НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ПОГОДЖЕНО **Декан факультету**	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ** **Завідувач кафедри інформаційних систем і технологій**
_____Ігор БОЛБОТ (підпис)	**_____Михайло ШВИДЕНКО** (підпис)
«_____» _____2026 р.	«_____» _____2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка смарт-контракту для управління ланцюгами постачання»
Спеціальність «126 Інформаційні системи та технології»
Освітньо-професійна програма «Інформаційні системи та технології»

Гарант освітньої програми
к.е.н., доцент _____ **Максим МОКРІЄВ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи _____ **Костянтин РОГОЗА**
(підпис)

Виконала _____ **Софія СТЕЦЮК**
(підпис)

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри інформаційних систем і технологій
к.е.н., доцент_____Михайло ШВИДЕНКО
(підпис)
«___» _____2026 р.

ЗАВДАННЯ ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Стецюк Софії Сергіївні

(прізвище, ім'я, по батькові)

Спеціальність «126 Інформаційні системи та технології»
Освітньо-професійна програма «Інформаційні системи та технології»
Тема бакалаврської кваліфікаційної роботи
«Розробка смарт-контракту для управління ланцюгами постачання» затверджена наказом від «19» грудня 2025 р. № 3205 «С»
Термін подання завершеної роботи на кафедру «23» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи: аналіз предметної області, існуючі рішення, інформація про учасників ланцюга постачання, дані про товари та їх характеристики, аналітичні та звітні дані, параметри безпеки та доступу.

Перелік питань, що підлягають дослідженню:

1. Аналіз існуючих підходів до управління смарт-контрактами.
2. Формування переліку функціональних і технічних вимог до проектованої системи.
3. Розробка архітектури блокчейн-системи.
4. Розробка програмної реалізації системи.
5. Оцінка ефективності розробленого рішення.

Дата видачі завдання « 19 » грудня 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Костянтин РОГОЗА**
(підпис)

Завдання взяв до виконання _____ **Софія СТЕЦЮК**
(підпис)

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

1. PoC (Proof of Capacity) — алгоритм консенсусу, що використовує доступний обсяг пам’яті, а не обчислювальну потужність, для верифікації транзакцій і майнінгу блоків.
2. PoW (Proof of Work) — алгоритм консенсусу, що базується на розв’язанні складних обчислювальних задач, потребує значних ресурсів і витрат електроенергії.
3. NFT (Non-Fungible Token) — унікальний токен, що представляє право власності на конкретний цифровий або фізичний актив і не підлягає взаємозаміні.
4. API (Application Programming Interface) — інтерфейс прикладного програмування, що дозволяє програмам взаємодіяти між собою.
5. GUI (Graphical User Interface) — графічний інтерфейс користувача, який забезпечує взаємодію користувача із системою через візуальні елементи.
6. SDK (Software Development Kit) — набір інструментів для розробки програмного забезпечення, що включає бібліотеки, документацію та зразки коду.
7. IDE (Integrated Development Environment) — інтегроване середовище розробки, що об’єднує інструменти для написання, компіляції та тестування програмного коду.
8. JDK (Java Development Kit) — набір інструментів для розробки програм на мові Java, що включає компілятор, бібліотеки, інструменти для налагодження та інше.
9. BouncyCastle — криптографічна бібліотека для Java, яка забезпечує реалізацію криптографічних алгоритмів і використовується для захисту даних.
10. Gradle — інструмент для автоматизації збирання проєктів, що полегшує управління залежностями, компіляцію та тестування.
11. Smart Contract — смарт-контракт, програма, яка автоматично виконує умови договору в блокчейні, забезпечуючи прозорість та незмінність.

12. Bytecode — байт-код, проміжний код, що виконується віртуальною машиною, наприклад, Java Virtual Machine (JVM).

13. Hash — хеш-функція, криптографічна функція, що генерує фіксований розмір вихідного рядка (хеш) з вхідних даних, використовується для захисту та перевірки цілісності даних.

14. Signum — блокчейн-платформа, яка використовує PoC для верифікації транзакцій і забезпечення безпеки, є основою для побудови системи у дипломному проєкті.

15. Emulator — емулятор, програмний інструмент для імітації роботи системи або її окремих компонентів у тестовому середовищі.

16. Rinkeby/Ropsten — тестові мережі блокчейну Ethereum, що використовуються для перевірки та тестування смарт-контрактів перед їх розгортанням у основній мережі.

17. Tx (Transaction) — транзакція, передача даних або цінностей між учасниками блокчейн-мережі, записується в блокчейн як запис, що підтверджує обмін.

18. SHA (Secure Hash Algorithm) — безпечний хеш-алгоритм, що використовується для генерації унікальних хешів даних, які забезпечують цілісність інформації.

ВСТУП

Актуальність теми. Сучасний етап розвитку інформаційних технологій характеризується інтенсивним впровадженням децентралізованих систем, серед яких ключове місце посідає технологія блокчейн. Вона стає фундаментом для трансформації фінансових операцій та автоматизації управління інтелектуальними активами. Проте широке розповсюдження блокчейну стримується низкою проблем, зокрема обмеженою масштабованістю та вразливістю систем безпеки. У зв'язку з цим, критичного значення набуває розробка методів ефективного адміністрування смарт-контрактів, які б гарантували конфіденційність даних за умови повної прозорості та децентралізації. Дослідження в цьому напрямку є стратегічно важливим для розбудови стійкої та продуктивної цифрової інфраструктури.

Мета роботи: проектування та програмна реалізація системи управління смарт-контрактами для управління ланцюгами постачання, яка за рахунок використання блокчейн-технологій забезпечує вищий рівень безпеки, прозорості та ефективності обміну інформацією між учасниками.

Об’єктом дослідження є процеси управління ланцюгами постачання в умовах використання блокчейн-технологій.

Предметом дослідження є процеси та механізми управління смарт-контрактами в ланцюгах постачання,

Для реалізації поставленої мети визначено такі завдання:

- проаналізувати існуючі підходи до управління смарт-контрактами та ідентифікувати їхні функціональні недоліки;
- сформувати перелік технічних вимог до проектованої блокчейн-системи;
- розробити архітектуру системи, адаптовану до вимог безпеки децентралізованого середовища;
- здійснити програмну реалізацію прототипу та провести його апробацію на відповідність визначеним критеріям;

- визначити перспективи масштабування та впровадження отриманих результатів.

Методи дослідження включають системний аналіз предметної області, порівняльний аналіз існуючих блокчейн-платформ і рішень, застосування функціонального та об’єктно-орієнтованого підходів до моделювання (UML-діаграми), проєктування логічної та фізичної моделей даних, розробку архітектури системи, програмну реалізацію смарт-контрактів, а також їх тестування в емуляторі та оцінювання ефективності й продуктивності розробленого рішення.

Практична значущість дослідження полягає у підвищенні ефективності, прозорості та безпеки управління ланцюгами постачання за рахунок використання смарт-контрактів, що забезпечують автоматизацію процесів, зменшення витрат, усунення посередників і підвищення надійності обміну даними в децентралізованому середовищі.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис смарт-контрактів і блокчейн-технологій.

Смарт-контракти є одним із фундаментальних елементів сучасних децентралізованих цифрових систем. Їх можна визначити як самовиконувані програмні механізми, що автоматично реалізують умови угоди, закладені у програмному коді [1, с. 34]. По суті, смарт-контракт являє собою спеціалізовану програму, яка забезпечує виконання транзакцій між сторонами за умови дотримання наперед визначених правил і параметрів. Поняття «смарт-контракт» уперше було запропоноване криптографом Ніком Сабо у 1994 році [1, с. 67]. Проте широкого поширення ця технологія набула лише з розвитком блокчейн-технологій, насамперед завдяки платформі Ethereum, яка надала можливість створювати та запускати смарт-контракти у межах децентралізованих мереж [1, с. 102].

Однією з головних переваг смарт-контрактів є їхня автономність, прозорість і високий рівень надійності [1, с. 28]. Такі контракти функціонують без участі посередників або контролюючих органів, що значно пришвидшує процес виконання угод і підвищує рівень довіри між сторонами. Після виконання встановлених умов смарт-контракт автоматично ініціює передбачені дії без додаткового втручання людини. Це дозволяє суттєво зменшити ризики шахрайства, маніпуляцій або помилок з боку третіх осіб [5, с. 39]. Крім того, використання смарт-контрактів сприяє скороченню витрат на проведення транзакцій, оскільки усувається необхідність залучення посередників та спрощується процес адміністрування угод [2, с. 45].

Сфера застосування смарт-контрактів є надзвичайно широкою. Зокрема, у фінансовій галузі автоматизація кредитних договорів дозволяє значно прискорити взаємодію між кредиторами та позичальниками, а також мінімізувати кількість бюрократичних процедур [2, с. 63]. У правовій сфері смарт-контракти можуть використовуватися для реєстрації прав власності,

підтвердження юридичних домовленостей та автоматичного контролю за виконанням умов угод [2, с. 92]. У логістиці вони дають змогу відстежувати переміщення товарів, оптимізувати ланцюги постачання та підвищувати прозорість операцій [2, с. 77]. Не менш важливим є їхнє застосування у сфері страхування, де автоматизовані виплати за страховими випадками допомагають підвищити ефективність обробки звернень і мінімізувати вплив людського фактора [2, с. 33].

Ключовою технологічною основою функціонування смарт-контрактів є блокчейн. Блокчейн являє собою децентралізовану розподілену базу даних, у якій усі транзакції зберігаються у вигляді послідовного ланцюжка блоків [2, с. 55]. Кожен блок містить інформацію про попередні операції, що забезпечує цілісність, послідовність і незмінність даних. Такий рівень захисту досягається завдяки використанню сучасних криптографічних методів шифрування та хешування, які роблять систему стійкою до зовнішніх втручань і несанкціонованих змін [2, с. 88].

