; j < owners.length - 1; j++) {
if(owners[j] == _addr[i]) {
 owners[j] = owners[owners.length-1];
 owners.pop();
```

```
 // Якщо залишився лише один голова, зупиняємо цикл
 if(owners.length == 1) break;
}
}
 }
}

// Видаляє всіх голів, крім відправника повідомлення
function oRemoveAllOwners() public onlyOwners {
 owners = new address[](1);
 owners[0] = msg.sender;
}

// Видаляє всіх голів, крім вказаної адреси
function oRemoveAllOwnersExcept(address _addr) public onlyOwners notZeroAddress(_addr)
isOwner(_addr) {
 owners = new address[](1);
 owners[0] = _addr;
}
}
```

A.2 Контракт, що надає функціональність для управління раундами, пропозиціями та вагами голосів

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.21;
import "./OwnersV2.sol";

//# RoundsProposalsWeights
// Контракт, що надає функціональність для управління раундами, пропозиціями та вагами голосів
abstract contract RoundProposaWeightBlock is OwnerBlock {
// structs
// Структура, що описує пропозицію
struct Proposal {
 string name; // Назва пропозиції
 uint voteCounter; // Кількість голосів, що підтримують пропозицію
}

// Структура, що описує вагу голосу
struct Weight {
 address voterAddress; // Адреса виборця
 uint weight; // Вага голосу
}

// Структура, що описує раунд голосування
struct Round {
 Proposal[] proposals;// Список пропозицій у цьому раунді
 Weight[] weights;// Список ваг у цьому раунді
 bool timeStart;// Статус раунду (почався чи ні)
 uint timeFinish; // Час завершення раунду
}

// State Variables
// Змінна для зберігання всіх раундів голосування
Round[] public rounds;

// Modifiers
// Модифікатор для перевірки валідності індексу раунду
```

```
modifier roundValidIndex(uint _roundIndex) {
 require(_roundIndex < rounds.length, "Index out of bounds");
 _;
}
}
```

А.3 Контракт, що надає функціональність для управління пропозиціями

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.21;
import "./OwnersV2.sol";
import "./RoundsProposalsWeights.sol";

//# Proposals
// Контракт, що надає функціональність для управління пропозиціями
abstract contract ProposalBlock is OwnerBlock, RoundProposaWeightBlock {
// Modifiers
// Модифікатор для перевірки валідності індексу пропозиції
modifier proposalValidIndex(uint _roundIndex, uint _proposalIndex) {
 require(_proposalIndex < rounds[_roundIndex].proposals.length, "Proposal index out of bounds");
 _;
}

// Functions
// Viev
// Функція для перегляду всіх пропозицій у вказаному раунді
function pfViewAllProposals(uint _roundIndex) public roundValidIndex(_roundIndex) view returns
(Proposal[] memory) {
 return rounds[_roundIndex].proposals;
}

// Adds
// Функція для додавання одного варіанту для голосування у вказаний раунд
function proposalsAddSingle(uint _roundIndex, string memory _proposalName) public onlyOwners
roundValidIndex(_roundIndex) {
 rounds[_roundIndex].proposals.push(Proposal(_proposalName, 0));
}

// Функція для додавання декількох варіантів для голосування в вказаний раунд
function pfAddSomeProposals(uint _roundIndex, string[] memory _proposalNames) public onlyOwners
roundValidIndex(_roundIndex) {
 Proposal[] storage targetProposals = rounds[_roundIndex].proposals;
 for (uint i = 0; i < _proposalNames.length; i++) {
targetProposals.push(Proposal(_proposalNames[i], 0));
 }
}

// Removes
// Функція для видалення варіанту голосування в вказаному раунді
function pfRemovesOneProposal(uint _roundIndex, uint _proposalIndex) public onlyOwners
proposalValidIndex(_roundIndex, _proposalIndex) {
 Proposal[] storage targetProposals = rounds[_roundIndex].proposals;
 targetProposals[_proposalIndex] = targetProposals[targetProposals.length - 1];
 targetProposals.pop();
}
```

```
// Функція для видалення варіанту голосування у вказаному раунді зі збереженням індексів варіантів голосування
function pfRemovesOneProposalKeepIndex(uint _roundIndex, uint _proposalIndex) public onlyOwners proposalValidIndex(_roundIndex, _proposalIndex) { // ! proposalsMinusSingleKeepIndex
 delete rounds[_roundIndex].proposals[_proposalIndex];
}

// Функція для очищення всіх варіантів голосування в вказаному раунді
function proposalsMinusAll(uint _roundIndex) public onlyOwners roundValidIndex(_roundIndex) { // ! pfRemovesAllProposals
 require(rounds[_roundIndex].proposals.length > 0, "No proposals to clear.");
 delete rounds[_roundIndex].proposals;
}