Однією з найважливіших характеристик блокчейну є децентралізація. Усі учасники мережі мають рівноправний доступ до спільної бази даних, а рішення щодо внесення змін до системи ухвалюються шляхом консенсусу [2, с. 103]. Завдяки цьому усувається необхідність у центральному керівному органі, що значно підвищує рівень безпеки та стійкості системи. Ще однією важливою перевагою блокчейну є прозорість, оскільки кожен учасник мережі має можливість перевіряти інформацію про всі транзакції, що містяться у блоках [2, с. 132]. Незмінність даних гарантує, що інформація, записана до блокчейну, не може бути змінена або видалена без погодження всієї мережі [2, с. 57]. Саме тому блокчейн є ефективним інструментом для зберігання критично важливої інформації, зокрема фінансових операцій, юридичних документів або медичних записів.

Поєднання блокчейн-технологій і смарт-контрактів формує новий рівень довіри між учасниками цифрових процесів, який не залежить від центральних посередників чи регуляторів [2, с. 69]. Це робить зазначені технології

надзвичайно важливими для розвитку сучасної цифрової економіки та створення нових моделей взаємодії між користувачами.

Публічні блокчейни, такі як Bitcoin та Ethereum, функціонують у відкритих мережах, де будь-який користувач може брати участь у підтвердженні транзакцій, переглядати інформацію та взаємодіяти із системою. Такий підхід забезпечує високий рівень прозорості, однак водночас потребує потужних механізмів захисту від кібератак і зловмисних дій. На відміну від публічних, приватні блокчейни створюються для використання всередині окремих організацій або компаній, де доступ до мережі є контрольованим і обмежується визначеним колом учасників. Такі системи забезпечують вищий рівень конфіденційності та дозволяють організаціям контролювати доступ до даних і транзакцій. Крім того, приватні блокчейни характеризуються більшою швидкістю обробки операцій, хоча при цьому частково втрачається рівень децентралізації та відкритості мережі. Основні характеристики блокчейн-технологій доцільно розглянути детальніше у таблиці 1.1.

Таблиця 1.1

Характеристика блокчейн-технологій

Характеристика	Опис
Децентралізація	Блокчейн не має централізованого органу або сервера. Всі учасники мережі мають рівний доступ до інформації, що забезпечує стійкість до атак та втручань третіх осіб.
Прозорість і незмінність	Всі транзакції є публічними, і будь-хто може перевірити їхню достовірність. Після додавання даних до блокчейну їх неможливо змінити або видалити, що захищає від маніпуляцій.
Безпека	Використання криптографії забезпечує захист даних від несанкціонованого доступу. Кожен блок має криптографічний підпис, що зв'язує його з попереднім блоком, створюючи безпечний ланцюг.
Смарт-контракти	Смарт-контракти автоматизують виконання угод між сторонами без участі посередників. Контракти виконуються автоматично при досягненні визначених умов.
Незмінність (Immutability)	Після того, як дані додані до блокчейну, вони не можуть бути змінені або видалені. Це гарантує постійну історію всіх транзакцій та високу надійність даних.

Продовження таблиці 1.1

Анонімність	Учасники можуть взаємодіяти з блокчейном без необхідності розкривати свою особистість. Це забезпечує конфіденційність і захист персональних даних.
Консенсусні алгоритми	Для підтвердження транзакцій блокчейн використовує різні механізми консенсусу (Proof of Work, Proof of Stake), які забезпечують єдину версію даних у мережі.
Транспарентність	Всі учасники можуть переглядати транзакції в режимі реального часу, що забезпечує високий рівень прозорості та запобігає шахрайству.

Підсумовуючи, синергія смарт-контрактів та блокчейн-технологій гарантує високий ступінь детермінованості та захищеності процесів реалізації угод. Це зумовлює стратегічну значущість зазначених інструментів для модернізації ключових галузей сучасної економіки та систем державного і корпоративного управління.

1.2 Аналіз існуючих рішень щодо управління смарт-контрактами.

До актуальних рішень у сфері адміністрування смарт-контрактів належать такі платформи, як Ethereum, Hyperledger Fabric, EOS та Tezos. Кожна з цих систем базується на унікальних архітектурних підходах до реалізації смарт-контрактів, пропонуючи різні співвідношення рівнів децентралізації, масштабованості та інформаційної безпеки.

Ethereum залишається найбільш розповсюдженою платформою для розгортання смарт-контрактів. Її децентралізована архітектура забезпечує розробникам можливість створення блокчейн-орієнтованих алгоритмів та запуск децентралізованих додатків (dApps). Ключовою перевагою Ethereum є розвинена екосистема та численна спільнота розробників, що забезпечує сталу підтримку та модернізацію платформи. Водночас до суттєвих недоліків системи належать висока вартість транзакційних витрат (Gas fees) та обмежена пропускна здатність мережі, що створює перешкоди для її масштабування. Деталізований процес виконання смарт-контрактів наведено на рис. 1.1.

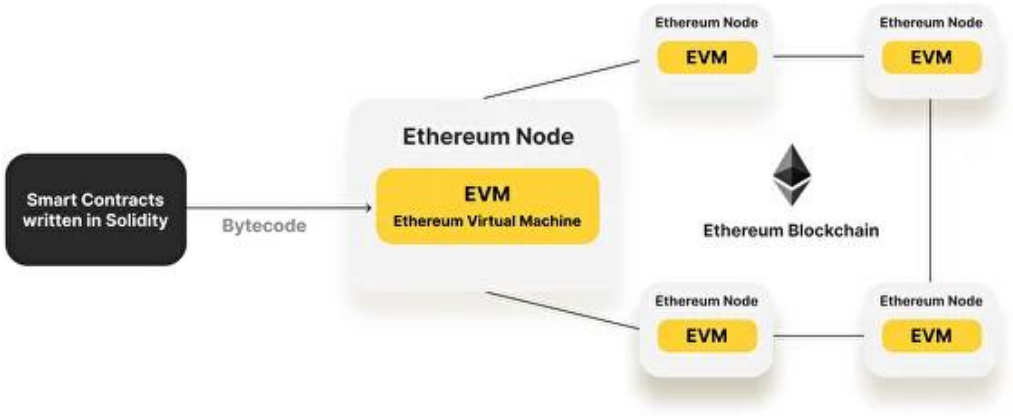

Рис.1.1 Процес виконання смарт-контрактів

Розробка смарт-контрактів здійснюється мовою програмування Solidity з подальшою компіляцією у байт-код. Зазначений код передається вузлам мережі (Ethereum Nodes), які інтегрують віртуальну машину Ethereum (EVM). Саме EVM відповідає за інтерпретацію та безпосереднє виконання байт-коду. Оскільки кожен вузол мережі зберігає ідентичну копію блокчейну, синхронне виконання операцій у межах EVM на всіх вузлах гарантує цілісність та узгодженість результатів у межах усієї децентралізованої інфраструктури.

На відміну від публічних мереж, Hyperledger Fabric є приватною блокчейн-платформою корпоративного рівня, розробленою під егідою Linux Foundation у рамках консорціуму Hyperledger. Фундаментальною особливістю Fabric є її модульна архітектура, що надає можливість гнучкого налаштування параметрів мережі відповідно до специфічних бізнес-вимог та корпоративних стандартів. Архітектурні особливості даної платформи схематично зображено на рис. 1.2.

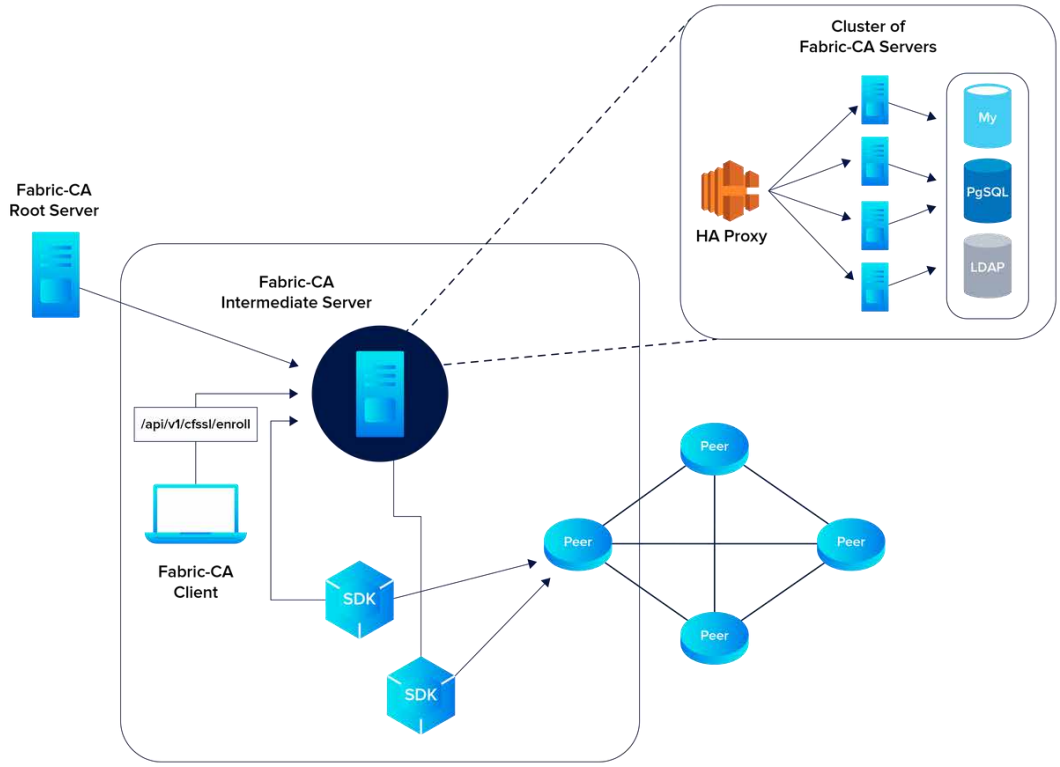

Рис.1.2 Архітектура Hyperledger Fabric

Архітектура платформи базується на функціонуванні кореневого сервера сертифікації (Fabric-CA Root Server), що виступає центральним вузлом для емісії та адміністрування цифрових сертифікатів і ключів доступу учасників. Проміжний сервер сертифікації (Fabric-CA Intermediate Server) забезпечує ієрархічну передачу сертифікатів від кореневого рівня до кінцевих користувачів або вузлів. Взаємодія з центром сертифікації здійснюється за допомогою клієнтського компонента Fabric-CA Client, який дозволяє суб'єктам отримувати необхідні облікові дані для автентифікації в мережі.

Ключовими елементами інфраструктури є вузли-учасники (Peer Nodes), відповідальні за зберігання локальних копій реєстру та виконання програмної логіки смарт-контрактів, відомих як «chaincode». Процес упорядкування транзакцій та забезпечення мережевого консенсусу перед фіксацією даних у блокчейні покладено на компонент Orderer. Для забезпечення відмовостійкості та ефективного розподілу навантаження в системі використовується кластеризація серверів сертифікації з інтеграцією балансувальника HA Proxy.

Важливою перевагою платформи є універсальність механізму смарт-контрактів (Chaincode), які можуть бути реалізовані на загальновживаних мовах

програмування, як-от Go або Java. Це забезпечує вищий ступінь гнучкості розробки порівняно з платформами, обмеженими спеціалізованими мовами.

Фундаментальними преференціями Hyperledger Fabric є адаптивність та конфіденційність, що реалізуються через механізм приватних каналів. Даний інструментарій дозволяє створювати ізольовані середовища для взаємодії окремих груп учасників, гарантуючи захист сенситивних даних від несанкціонованого доступу третіх сторін. Такий розподіл рівнів доступу є критичним для корпоративного сектору, де пріоритетними є контроль та інформаційна приватність.

Fabric підтримує плюралістичну модель консенсусу, дозволяючи інтегрувати різні алгоритми валідації транзакцій (зокрема Practical Byzantine Fault Tolerance (PBFT) або Raft) залежно від бізнес-потреб. Це сприяє досягненню високої пропускної здатності та масштабованості, що необхідно для обробки значних масивів даних. Функціонуючи за принципом дозволеного (permissioned) блокчейну, система вимагає попередньої авторизації учасників, що підвищує загальний рівень безпеки та забезпечує відповідність регуляторним нормам у фінансових та державних інституціях.

На противагу приватним рішенням, платформа EOS є публічним блокчейном, спроектованим для подолання обмежень масштабованості та продуктивності. Застосування механізму делегованого підтвердження частки (Delegated Proof of Stake, DPoS) дозволяє мережі опрацьовувати тисячі операцій за секунду, що суттєво перевищує показники Bitcoin або Ethereum. Орієнтованість на високу пропускну здатність робить EOS оптимальним середовищем для ресурсномістких сервісів, таких як соціальні мережі, ігрові додатки та платіжні шлюзи.

В основі моделі DPoS лежить принцип елективної валідації: власники токенів обирають делегатів, на яких покладається відповідальність за генерацію блоків та верифікацію транзакцій. Схематичне відображення процесу делегованого підтвердження частки наведено на рис. 1.3.

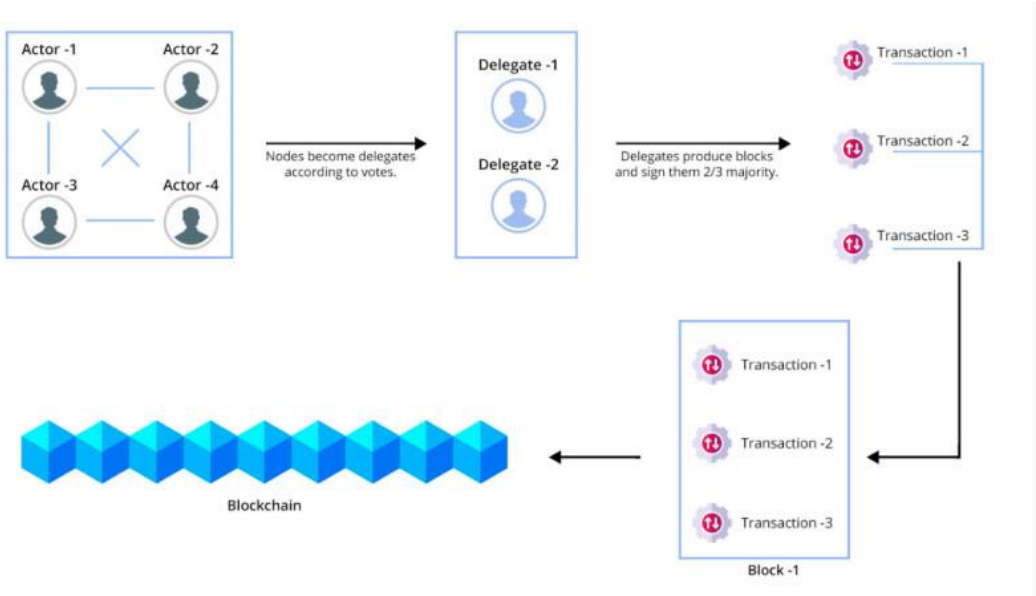

Рис.1.3 Процес делегованого підтвердження частки DPOS в блокчейн-платформі

Згідно з наведеною схемою, на початковому етапі в мережі функціонує сукупність акторів (Actors), кожен з яких володіє правом голосу. За результатами елективного процесу визначаються вузли-делегати (Delegates), які наділяються повноваженнями щодо генерації блоків. Делегати здійснюють формування нових блоків шляхом інклюзії транзакцій, при цьому верифікація блоку потребує підтвердження кваліфікованою більшістю (2/3 голосів) обраних делегатів.

Після завершення етапу агрегації та обробки, транзакції консолідуються у блок (Block), який інтегрується у структуру ланцюжка (Blockchain). Отже, модель DPoS базується на делегуванні повноважень щодо виконання операцій та створення блоків на основі волевиявлення учасників, що забезпечує високу операційну ефективність та оперативне підтвердження транзакцій.

Даний підхід дозволяє радикально скоротити час очікування фіналізації операцій та оптимізувати продуктивність мережі. На відміну від концепції Proof of Work (PoW), де право на генерацію блоку здобувається через конкурентне розв'язання складних обчислювальних задач, у моделі DPoS привілей створення блоків належить обмеженій групі обраних суб'єктів. Це нівелює надлишкове споживання ресурсів і забезпечує високу енергоефективність платформи.

Додатковою перевагою EOS є можливість паралельної обробки транзакцій, що суттєво підвищує загальну пропускну здатність системи.

Розробка смарт-контрактів в екосистемі EOS здійснюється з використанням мов C++ та стандартів WebAssembly (WASM), що забезпечує гнучкість інструментарію для розробників. Висока економічна ефективність платформи зумовлена відсутністю прямих комісій за транзакції; натомість використовується модель резервування системних ресурсів (CPU, RAM, Network). Такий підхід робить EOS привабливою базою для масштабних і ресурсомістких децентралізованих проектів.

Проте критичним аспектом архітектури EOS є ризик централізації, притаманний моделі DPoS. Концентрація повноважень у вузького кола делегатів може призвести до узурпації управління найбільш капіталізованими учасниками, що знижує ступінь децентралізації порівняно з системами типу Bitcoin чи Ethereum. Попри регулярність процедур переобрання, існує ймовірність впливу великих власників токенів на результати голосування, що створює загрозу олігополістичного управління мережею.

Tezos представляє собою децентралізовану платформу для управління смарт-контрактами, відмінною рисою якої є механізм самокерованого оновлення (self-amendment). Ця властивість дозволяє модернізувати протокол без ініціації хардфорків — примусового розділення ланцюжка, що є характерним для Bitcoin або Ethereum. Завдяки системі внутрішнього голосування мережа здатна інтегрувати функціональні інновації та усувати вразливості, зберігаючи цілісність інфраструктури.

Механізм ончейн-управління (on-chain governance) є фундаментальним елементом Tezos. Власники токенів (XTZ) беруть безпосередню участь у стратегічному розвитку платформи: кожна пропозиція щодо вдосконалення протоколу проходить багатоетапну процедуру валідації спільнотою. Така демократична модель спрямована на забезпечення стабільності та еволюційного розвитку блокчейну на основі консенсусу, що виключає ризики розколу мережі.

Важливою характеристикою Tezos є використання алгоритму консенсусу Proof of Stake (PoS). У межах цієї моделі право на генерацію блоків та верифікацію транзакцій корелює з обсягом активів, що перебувають у власності або управлінні учасника. Даний підхід є значно менш енергоємним порівняно з PoW-системами. Економічна стимуляція безпеки мережі реалізується через систему винагород, пропорційних внеску кожного валідатора.

Також Tezos підтримує модель делегування (LPoS), яка дозволяє власникам активів передавати свої права на підтвердження блоків іншим вузлам (бейкерам) без відчуження самих токенів. Це залучає до процесів підтримки мережі широке коло учасників, забезпечуючи масштабованість та енергетичну ефективність системи. Модульна архітектура та здатність до безперервної адаптації роблять Tezos стабільною основою для довгострокових проектів та децентралізованих додатків (dApps). Архітектурна схема функціонування даної блокчейн-платформи представлена на рис. 1.4.