// Changes
// Функція для заміни варіанту голосування у вказаному раунді
function pfChange(uint _roundIndex, uint _proposalIndex, string memory _newName) public onlyOwners proposalValidIndex(_roundIndex, _proposalIndex) { // ! proposalsChange
 rounds[_roundIndex].proposals[_proposalIndex].name = _newName;
}
}
```

A.4 Контракт для управління вагами

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.21;
import "./OwnersV2.sol";
import "./RoundsProposalsWeights.sol";

//# Weighs
abstract contract WeightBlock is OwnerBlock, RoundProposaWeightBlock {
// Modifiers
modifier addressNotInWeights(uint _roundIndex, address _voter) {
 bool exists = false;
 for (uint i = 0; i < rounds[_roundIndex].weights.length; i++) {
if(rounds[_roundIndex].weights[i].voterAddress == _voter) {
exists = true;
break;
}
 }
 require(!exists, "Address already exists in weights");
 _;
}



// Functions
// View
// Функція для перегляду ваги виборця по адресу в вказаному раунді
function wViewOneAddrWeight(uint _roundIndex, address _voter) public view roundValidIndex(_roundIndex) returns (uint) {
 for (uint i = 0; i < rounds[_roundIndex].weights.length; i++) {
if (rounds[_roundIndex].weights[i].voterAddress == _voter) {
return rounds[_roundIndex].weights[i].weight;
}
 }
 return 0;
}
```

```
// Функція для перегляду всіх ваг у вказаному раунді
function wViewAllAddrWeightweights(uint _roundIndex) public view roundValidIndex(_roundIndex)
returns (Weight[] memory) {
 return rounds[_roundIndex].weights;
}

// Adds
// Функція для додавання ваги виборцю у вказаному раунді
function wAddAddressWeightNew (uint _roundIndex, address _voter, uint _Weight) public onlyOwners
roundValidIndex(_roundIndex) {
 Weight[] storage weights = rounds[_roundIndex].weights;
 bool addressFound = false;
 for (uint i = 0; i < weights.length; i++) {
if (weights[i].voterAddress == _voter) {
if (_Weight >= 1) {
 weights[i].weight = _Weight;
} else {
 // Видаляємо виборця з вагою менше 1
 weights[i] = weights[weights.length - 1];
weights.pop();
}
addressFound = true;
break; // Вихід із циклу після оновлення або видалення
}
 }
 if (!addressFound && _Weight >= 1) {
weights.push(Weight(_voter, _Weight));
 }
}

// Функція для додавання декількох ваг виборців по адресах у вказаний раунд
function wAddAddressWeights(uint _roundIndex, address[] memory _addresses, uint[] memory _weights)
public onlyOwners roundValidIndex(_roundIndex) {
 require(_addresses.length == _weights.length, "Arrays must have the same length");
 Weight[] storage weights = rounds[_roundIndex].weights;
 for (uint i = 0; i < _addresses.length; i++) {
bool addressFound = false;
for (uint j = 0; j < weights.length; j++) {
if (weights[j].voterAddress == _addresses[i]) {
 if (_weights[i] >= 1) {
weights[j].weight = _weights[i];
 } else {
weights[j] = weights[weights.length - 1];
weights.pop();
 }
 addressFound = true;
 break;
}
}
if (!addressFound && _weights[i] >= 1) {
weights.push(Weight(_addresses[i], _weights[i]));
}
 }
}

// Removes
// Функція для видалення ваги виборця по адресу у вказаному раунді
function wRevoveOneAddressWeight(uint _roundIndex, address _voter) public onlyOwners
roundValidIndex(_roundIndex) {
 Weight[] storage weights = rounds[_roundIndex].weights;
 for (uint i = 0; i < weights.length; i++) {
if(weights[i].voterAddress == _voter) {
weights[i] = weights[weights.length - 1];
```

```
weights.pop();
return;
}
 }
}

// Функція для очищення всіх ваг у вказаному раунді
function wRevoveAllAddressWeights(uint _roundIndex) public onlyOwners roundValidIndex(_roundIndex)
{
 delete rounds[_roundIndex].weights;
}
}
```

А.5 Контракт для делегування

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.21;
import "./OwnersV2.sol";
import "./RoundsProposalsWeights.sol";
import "./Weights.sol";

//# WeightsDelegate
abstract contract DelegateWeightBlock is OwnerBlock, RoundProposaWeightBlock, WeightBlock {
// Functions
/
 @dev Делегує вказану вагу від відправника до одержувача в вказаному раунді
 @param _roundIndex Індекс раунду, в якому відбувається делегування
 @param _to Адреса одержувача
 @param _weight Вага, яку потрібно делегувати
/
function wDelegateWeight(uint _roundIndex, address _to, uint _weight) public
roundValidIndex(_roundIndex) notZeroAddress(_to) {
 require(_to != msg.sender, "Cannot delegate to yourself"); // Неможливо делегувати самому собі
 require(_weight > 0, "Weight to delegate should be greater than zero"); // Вага, яку потрібно делегувати,
повинна бути більше нуля