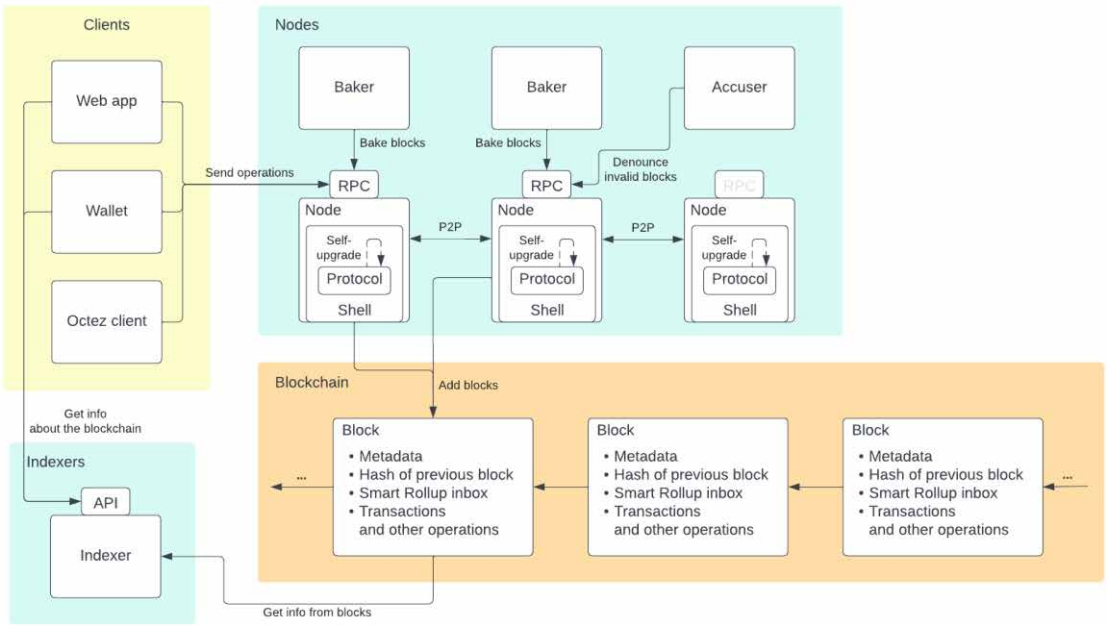

Рис. 1.4 Архітектура блокчейн-платформи Tezos

Клієнтська частина екосистеми включає вебдодатки, криптовалютні гаманці та клієнт Octez, які забезпечують взаємодію користувачів із блокчейном шляхом надсилання операцій і отримання актуальної інформації про стан мережі

через API та індексатори. Саме ці компоненти дозволяють користувачам здійснювати транзакції, перевіряти баланси, взаємодіяти зі смарт-контрактами та отримувати доступ до даних блокчейну у зручному форматі.

Вузли мережі виконують широкий спектр функцій, необхідних для стабільного функціонування блокчейну. Зокрема, вони можуть створювати нові блоки, що у мережі Tezos називається процесом «baking», перевіряти коректність і валідність блоків, а також забезпечувати дотримання правил та консенсусу мережі. Кожен вузол складається з протоколу та спеціальної оболонки, які мають можливість автоматичного оновлення, що дозволяє підтримувати актуальність програмного забезпечення без необхідності повної зупинки мережі. Для забезпечення синхронізації даних вузли взаємодіють між собою через однорангові механізми P2P, завдяки чому всі учасники мережі мають доступ до однакової та узгодженої інформації.

Блокчейн виконує роль основного сховища даних, у якому зберігаються всі транзакції, операції та службова інформація у вигляді послідовності блоків. Кожен блок містить хеш попереднього блоку, метадані, інформацію про транзакції та інші операції, що забезпечує цілісність і незмінність структури мережі. Такий підхід дозволяє гарантувати безпечне зберігання інформації та унеможливлює несанкціоноване втручання у вже записані дані.

Важливу роль у роботі екосистеми відіграють індексатори, які відповідають за обробку інформації про блоки та надання швидкого доступу до неї через API для зовнішніх застосунків і сервісів. Завдяки індексаторам сторонні програми можуть оперативно отримувати необхідні дані про транзакції, блоки, адреси та стан смарт-контрактів, що значно спрощує інтеграцію блокчейн-технологій у різноманітні цифрові сервіси.

Економічна модель Tezos також базується на системі стимулів для учасників мережі, які підтримують її безпеку, стабільність та ефективність. Учасники отримують винагороди за створення нових блоків («бейкінг»), а також за участь у голосуванні щодо оновлень і пропозицій розвитку мережі. Такий механізм формує стійку економічну мотивацію для користувачів, заохочуючи їх

активно підтримувати функціонування блокчейну та брати участь у процесах управління мережею. Завдяки цьому Tezos забезпечує не лише технічну децентралізацію, а й ефективну модель саморегулювання та розвитку екосистеми.

Загалом платформа Tezos пропонує сучасний підхід до розвитку блокчейн-технологій, поєднуючи стійкість до форків, енергоефективний механізм Proof of Stake та гнучке управління протоколом через систему голосування. Це дозволяє мережі адаптуватися до змін без поділу блокчейну та забезпечує стабільний розвиток екосистеми.

Завдяки високому рівню безпеки, масштабованості та можливості децентралізованого управління Tezos вважається однією з найбільш перспективних платформ для створення децентралізованих застосунків і бізнес-рішень у блокчейн-середовищі. Узагальнення основних характеристик платформи наведено у таблиці 1.2.

Таблиця 1.2

Плюси і мінуси блокчейн-платформ

Платформа	Плюси	Мінуси
Ethereum	- Широка екосистема і велика кількість розробників - Підтримка децентралізованих додатків (dApps) - Популярна мова Solidity	- Високі комісії за транзакції (Gas fees) - Низька масштабованість - Проблеми з продуктивністю під час завантажень
Hyperledger Fabric	- Приватний блокчейн із високою конфіденційністю - Гнучкість налаштування під корпоративні потреби - Підтримка модульної архітектури	- Підходить переважно для приватних систем - Складніша у впровадженні та управлінні в порівнянні з публічними блокчейнами
EOS	- Висока пропускна здатність - Низькі комісії за транзакції - Підтримка C++ та WebAssembly для смарт-контрактів	- Можлива централізація через DPoS - Критика за зниження рівня децентралізації
Tezos	- Самооновлення без необхідності хардфорків - Використання Proof of Stake з низьким енергоспоживанням	- Складніший процес голосування та управління - Порівняно менша спільнота розробників

Аналіз існуючих рішень у сфері управління смарт-контрактами свідчить, що кожна з провідних блокчейн-платформ — Ethereum, Hyperledger Fabric, EOS та Tezos — має власний набір сильних і слабких сторін, які визначають доцільність її використання в різних сценаріях. Ethereum, як одна з найпоширеніших публічних платформ для розгортання смарт-контрактів, забезпечує розвинену екосистему та широкий набір інструментів для розробників, однак водночас стикається з обмеженнями масштабованості та високими комісіями за транзакції. Hyperledger Fabric характеризується високим рівнем гнучкості, модульною архітектурою та посиленим захистом конфіденційності, проте його застосування переважно орієнтоване на корпоративні та приватні рішення, а не на публічні блокчейн-мережі.

Платформа EOS забезпечує високу швидкість обробки транзакцій і відносно низькі комісійні витрати, що робить її привабливою для високонавантажених застосунків, однак вона часто піддається критиці через потенційні ризики централізації управління мережею. У свою чергу, Tezos вирізняється механізмом самооновлення протоколу та енергоефективною моделлю консенсусу Proof of Stake, хоча при цьому потребує більш складного підходу до організації процесів управління та оновлення мережі.

Таким чином, вибір конкретної блокчейн-платформи безпосередньо залежить від вимог проєкту, зокрема щодо рівня продуктивності, децентралізації, безпеки та конфіденційності, що визначає оптимальність її застосування в кожному окремому випадку.

1.3 Огляд нормативної бази та вимог до блокчейн-технологій.

Наступним логічним етапом системного аналізу програмного забезпечення є комплексне дослідження та детальна специфікація функціональних і нефункціональних вимог, що висуваються до проектованої системи. Це дозволяє сформувати цілісне бачення архітектури та забезпечити відповідність розробки встановленим критеріям якості.

Функціональні вимоги до системи управління смарт-контрактами, яка базується на технологіях розподіленого реєстру, детально описують повний спектр можливостей, необхідних для коректного та ефективного функціонування алгоритмів у децентралізованому середовищі. Ці вимоги охоплюють усі аспекти життєвого циклу контрактів – від їх ініціалізації до виконання складних транзакційних операцій. Основні блоки вимог включають наступні складові (детальніше представлено у табл. 1.3):

- Комплексна емуляція блокчейн-середовища: передбачає створення програмної моделі для послідовної генерації блоків, забезпечення цілісності ланцюга та імітацію процесу обробки вхідних транзакцій з урахуванням усіх необхідних метаданих, що є критично важливим для відтворення реальних умов експлуатації.
- Адміністрування та життєвий цикл смарт-контрактів: включає процеси реєстрації нових контрактів у мережі з автоматичним розрахунком активаційної комісії, підтримку методів прямої та опосередкованої взаємодії між вузлами, а також інтелектуальне управління станами контрактів, зокрема реалізацію режиму «очікування» для оптимізації ресурсів.
- Механізми обробки ставок та аукціонна логіка: забезпечення технічної можливості проведення різнотипних торгів (класичних, голландських тощо) з чітким дотриманням часових рамок (тайм-аутів) та автоматичною фіксацією переможця при досягненні заданих цінових параметрів.