 Weight[] storage weights = rounds[_roundIndex].weights;

 uint senderWeight = 0;
 bool receiverFound = false;

 // Пошук відправника та одержувача в одному циклі
 for (uint i = 0; i < weights.length; i++) {
// Зменшення ваги відправника
if (weights[i].voterAddress == msg.sender) {
senderWeight = weights[i].weight;
require(senderWeight >= _weight, "Not enough weight to delegate");
weights[i].weight -= _weight;
if (weights[i].weight == 0) {
 // Видалення відправника, якщо його вага дорівнює нулю
 weights[i] = weights[weights.length - 1];
 weights.pop();
}
}

// Збільшення ваги одержувача
if (weights[i].voterAddress == _to) {
weights[i].weight += _weight;
receiverFound = true;
```

```
}
 }

 // Перевірка на наявність достатнього ваги у відправника
 require(senderWeight >= _weight, "Sender not found or not enough weight");

 // Якщо одержувача не знайдено, додайте його з новою вагою
 if (!receiverFound) {
weights.push(Weight(_to, _weight));
 }
}
}
```

A.6 Контракт, що надає функціональність для управління тривалістю раундів

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.21;
import "./OwnersV2.sol";
import "./RoundsProposalsWeights.sol";
import "./Weights.sol";

//# Times
/
@title TimeBlock
@dev Контракт для керуванням часом голосування
 /
abstract contract TimeBlock is OwnerBlock, RoundProposaWeightBlock {
// Modifiers
// Перевірка чи стартувало голосування
modifier tmVoteHasStarted(uint _roundIndex) {
 require(rounds[_roundIndex].timeStart, "Voting has not started");
 _;
}

// Перевірка чи не стартувало голосування
modifier tmVoteHasNotStarted(uint _roundIndex) {
 require(!rounds[_roundIndex].timeStart, "Voting has already started");
 _;
}

// Functions
/
 @dev Запускає голосування для вказаного раунду
 @param _roundIndex Індекс раунду
 @param _votingDurationInSeconds Тривалість голосування в секундах
/
function tfStartVoting(uint _roundIndex, uint _votingDurationInSeconds) public onlyOwners
roundValidIndex(_roundIndex) {
 require(block.timestamp > rounds[_roundIndex].timeFinish, "A voting session is already in progress");
 require(!rounds[_roundIndex].timeStart, "Voting has already started");

 rounds[_roundIndex].timeStart = true;
 rounds[_roundIndex].timeFinish = block.timestamp + _votingDurationInSeconds;
}

/
 @dev Ручне завершення голосування для вказаного раунду
```

```
 @param _roundIndex Індекс раунду
/
function tfFinisVotinghManually(uint _roundIndex) public onlyOwners roundValidIndex(_roundIndex) {
 require(rounds[_roundIndex].timeStart == true, "Voting has not started yet");

 // Скидання змінних стану для цього раунду
 rounds[_roundIndex].timeStart = false;
 rounds[_roundIndex].timeFinish = 0;
}
}
```

А.7 Контракт для управління раундами голосування

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.21;
import "./OwnersV2.sol";
import "./RoundsProposalsWeights.sol";
import "./Proposals.sol";
import "./Weights.sol";

//# Round
/
@title RoundBlock
@dev Контракт для управління раундами голосування
 /
abstract contract RoundBlock is OwnerBlock, RoundProposaWeightBlock {
// Functions
/
 @notice Створює новий раунд голосування
 @dev Створює новий раунд з заданими предложеніями та вагами голосів для кожного виборця
 @param _proposalNames Масив назв предложень
 @param _addresses Масив адресів виборців
 @param _weights Масив ваг для виборців
/
function rfCreateNewRouund(string[] memory _proposalNames, address[] memory _addresses, uint[]
memory _weights) public onlyOwners {
 require(_addresses.length > 0, "No voters to add");
 require(_addresses.length == _weights.length, "Arrays must have the same length");
 require(_proposalNames.length > 1, "At least two proposals are required");

 // Ініціалізація нового раунду голосування
 rounds.push();
 uint newRoundIndex = rounds.length - 1;
 rounds[newRoundIndex].timeStart = false;
 rounds[newRoundIndex].timeFinish = 0;

 // Додавання пропозицій до раунду
 for (uint i = 0; i < _proposalNames.length; i++) {
rounds[newRoundIndex].proposals.push(Proposal(_proposalNames[i], 0));
 }

 // Встановлення ваг для виборців
 for (uint i = 0; i < _addresses.length; i++) {
rounds[newRoundIndex].weights.push(Weight(_addresses[i], _weights[i]));
 }
}
}
```

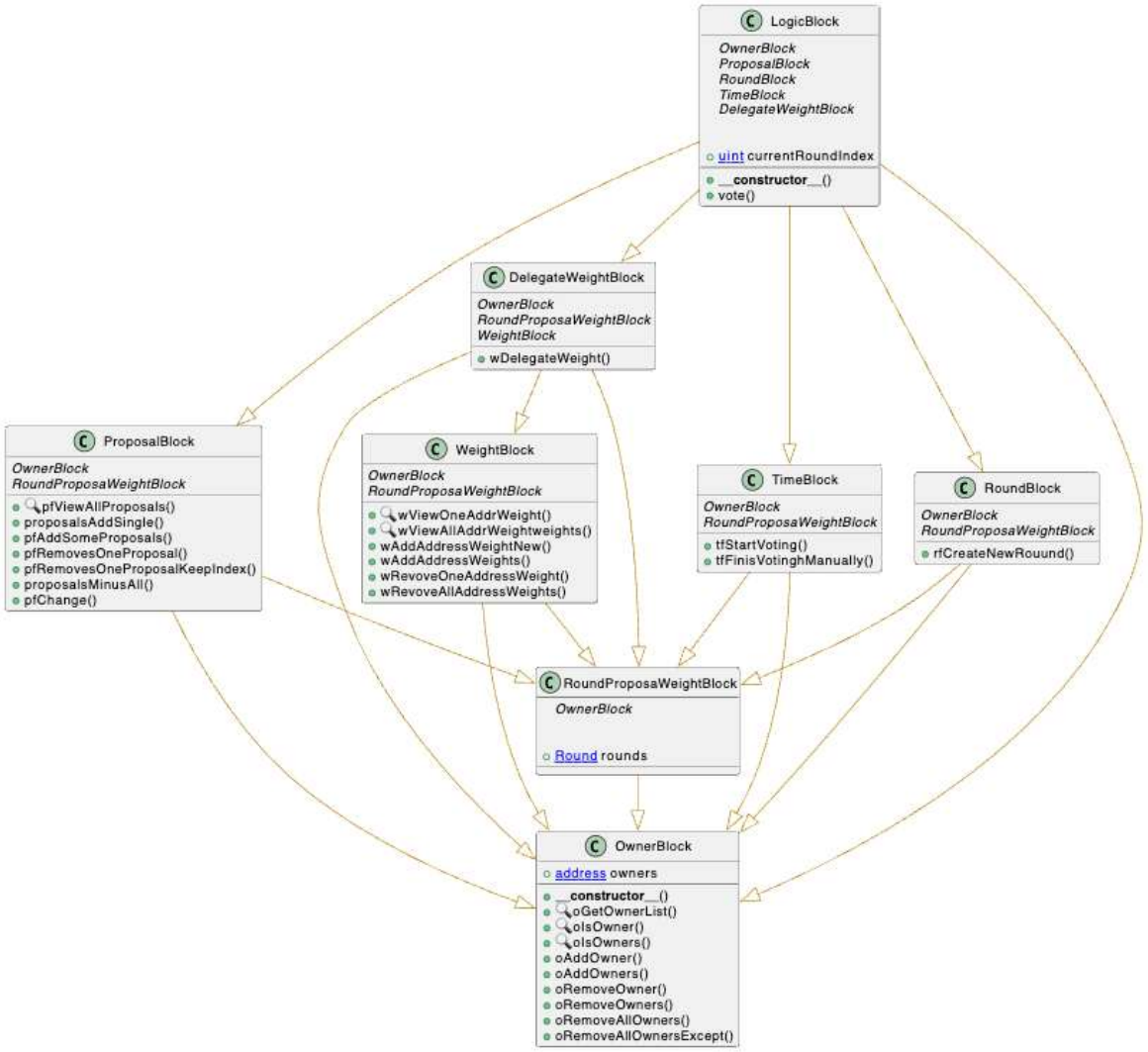
А.8 Основний контракт для логіки голосування

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.21;
import "./OwnersV2.sol";
import "./RoundsProposalsWeights.sol";
import "./Proposals.sol";
import "./Weights.sol";
import "./RoundsV2.sol";
import "./TimesV2.sol";
import "./DelegateWeightsV2.sol";

//# Logics
/
@title LogicBlock
@dev Основний контракт для логіки голосування
 /
contract LogicBlock is OwnerBlock, ProposalBlock, RoundBlock, TimeBlock, DelegateWeightBlock {
uint public currentRoundIndex = 0;// Індекс поточного активного раунду

/
 @dev Конструктор для ініціалізації власників
 @param _owners Масив адрес власників
/
constructor(address[] memory _owners) OwnerBlock(_owners) {}

/
 @notice Функція голосування
 @dev Виборець може проголосувати лише один раз
 @param _roundIndex Індекс раунду
 @param _proposalIndex Індекс пропозиції
 @param _weightToUse Вага для голосування
/
function vote(uint _roundIndex, uint _proposalIndex, uint _weightToUse) public
proposalValidIndex(_roundIndex, _proposalIndex) tmVoteHasStarted(_roundIndex) {
 require(rounds[currentRoundIndex].timeFinish == 0 || block.timestamp <
rounds[currentRoundIndex].timeFinish, "Voting period has ended");

 uint voterIndex;
 bool found = false;

 Proposal[] storage targetProposals = rounds[currentRoundIndex].proposals;
 Weight[] storage targetWeights = rounds[currentRoundIndex].weights;

 // Пошук виборця в масиві
 for(uint i = 0; i < targetWeights.length; i++) {
if(targetWeights[i].voterAddress == msg.sender) {
voterIndex = i;
found = true;
break;
}
 }

 require(found, "Voter not found");
 require(targetWeights[voterIndex].weight >= _weightToUse, "You do not have enough weight to vote");
 require(_weightToUse > 0, "Weight to use should be greater than zero");
 require(bytes(targetProposals[_proposalIndex].name).length != 0, "Cannot vote for a deleted proposal");

 targetProposals[_proposalIndex].voteCounter += _weightToUse;
 targetWeights[voterIndex].weight -= _weightToUse;
```

```
}
}
```