- Інтеграція та підтримка екосистеми NFT: реалізація функціоналу для безперешкодної передачі прав власності на цифрові активи, а також впровадження алгоритмів автоматичного нарахування роялті авторам та сервісних зборів платформі під час проведення вторинних продажів.
- Алгоритмізація ігрових механік та лотерей: розробка математично обґрунтованих модулів для проведення децентралізованих розіграшів, де результати базуються на криптографічних хешах блоків, що гарантує неупередженість процесу та автоматизацію виплат виграшів.
- Верифікація даних та інформаційна безпека: застосування передових стандартів хешування (зокрема родини SHA256) для забезпечення цілісності кожного повідомлення в системі, а також багаторівнева перевірка даних для запобігання несанкціонованим маніпуляціям.
- Управління суб’єктами та обліковими записами: надання кожному учаснику унікального ідентифікатора з підтримкою актуального балансу, а також забезпечення можливості адміністрування адрес, що належать як користувачам, так і самим контрактам.
- Системний аудит та логування активності: створення деталізованого журналу подій, який дозволяє здійснювати ретроспективний аналіз усіх дій у системі (створення блоків, виклик методів, транзакції), забезпечуючи тим самим абсолютну прозорість функціонування.

Нефункціональні вимоги відіграють фундаментальну роль у забезпеченні стабільності, живучості та загальної надійності системи управління смарт-контрактами. Вони визначають не лише те, «що» робить система, а й те, «як» вона виконує свої функції з точки зору якості, безпеки та користувацького досвіду.

Ці вимоги окреслюють технічні рамки, у межах яких система повинна демонструвати високу ефективність навіть за умов пікових навантажень. Зокрема, вони охоплюють такі критичні параметри (детальніше у табл. 1.3):

- Продуктивність та чутливість: здатність архітектури оперативно опрацьовувати значні масиви транзакцій та запитів користувачів, мінімізуючи

затримки (latency) і забезпечуючи стабільно високу швидкість реагування на зовнішні події.

- Системна безпека та конфіденційність: гарантування цілісності інформаційних активів та захист від зовнішніх загроз через використання стійких криптографічних протоколів, механізмів суворої аутентифікації та розмежування прав доступу.
- Масштабованість та адаптивність: можливість системи зберігати задані показники ефективності при експоненціальному зростанні обсягів даних, кількості активних користувачів або вузлів мережі, що забезпечує довгострокову життєздатність проекту в умовах цифрової трансформації.

Таблиця 1.3

Нефункціональні вимоги до системи

Категорія	Вимоги
Продуктивність	Обробка великої кількості транзакцій та запитів із мінімальною затримкою; швидке підтвердження операцій та низька латентність.
Безпека	Захист від несанкціонованого доступу та маніпуляцій через криптографічні алгоритми, шифрування даних та механізми аутентифікації.
Масштабованість	Підтримка зростаючого обсягу транзакцій, користувачів та даних без втрати продуктивності; можливість горизонтального масштабування.
Зручність використання	Інтуїтивний та простий інтерфейс для налаштування та управління контрактами; документація та навчальні матеріали для користувачів.
Доступність	Постійний доступ до системи 24/7 з мінімальним простоєм; швидке відновлення після збоїв або пікових навантажень.
Сумісність	Підтримка інтеграції з іншими платформами та середовищами; відповідність стандартам блокчейн-мереж для забезпечення інтероперабельності.
Відмовостійкість	Підтримка стабільної роботи при збої одного або кількох компонентів; автоматичне відновлення з мінімальним впливом на роботу системи.
Підтримка та оновлення	Можливість внесення оновлень і поліпшень без порушення роботи поточних контрактів; надання оновлень без значних простоїв системи.

Таким чином, нефункціональні вимоги служать основою для створення надійної, масштабованої, безпечної та продуктивної платформи, що здатна підтримувати динамічний розвиток у сфері управління смарт-контрактами на блокчейні.

1.4 Постановка завдання для розробки системи управління смарт-контрактами.

Стратегічна мета розробки системи управління смарт-контрактами полягає у проектуванні та реалізації комплексної програмної платформи, що забезпечує повний життєвий цикл децентралізованих алгоритмів – від їхнього моделювання та ініціалізації до безпосереднього виконання в межах блокчейн-інфраструктури.

Ключовим вектором реалізації даного проекту є впровадження механізмів транспарентного та повністю автономного функціонування цифрових угод, що дозволяє нівелювати потребу в залученні централізованих посередників або третіх сторін. Окрім автоматизації бізнес-логіки, особливий акцент у роботі зроблено на забезпеченні високої пропускної здатності та потенціалу масштабованості системи. Одночасно з цим, критично важливими аспектами є дотримання суворих стандартів інформаційної безпеки та створення ергономічного інтерфейсу взаємодії, що дозволить кінцевим користувачам ефективно оперувати інструментарієм децентралізованих технологій без глибокої технічної підготовки.

Розробка користувацького інтерфейсу: створення зручного інтерфейсу для взаємодії зі смарт-контрактами, який дає змогу користувачам легко створювати, редагувати та запускати контракти завдяки інтуїтивній навігації та простій структурі елементів.

Система керування смарт-контрактами: реалізація функціоналу для розгортання, активації, зміни та завершення смарт-контрактів із автоматичним виконанням умов відповідно до заданого алгоритму підтвердження.

Підтримка різних мов програмування для смарт-контрактів: забезпечення можливості розробки смарт-контрактів із використанням різних мов, зокрема Solidity та C++, що підвищує гнучкість системи та розширює коло розробників.

Обробка транзакцій: інтеграція фінансових операцій із підтримкою криптовалют і токенів, а також можливістю підключення криптогаманців та сервісів обміну для виконання операцій у межах системи.

Масштабованість платформи: забезпечення стабільної роботи системи при значному зростанні кількості користувачів і транзакцій без втрати продуктивності, з використанням підходів рівня Layer 2.

Безпека та захист даних: впровадження механізмів шифрування, багаторівневої автентифікації, захисту від DDoS-атак, резервного копіювання даних та дотримання вимог політик AML/CFT.

Журналювання та моніторинг операцій: реалізація системи ведення журналу транзакцій для перегляду історії операцій, що забезпечує прозорість, контроль і можливість аудиту виконання смарт-контрактів.

Підтримка мультипідпису: впровадження механізму мультипідпису для критичних транзакцій, які потребують підтвердження кількома учасниками, що підвищує рівень безпеки системи.

Вимоги до продуктивності та безпеки:

Продуктивність: забезпечення високої швидкості обробки великої кількості транзакцій без затримок і збоїв;

Безпека: захист від зовнішніх атак, шифрування даних, надійна автентифікація користувачів та протидія DDoS-атакам;

Масштабованість: здатність системи ефективно працювати при зростанні навантаження, кількості користувачів і обсягу даних.

Узагальнюючи викладене, проектована система повинна поєднувати в собі безкомпромісну надійність функціонування, високу обчислювальну потужність та багаторівневий захист інформаційних активів. Водночас критично важливою умовою є дотримання принципів юзабіліті, що забезпечує інтуїтивно зрозумілу взаємодію кінцевих користувачів із децентралізованим середовищем, нівелюючи складність базових технологічних процесів.

2 МОДЕЛЮВАННЯ СИСТЕМИ УПРАВЛІННЯ СМАРТ-КОНТРАКТАМИ

2.1 Вибір підходів до моделювання: функціональний та об'єктно-орієнтований підходи.

Під час моделювання системи за допомогою функціонального та об’єктно-орієнтованого підходів у UML застосовуються різні типи діаграм, які відображають функціональність, структуру та динаміку роботи системи. Вибір конкретного підходу залежить від цілей і завдань моделювання.

Функціональний підхід орієнтований на визначення та опис функцій системи, її основних процесів і взаємодії користувачів із цими процесами. Його головна мета полягає у визначенні того, які операції виконує система, без детального розгляду технічної реалізації. Основна увага приділяється тому, що саме робить система, а не тому, яким чином це реалізовано [10, с. 54].

Для моделювання функціональних вимог у межах цього підходу використовуються діаграми прецедентів та діаграми активності. Вони дозволяють на високому рівні абстракції відобразити бізнес-процеси та логіку роботи системи.

Функціональний підхід допомагає аналізувати взаємодію між користувачами та системою, забезпечуючи зрозуміле представлення її можливостей для всіх учасників проєкту – користувачів, аналітиків і розробників. Таке моделювання особливо корисне на початкових етапах розробки, оскільки дозволяє чітко визначити вимоги до системи та зменшити ризик виникнення помилок у подальшому проєктуванні й реалізації [10, с. 102].

Одним із ключових інструментів функціонального моделювання є діаграма прецедентів (use case diagram). Вона відображає основні функції системи – прецеденти – та показує взаємодію між користувачами (акторами) і системою. Основне призначення такої діаграми полягає у візуальному представленні

функцій, які повинна забезпечувати система, а також ролі користувачів у виконанні цих функцій. Це дає змогу визначити основні сценарії використання системи та її функціональні можливості, що є важливим як для бізнес-аналітиків, так і для розробників [11, с. 54].

Кожен прецедент на діаграмі описує певну функцію системи, яка виконує конкретне завдання або надає певну послугу користувачу чи іншій системі [11, с. 42]. Наприклад, у системі управління контрактами одним із прецедентів може бути «Реєстрація нового контракту», а актором – адміністратор, який відповідає за створення та керування контрактами [11, с. 102].

Завдяки наочному поданню функцій і взаємодій діаграма прецедентів покращує комунікацію між учасниками проєкту та допомагає всім зацікавленим сторонам чітко розуміти можливості системи та способи роботи з нею [11, с. 17]. Детальніше діаграму прецедентів для нашої системи наведено на рис. 2.1.