Додаток Б

Детальна схема залежностей дочірніх контрактів від батьків



Додаток В

Структурний аналіз смартконтракту

Таблиця В.1 – Структурний аналіз за допомогою інструменту Sūrya

Контракт	Тип	Бази		
└	Назва функції	Видимість	Mutability	Модифікатори
OwnerBlock	Implementation			
└	<Constructor>	Public □	□	NO□
└	oGetOwnerList	Public □		NO□
└	oIsOwner	Public □		NO□
└	oIsOwners	Public □		NO□

└	oAddOwner	Public □	□	onlyOwners notZeroAddress isNotOwner
└	oAddOwners	Public □	□	onlyOwners
└	oRemoveOwner	Public □	□	onlyOwners isOwner notZeroAddress
└	oRemoveOwners	Public □	□	onlyOwners
└	oRemoveAllOwners	Public □	□	onlyOwners
└	oRemoveAllOwnersExcept	Public □	□	onlyOwners notZeroAddress isOwner
RoundProposaWeightBlock	Implementation	OwnerBlock		
ProposalBlock	Implementation	OwnerBlock, RoundProposaWeightBlock		
└	pfViewAllProposals	Public □		roundValidIndex
└	proposalsAddSingle	Public □	□	onlyOwners roundValidIndex
└	pfAddSomeProposals	Public □	□	onlyOwners roundValidIndex
└	pfRemovesOneProposal	Public □	□	onlyOwners proposalValidIndex
└	pfRemovesOneProposalKeepIndex	Public □	□	onlyOwners proposalValidIndex
└	proposalsMinusAll	Public □	□	onlyOwners roundValidIndex
└	pfChange	Public □	□	onlyOwners proposalValidIndex
WeightBlock	Implementation	OwnerBlock, RoundProposaWeightBlock		
└	wViewOneAddrWeight	Public □		roundValidIndex
└	wViewAllAddrWeightweights	Public □		roundValidIndex
└	wAddAddressWeightNew	Public □	□	onlyOwners roundValidIndex
└	wAddAddressWeights	Public □	□	onlyOwners roundValidIndex
└	wRevoveOneAddressWeight	Public □	□	onlyOwners roundValidIndex
└	wRevoveAllAddressWeights	Public □	□	onlyOwners roundValidIndex
DelegateWeightBlock	Implementation	OwnerBlock, WeightBlock RoundProposaWeightBlock,		
└	wDelegateWeight	Public □	□	roundValidIndex notZeroAddress
TimeBlock	Implementation	OwnerBlock, RoundProposaWeightBlock		
└	tfStartVoting	Public □	□	onlyOwners roundValidIndex
└	tfFinisVotinghManually	Public □	□	onlyOwners roundValidIndex
RoundBlock	Implementation	OwnerBlock, RoundProposaWeightBlock		
└	rfCreateNewRouund	Public □	□	onlyOwners
LogicBlock	Implementation	OwnerBlock, ProposalBlock, RoundBlock, TimeBlock, DelegateWeightBlock		
└	<Constructor>	Public □	□	OwnerBlock
└	vote	Public □	□	proposalValidIndex tmVoteHasStarted

Додаток Г

Документація до смартконтрактів

Г.1 Документація до смартконтракту OwnerBlock

OwnerBlock - це контракт, призначений для управління списком голів. Цей контракт надає можливість додавання, видалення та перевірки адрес в якості голів.

Властивості:

- owners
    - Тип: address[] public

    - Опис: Масив адрес голів.

Модифікатори:

- onlyOwners
    - Перевіряє, чи є відправник повідомлення головою. Якщо відправник не є головою, викликається помилка.
- notZeroAddress
    - Перевіряє, чи є введена адреса нульовою. Якщо адреса є нульовою, викликається помилка.
- isOwner
    - Перевіряє, чи належить введена адреса до списку голів. Якщо адреса не належить до списку голів, викликається помилка.
- isNotOwner
    - Перевіряє, чи не належить введена адреса до списку голів. Якщо адреса належить до списку голів, викликається помилка.

Функції:

- oGetOwnerList()
    - Тип: public view
    - Опис: Повертає список всіх голів.
- oIsOwner(address _addr)
    - Тип: public view
    - Опис: Перевіряє, чи є вказана адреса головою.
- oIsOwners(address[] memory _ownersToCheck)
    - Тип: public view
    - Опис: Перевіряє, чи є всі вказані адреси головами.
- oAddOwner(address _addr)
    - Тип: public
    - Опис: Додає нову адресу до списку голів.
- oAddOwners(address[] memory _newOwners)
    - Тип: public
    - Опис: Додає декілька нових адрес до списку голів.

- oRemoveOwner(address _addr)
    - Тип: public
    - Опис: Видаляє вказану адресу зі списку голів.
- oRemoveOwners(address[] memory _addr)
    - Тип: public
    - Опис: Видаляє декілька адрес зі списку голів.
- oRemoveAllOwners()
    - Тип: public
    - Опис: Видаляє всіх голів, крім відправника повідомлення.
- oRemoveAllOwnersExcept(address _addr)
    - Тип: public
    - Опис: Видаляє всіх голів, крім вказаної адреси.

Ця документація була створена для смартконтракту OwnerBlock та слід використовувати її для розуміння та інтеграції контракту.

Г.2 Документація до контракту RoundProposaWeightBlock

Контракт RoundProposaWeightBlock надає функціональність для управління раундами, пропозиціями та вагами голосів.

Структури даних:

- Proposal
    - Опис: Структура, що описує пропозицію.
    - name (string): Назва пропозиції.
    - voteCounter (uint): Кількість голосів, що підтримують пропозицію.
- Weight
    - Структура, що описує вагу голосу.
    - voterAddress (address): Адреса виборця.
    - weight (uint): Вага голосу.
- Round

- Структура, що описує раунд голосування.
- proposals (Proposal[]): Список пропозицій у цьому раунді.
- weights (Weight[]): Список ваг у цьому раунді.
- timeStart (bool): Статус раунду (почався чи ні).
- timeFinish (uint): Час завершення раунду.

Змінні стану:

- rounds
  - Змінна для зберігання всіх раундів голосування. Тип: Round[].

Модифікатори:

- roundValidIndex(uint _roundIndex)
  - Модифікатор для перевірки валідності індексу раунду. Перевіряє, чи не виходить переданий індекс за межі допустимого діапазону.

Документація надає короткий огляд функціональності контракту RoundProposaWeightBlock. Для докладного розуміння рекомендується звертатися до вихідного коду контракту.

Г.3 Документація до контракту ProposalBlock

Контракт `ProposalBlock` надає функціональність для управління пропозиціями голосувань.

Модифікатори:

- proposalValidIndex
  - Опис: Модифікатор для перевірки валідності індексу пропозиції. Перевіряє, чи не виходить переданий індекс пропозиції за межі допустимого діапазону.

Функції:

- pfViewAllProposals
  - Опис: Функція для перегляду всіх пропозицій у вказаному раунді голосувань.

- proposalsAddSingle
    - Опис: Функція для додавання одного варіанту пропозиції для голосування у вказаний раунд.
- pfAddSomeProposals
    - Опис: Функція для додавання декількох варіантів пропозицій для голосування в вказаний раунд.
- pfRemovesOneProposal
    - Опис: Функція для видалення одного варіанту пропозиції голосування в вказаному раунді.
- pfRemovesOneProposalKeepIndex
    - Опис: Функція для видалення одного варіанту пропозиції голосування у вказаному раунді зі збереженням індексів інших варіантів.
- proposalsMinusAll
    - Опис: Функція для очищення всіх варіантів пропозицій голосування в вказаному раунді.
- pfChange
    - Опис: Функція для заміни існуючої пропозиції голосування на нову у вказаному раунді.