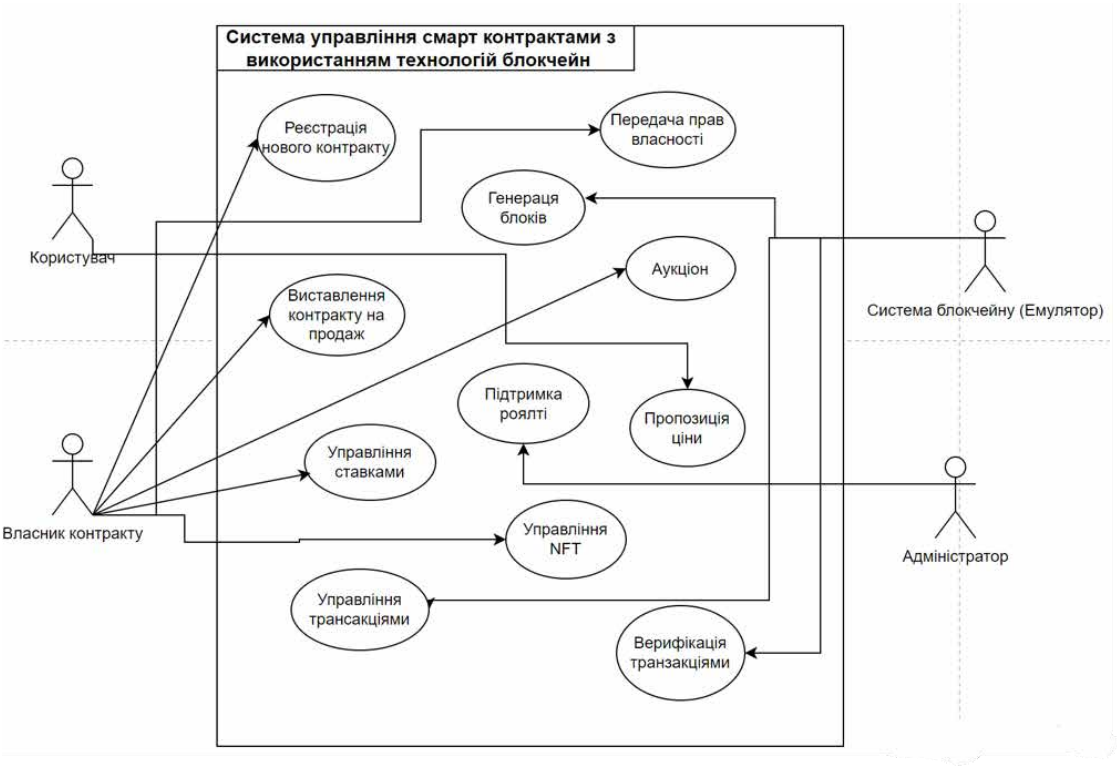

Рис. 2.1 Діаграма прецедентів системи управління смарт контрактами

Детальний аналіз діаграми прецедентів дозволяє ідентифікувати ключових суб'єктів системи, серед яких виділено такі ролі: «Користувач», «Система блокчейну», «Власник контракту» та «Адміністратор». Взаємодія зазначених акторів із платформою реалізується через розгалужений функціональний

інструментарій. Особливої уваги заслуговує механізм верифікації транзакцій, який базується на застосуванні криптографічного стандарту хешування SHA-256. Впровадження цього алгоритму забезпечує фундаментальні властивості системи – цілісність та автентичність інформаційних потоків, що є пріоритетним завданням децентралізованих технологій у контексті захисту даних. Підсумовуючи, архітектура системи пропонує користувачеві вичерпний набір функцій для управління цифровими активами та контрактами.

Наступним етапом моделювання є побудова діаграми послідовності – однієї з базових структурних діаграм мови UML. Вона призначена для детермінації хронологічного порядку операцій та взаємодій у межах системи. Дана діаграма візуалізує процеси обробки подій у формі впорядкованих кроків, що можуть включати логічні розгалуження, ітераційні цикли, паралельне виконання потоків та фіксацію кінцевих станів. Основним призначенням діаграми активності є графічна репрезентація алгоритмів, що сприяє глибинному розумінню логіки функціонування програмного забезпечення та дозволяє здійснювати ефективний аналіз і оптимізацію бізнес-процесів. Деталізована структура діаграми активності представлена на рис. 2.2.

Процес взаємодії розпочинається з ініціації «Користувачем» відповідного запиту до «Системи», що може стосуватися як генерації нового смарт-контракту, так і виконання інших функціональних операцій. Отримавши запит, «Система» здійснює його аналіз для детермінації типу необхідної транзакції. У разі, якщо процедура передбачає створення нового контракту, «Система» надсилає інформаційне повідомлення «Власнику контракту» з метою верифікації та отримання дозволу на розгортання програмного коду.

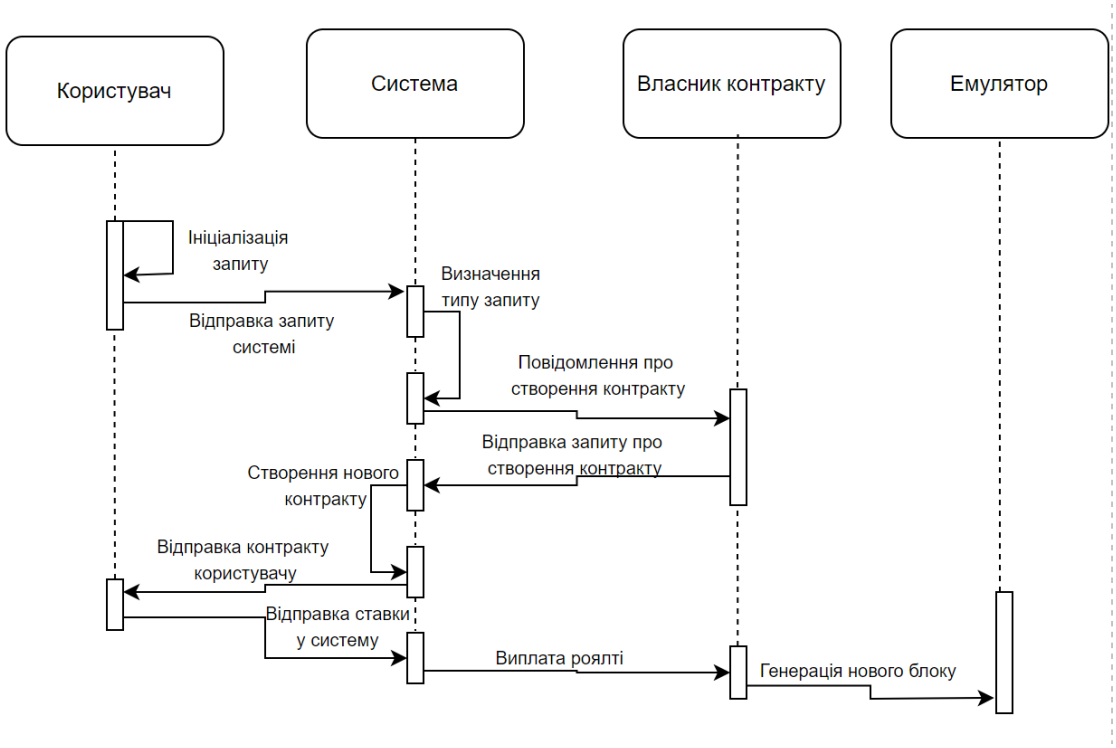

Рис.2.2 Діаграма послідовності системи

Після отримання відповідного сповіщення, «Власник контракту» підтверджує легітимність операції, відправляючи зворотний запит. На основі цього «Система» ініціалізує новий контракт і передає його «Користувачу», надаючи останньому можливість здійснювати подальші транзакції або взаємодіяти з логікою контракту. У ситуаціях, що передбачають фінансові операції (наприклад, здійснення ставки або проведення оплати), «Система» опрацьовує отримані дані та, відповідно до закладених алгоритмів, автоматично виконує нарахування роялті на користь «Власника контракту».

Фінальний етап життєвого циклу операції реалізується через дії «Емулятора», який ініціює генерацію нового блоку. Цей процес фіксує всі внесені зміни, транзакційні записи та системні оновлення, ініційовані «Користувачем». Процедура формування нового блоку є критично важливою, оскільки вона гарантує цілісність, незмінність та високий рівень захищеності даних у межах блокчейн-інфраструктури.

Графічна інтерпретація зазначених процесів наведена на діаграмі активності (рис. 2.3), яка за своєю структурою нагадує блок-схему алгоритму розв’язку задачі. Вона візуалізує динамічну зміну станів системи, де кожен стан

корелює з виконанням конкретної технологічної операції. Перехід до наступного етапу можливий виключно за умови успішного завершення всіх процесів у поточному стані, що забезпечує сувору логічну послідовність функціонування платформи.