Примітка: Документація надає короткий огляд функціональності контракту `ProposalBlock`. Для докладного розуміння рекомендується звертатися до вихідного коду контракту.

Г.4 Документація до контракту WeightBlock

Контракт WeightBlock розширює можливості управління вагами голосів в раундах голосувань.

Модифікатори:

- addressNotInWeights: Перевіряє, чи вказана адреса вже є у списку ваг для вказаного раунду голосувань.

Функції:

- Перегляд:
  - wViewOneAddrWeight: Повертає вагу голосу конкретного виборця (за адресою) у вказаному раунді.
  - wViewAllAddrWeightweights: Повертає список усіх ваг голосів у вказаному раунді.
- Додавання:
  - wAddAddressWeightNew: Додає або оновлює вагу голосу конкретного виборця у вказаному раунді. Якщо вага менше одиниці, виборець видаляється зі списку.
  - wAddAddressWeights: Дозволяє додати або оновити ваги голосів для декількох виборців одразу.
- Видалення:
  - wRevoveOneAddressWeight: Видаляє вагу голосу конкретного виборця з вказаного раунду.
  - wRevoveAllAddressWeights: Очищає список усіх ваг голосів у вказаному раунді.

Г.5 Документація до контракту DelegateWeightBlock

Контракт DelegateWeightBlock розширює можливості управління вагами голосів в раундах голосувань, дозволяючи делегування ваги від одного учасника до іншого.

Функції:

- wDelegateWeight:
  - Опис:
    - Делегування ваги. Делегує вказану вагу від відправника до одержувача в вказаному раунді.
  - Параметри:
    - _roundIndex: Індекс раунду, в якому відбувається делегування.

- _to: Адреса одержувача.
- _weight: Вага, яку потрібно делегувати.

- Нотатки:
  - Неможливо делегувати самому собі.
  - Вага для делегування повинна бути більшою за нуль.
  - Перевіряється, що вага відправника більша або дорівнює вазі, яку він хоче делегувати.
  - Якщо після делегування вага відправника дорівнює нулю, він видаляється зі списку.
  - Якщо одержувач не знайдений у списку, його додають з новою вагою.

Г.6 Документація до контракту TimeBlock

Контракт TimeBlock вводить можливості керування часовими рамками для голосування у вказаних раундах.

Функції:

- tfStartVoting:
  - Опис: Старт голосування. Запускає голосування для вказаного раунду.
  - Параметри:
    - _roundIndex: Індекс раунду.
    - _votingDurationInSeconds: Тривалість голосування в секундах.
  - Нотатки:
    - Голосування може бути запущено тільки власником.
    - Час початку голосування визначається поточним часом блоку.
- tfFinisVotinghManually:
  - Опис: Ручне завершення голосування для вказаного раунду.
  - Параметри: _roundIndex: Індекс раунду.

- Нотатки:
  - Голосування може бути завершено тільки власником.
  - Після завершення голосування, статус голосування та час завершення скидаються.

Модифікатори:

- tmVoteHasStarted: Перевіряє, чи стартувало голосування для вказаного раунду.
- tmVoteHasNotStarted: Перевіряє, чи не стартувало голосування для вказаного раунду.

Г.7 Документація до контракту RoundBlock

Контракт для керування раундами голосування.

Функції:

- rfCreateNewRouund
  - Опис: Створює новий раунд голосування з вказаними пропозиціями та вагами голосів для кожного виборця.
- Параметри:
  - _proposalNames: Масив назв пропозицій. Тип: string[] (у пам'яті).
  - _addresses: Масив адрес виборців. Тип: address[] (у пам'яті).
  - _weights: Масив ваг для виборців. Тип: uint[] (у пам'яті).
- Обмеження:
  - Функцію може викликати лише власник контракту.
  - Масив адрес не повинен бути порожнім.
  - Масиви адрес та ваг повинні мати однакову довжину.
  - Потрібно мати щонайменше дві пропозиції.
- Логіка виконання:
  - Ініціалізація нового раунду голосування.
  - Додавання пропозицій до раунду.
  - Встановлення ваг для виборців.

Г.8 Документація до контракту LogicBlock

Основний контракт для логіки голосування.