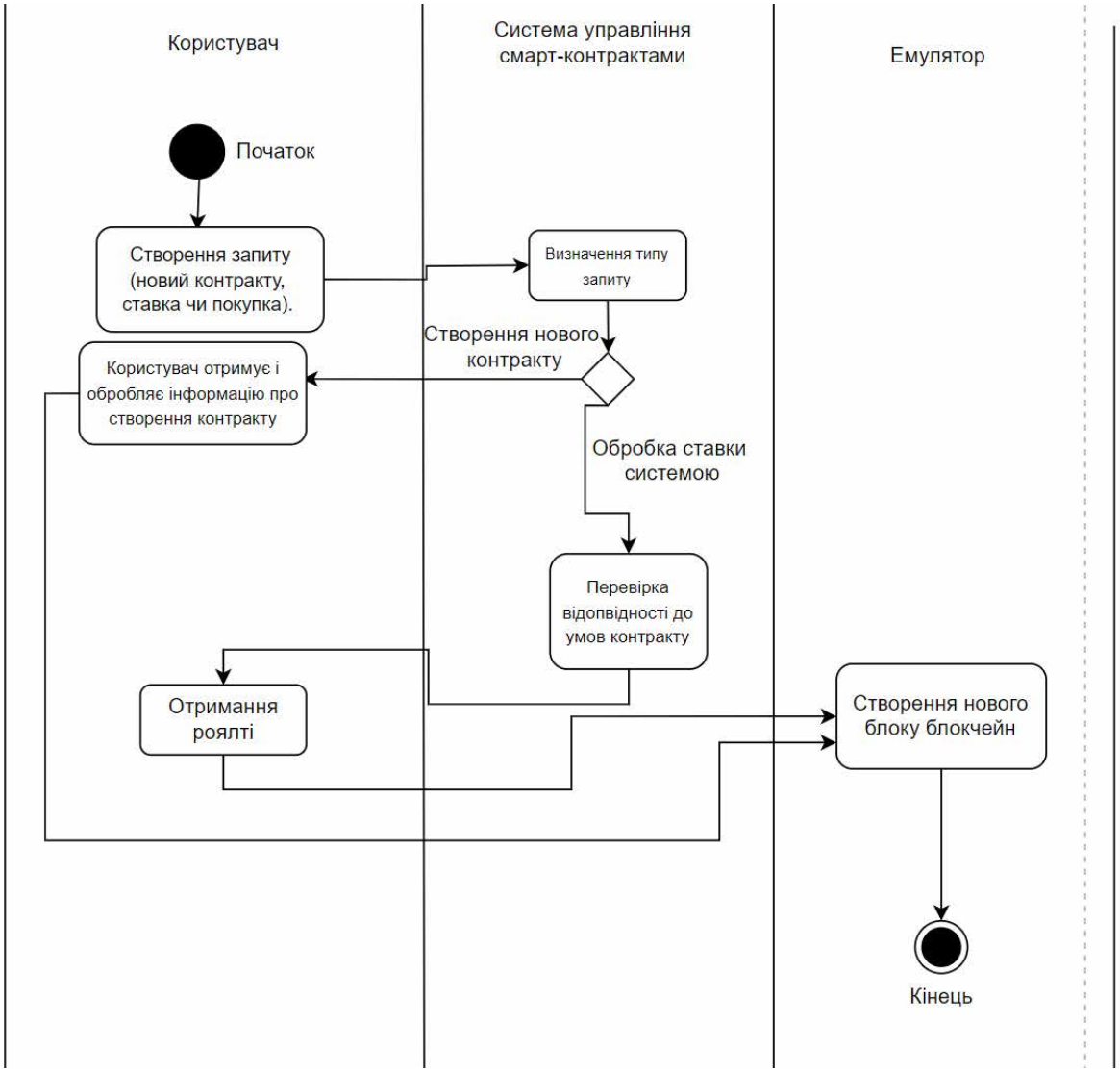

Рис. 2.3 Діаграма активності системи

Спочатку користувач надсилає запит на виконання певної операції, наприклад створення нового контракту, здійснення ставки або покупки. Цей запит передається до системи управління смарт-контрактами, де визначається тип необхідної дії.

Якщо запит стосується створення нового контракту, система запускає процес його формування. Після успішного завершення користувач отримує повідомлення та інформацію про створений контракт. Якщо ж запит пов’язаний

зі ставкою або покупкою, система переходить до етапу обробки цієї операції. У процесі виконання перевіряється відповідність дій умовам смарт-контракту.

Після завершення перевірки та виконання необхідних операцій користувач може отримати роялті, якщо це передбачено умовами контракту. Усі виконані дії фіксуються у новому блоці блокчейну, який створюється емулятором системи. Після генерації нового блоку процес завершується, а всі зміни надійно зберігаються в блокчейні, що забезпечує цілісність та незмінність даних.

Об’єктно-орієнтований підхід зосереджується на побудові системи через об’єкти, які представляють окремі елементи з власними даними та функціями. Кожен об’єкт є самостійною сутністю, яка поєднує стан (властивості) та поведінку (методи). Завдяки цьому система може відображати складні взаємозв’язки між компонентами та моделювати як реальні об’єкти, так і абстрактні поняття.

Такий підхід дозволяє створювати гнучкі, модульні та масштабовані системи, які легко змінювати та розширювати відповідно до потреб. Важливу роль у ньому відіграють зв’язки між об’єктами, зокрема асоціація, агрегація, композиція та наслідування. Наслідування дає змогу новим класам отримувати властивості та методи базових класів, що зменшує дублювання коду та спрощує повторне використання функціоналу. Агрегація та композиція описують способи об’єднання об’єктів у складніші структури.

Основними принципами об’єктно-орієнтованого підходу є інкапсуляція, наслідування та поліморфізм. Інкапсуляція приховує внутрішню реалізацію об’єкта та обмежує прямий доступ до його даних, що дозволяє змінювати внутрішню логіку без впливу на інші частини системи. Поліморфізм забезпечує можливість використовувати спільні інтерфейси для різних об’єктів, які можуть по-різному реалізовувати однакові дії.

Діаграма класів є одним із типів статичних діаграм UML і використовується для відображення структури системи. Вона показує класи, їхні атрибути, методи та зв’язки між ними. Така діаграма допомагає зрозуміти взаємодію компонентів системи, їхню ієрархію та залежності.

Кожен клас на діаграмі подається у вигляді прямокутника, поділеного на три частини. У верхній частині вказується назва класу, у середній – його атрибути або властивості, а в нижній – методи чи операції, які виконує клас. Між класами можуть встановлюватися різні типи зв’язків:

- наслідування – позначається суцільною лінією зі стрілкою від похідного класу до базового та показує успадкування властивостей і поведінки;
- асоціація – зображується простою лінією між класами та вказує на взаємодію або залежність між ними;
- залежність – позначається пунктирною лінією зі стрілкою та означає тимчасовий або непрямий зв’язок між класами;
- агрегація та композиція – демонструють відношення типу «ціле–частина». Агрегація позначається відкритим ромбом і означає слабший зв’язок між об’єктами, тоді як композиція зображується зафарбованим ромбом і показує, що частина не може існувати окремо від цілого.

Діаграми класів допомагають отримати чітке уявлення про структуру системи та взаємозв’язки між її елементами. Вони спрощують процес проєктування, розробки та підтримки програмного забезпечення, а також покращують комунікацію між учасниками розробки. Особливо важливими такі діаграми є в об’єктно-орієнтованому програмуванні, оскільки вони забезпечують наочне представлення взаємодії класів та сприяють створенню зрозумілого й організованого коду.

Розглянемо діаграму класів детальніше на рис. 2.4.

Фундаментальним елементом архітектури є базовий клас Contract, що слугує шаблоном для всіх смарт-контрактів у межах проектованої системи. Він агрегує ключові атрибути, зокрема ідентифікаційну адресу контракту, адресу ініціатора (творця), часову мітку створення, посилання на поточну транзакцію та параметри активаційного платежу. Крім того, клас детермінує методи ініціації транзакцій та передачі інформаційних повідомлень. Спеціалізовані класи, такі як

SignumArt, SignumArt2, Bicho та Cryptoball, базуються на принципі успадкування, розширюючи базовий функціонал Contract специфічною логікою.

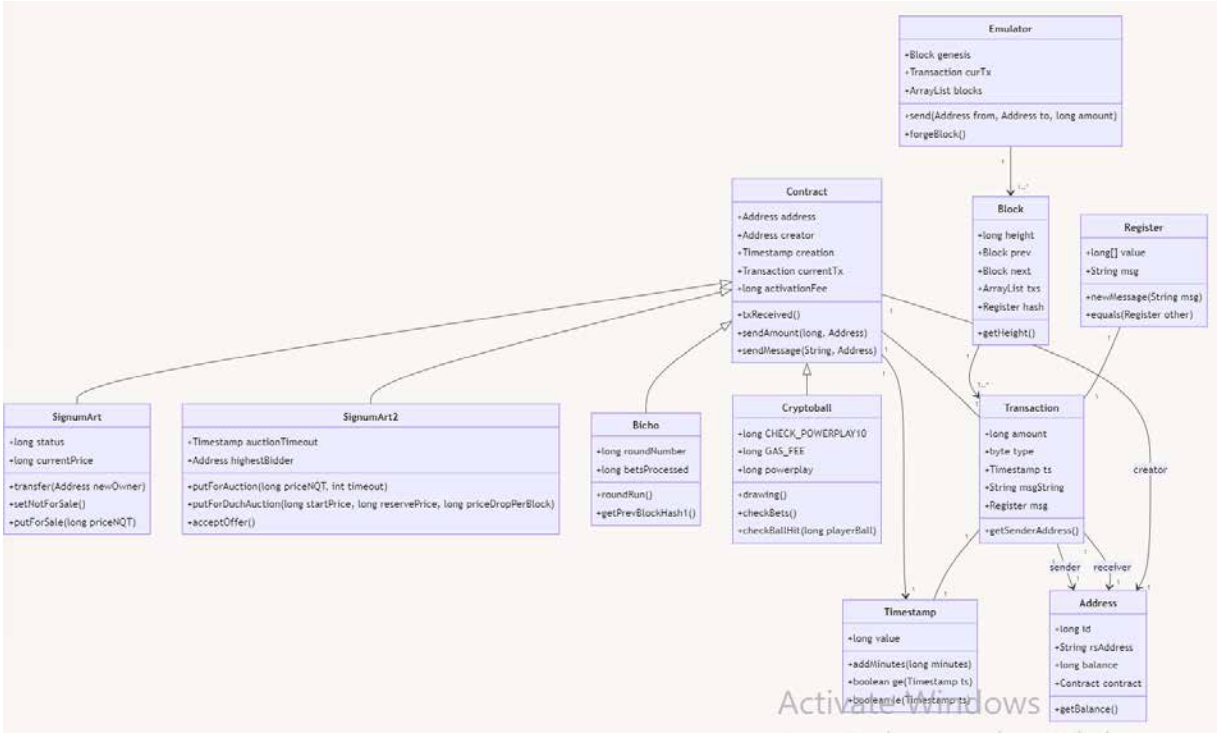

Рис. 2.4 Діаграма класів програмної системи

Клас SignumArt орієнтований на оперування об'єктами NFT або іншими цифровими активами, забезпечуючи механізми реалізації та делегування прав власності. Його модифікація, SignumArt2, доповнює зазначений інструментарій алгоритмами проведення класичних та голландських аукціонів. Ігрова логіка та лотерейні механізми реалізовані у класах Bicho (адміністрування ігрових раундів та облік ставок) і Cryptoball (генерація псевдовипадкових чисел, обробка ставок та ідентифікація виграшних комбінацій).