Функції:

- vote
  - Опис: Функція голосування. Кожен виборець може голосувати тільки один раз.
  - Параметри:
    - _roundIndex: Індекс раунду. Тип: uint.
    - _proposalIndex: Індекс пропозиції. Тип: uint.
    - _weightToUse: Вага для голосування. Тип: uint.
  - Обмеження:
    - Функцію може викликати будь-який адрес.
    - Індекс пропозиції має бути валідним.
    - Голосування повинно бути розпочате.
    - Період голосування не повинен бути завершеним.
    - Виборець має бути знайдений в масиві ваг.
    - Вага для голосування має бути більше нуля.
    - Неможливо голосувати за видалену пропозицію.
  - Логіка виконання:
    - Шукаємо виборця в масиві ваг.
    - Перевіряємо достатність ваги для голосування.
    - Голосуємо за вибрану пропозицію, додаючи вагу до лічильника голосів пропозиції.
    - Зменшуємо вагу виборця на вказане значення.

Додаток Д

Тестування

```
// Импортуємо необхідні модулі та класи для розробки та тестування
смартконтрактів.
import { ethers } from "hardhat";
import { assert } from "chai";
import { Contract } from "ethers";
import { Signer } from "ethers";

// Визначаємо основний блок тестів для смартконтракту "LogicBlock".
describe("LogicBlock", function() {

  // Оголошуємо змінні, які будуть використовуватися в тестах.
  let logicBlock: Contract;
  let owner: Signer;
  let voter1: Signer;
  let voter2: Signer;
```

```
 // Функція, яка буде виконуватися перед кожним тестом у цьому блоку.
 beforeEach(async () => {
// Отримуємо доступні аккаунти в локальному середовищі.
[owner, voter1, voter2] = await ethers.getSigners();

// Створюємо новий екземпляр смартконтракту "LogicBlock".
const LogicBlockFactory = await ethers.getContractFactory("LogicBlock");
const ownerAddress = await owner.getAddress();
logicBlock = await LogicBlockFactory.deploy([ownerAddress]);
await logicBlock.deployed();
 });

 // Основний сценарій тестування.
 it("should allow valid voting", async () => {

// 1. Створити новий раунд голосування з пропозиціями та вагами.
const voter1Address = await voter1.getAddress();
const voter2Address = await voter2.getAddress();

await logicBlock.connect(owner).rfCreateNewRouund(
["Proposal1", "Proposal2"],
[voter1Address, voter2Address],
[10, 10]
);

// 2. Розпочати процес голосування.
await logicBlock.connect(owner).tfStartVoting(0, 1000);  // Голосування триватиме
1000 секунд

// 3. Виборець1 голосує за Пропозицію1 з вагою 5.
await logicBlock.connect(voter1).vote(0, 0, 5);

// Перевірте, чи була зменшена вага, а пропозиція отримала голос.
const proposal = await logicBlock.pfViewAllProposals(0);
assert.equal(proposal[0].voteCounter, 5);
assert.equal(await logicBlock.wViewOneAddrWeight(0, voter1Address), 5);

// 4. Спроба проголосувати, використовуючи більше, ніж доступна вага.
try {
await logicBlock.connect(voter1).vote(0, 0, 10);
assert.fail("Expected vote to revert due to insufficient weight, but it didn't");
 } catch (error) {
// Управління очікуваною помилкою, якщо це необхідно.
 }

// 5. Спроба проголосувати після завершення тривалості голосування.
// ... Маніпулювати часом тут, щоб перевищити тривалість голосування ...
// await assert.reverted(logicBlock.connect(voter1).vote(0, 0, 5), "Should revert because
voting has ended");
  });
});
```

Додаток Е

Документація до тесту

Огляд:

Тест написано для контракту `LogicBlock`, який надає функціональність голосування на основі ваги учасників.

Опис тесту:

Тест перевіряє основний сценарій голосування, гарантуючи, що голосування відбувається правильно і що вага учасників коректно враховується.

Сценарії:

1. Ініціалізація

- Тест починається з розгортання контракту `LogicBlock` із власником `owner`.

2. Створення нового раунду голосування

- Власник контракту (`owner`) створює новий раунд голосування з двома пропозиціями (`Proposal1` і `Proposal2`) і двома учасниками (`voter1` і `voter2`), кожній з яких присвоюється вага в 10 одиниць.

3. Запуск голосування

- Власник запускає голосування з тривалістю в 1000 секунд.

4. Процес голосування

- Учасник `voter1` голосує за `Proposal1`, використовуючи 5 одиниць своєї ваги.
- Тест перевіряє, що `Proposal1` отримав 5 голосів і що вага `voter1`, що залишилася, становить 5 одиниць.

5. Перевірка помилкових сценаріїв.

- Учасник `voter1` намагається проголосувати знову, але з вагою в 10 одиниць, що перевищує його поточну вагу, що залишилася. Тест перевіряє, що таке голосування викликає помилку.
- (Опціонально) Після завершення часу голосування, учасник `voter1` намагається проголосувати знову. Тест перевіряє, що таке голосування викликає помилку. (Для цього сценарію потрібна маніпуляція часом у середовищі виконання тесту).