Інфраструктурний компонент Emulator забезпечує імітацію ключових функцій розподіленого реєстру: формування блоків, верифікацію транзакцій та координацію взаємодії між адресами системи. Структурна одиниця блокчейну моделюється класом Block, який інкапсулює дані про висоту блоку, його унікальний хеш та реєстр інклюдованих транзакцій. Клас Transaction, у свою чергу, детермінує параметри окремої операції (реквізити сторін, суму, тип транзакції, часову мітку) та має логічну прив'язку до відповідного блоку.

Допоміжний інструментарій системи включає такі класи:

- Register: призначений для архівації повідомлень та верифікації даних шляхом генерації та компаративного аналізу хеш-значень.
- Timestamp: забезпечує часову детермінацію процесів, дозволяючи контролювати терміни виконання операцій та здійснювати інкремент часових інтервалів.
- Address: репрезентує обліковий запис суб'єкта, зберігаючи інформацію про баланс, ідентифікатор та кореляцію зі смарт-контрактом, що є необхідним для здійснення операційних взаємодій.

Взаємозв'язки між компонентами системи побудовані на ієрархічній та функціональній залежності. Emulator інтегрується з Block та Transaction для адміністрування реєстру. Клас Contract корелює з Address, Transaction та Timestamp, що дозволяє забезпечити точну ідентифікацію контрагентів та хронологічну послідовність подій.

Узагальнюючи, спроектована об’єктно-орієнтована модель формує надійний каркас для управління цифровими активами та децентралізованими сервісами. Вона забезпечує структуроване середовище для безпечної взаємодії суб’єктів з емульованими компонентами блокчейну, реалізуючи повноцінну систему адміністрування смарт-контрактів.

2.2 Логічна та фізична модель даних

Логічна модель даних зазвичай подається у вигляді ER-діаграми (Entity-Relationship Diagram), яка використовується для візуального представлення структури майбутньої реляційної бази даних. Такий підхід дозволяє розробникам краще зрозуміти організацію даних у системі, а також виявити необхідність їх нормалізації для підвищення ефективності та узгодженості зберігання інформації.

Під час побудови ER-діаграми використовуються основні елементи: сутності (таблиці), атрибути та зв’язки між ними. Зв’язки поділяються на два основні типи — ідентифікуючі та неідентифікуючі. Ідентифікуючий зв’язок передбачає включення зовнішнього ключа до складу первинного ключа дочірньої таблиці, що підсилює зв’язок між сутностями та гарантує неможливість існування записів у дочірній таблиці без відповідного запису в батьківській. Натомість неідентифікуючий зв’язок не включає зовнішній ключ до первинного, тому дочірні записи можуть існувати незалежно від наявності відповідних записів у батьківській таблиці.

Загалом виділяють три рівні деталізації моделей даних. Концептуальна модель є найвищим рівнем абстракції та містить мінімальну кількість деталей, відображаючи загальну структуру системи. Логічна модель є більш деталізованою і включає опис сутностей, їх атрибутів та взаємозв’язків. Фізична модель, яка створюється на основі логічної, містить усі технічні параметри, необхідні для безпосередньої реалізації бази даних у конкретній СКБД.

У межах даної роботи ER-діаграма системи наведена на рисунку 2.5. Вона забезпечує структурований огляд сутностей, їх взаємозв’язків та логіки взаємодії в межах блокчейн-системи, що дозволяє краще зрозуміти організацію даних та забезпечити коректну і безпечну взаємодію між компонентами системи.

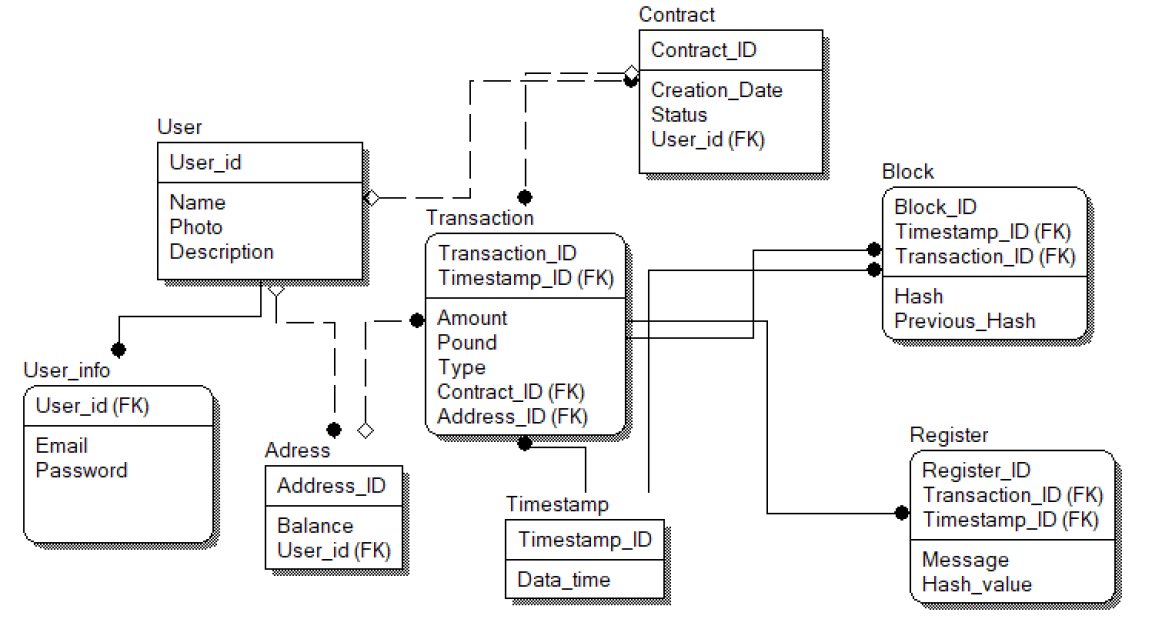

Рис.2.5 ER діаграма у нашій системі

Розглянемо сутності детальніше у таблиці 2.1

Таблиця 2.1

Сутності та їх атрибути

Сутність	Атрибути	Опис
User	User_id (PK), Name, Photo, Description	Ідентифікатор, ім'я, фото та опис.
User_info	User_id (FK), Email, Password	Зберігає додаткову інформацію про користувача, включаючи email і пароль, пов'язана з User.
Contract	Contract_ID (PK), Creation_Date, Status, User_id	Включає дату створення, статус та зв'язок із користувачем User.
Transaction	Transaction_ID (PK), Timestamp_ID (FK), Amount, Pound, Type, Contract_ID (FK),	Відображає окрему транзакцію, включає суму, тип, зв'язки з контрактом та адресою.
Address	Address_ID (PK), Balance, User_id (FK)	Представляє адресу в блокчейні, зберігає баланс і зв'язок із користувачем User.
Timestamp	Timestamp_ID (PK), Data_time	Зберігає інформацію про дату та час події.
Block	Block_ID (PK), Timestamp_ID (FK), Transaction_ID (FK), Hash, Previous_Hash	Блок у блокчейні, включає хеш, попередній, та зв'язки з транзакціями.
Register	Register_ID (PK), Transaction_ID (FK), Timestamp_ID (FK), Message, Hash_value	Зберігає повідомлення та хеш-значення, пов'язані з транзакціями, включає зв'язки з Transaction.

Фізична модель даних показує, як логічна модель реалізується в конкретній системі керування базами даних (СКБД). Вона описує, яким чином дані будуть зберігатися на фізичному рівні для забезпечення стабільної та швидкої роботи системи. У цій моделі визначаються всі технічні аспекти: типи даних, ключі, індекси та обмеження, які допомагають підтримувати цілісність даних і високу продуктивність бази даних.

Основою фізичної моделі є таблиці, які представляють головні сутності системи. Наприклад, у системі керування смарт-контрактами можуть використовуватися таблиці User, Transaction, Contract та Address. Кожна таблиця містить набір стовпців, що відповідають характеристикам певної сутності. Для кожного стовпця задається відповідний тип даних. Наприклад, таблиця User

може містити поля user_id типу INTEGER, name типу VARCHAR і photo типу BLOB для зберігання зображень. Це забезпечує впорядковане та зручне зберігання інформації.

Важливу роль у фізичній моделі відіграють ключі та обмеження. Первинний ключ забезпечує унікальність кожного запису в таблиці, наприклад user_id у таблиці User або transaction_id у таблиці Transaction. Для зв’язку між таблицями використовуються зовнішні ключі. Наприклад, поле contract_id у таблиці Transaction може посилатися на таблицю Contract, що дозволяє пов’язувати транзакції з відповідними контрактами. Такі зв’язки підтримують цілісність даних і дають змогу виконувати складні запити з кількох таблиць одночасно.

Для підвищення швидкості роботи запитів у фізичній моделі використовуються індекси. Їх зазвичай створюють для полів, до яких система звертається найчастіше. Наприклад, індекси можуть бути додані до полів transaction_id або timestamp у таблиці Transaction, що значно прискорює пошук та обробку даних.

Також у базах даних можуть застосовуватися нормалізація та денормалізація. Нормалізація допомагає уникнути дублювання інформації шляхом розподілу даних між окремими таблицями. Наприклад, інформація про користувача (User_info) може зберігатися окремо від основної таблиці User. Денормалізація, навпаки, використовується тоді, коли пріоритетом є швидкість роботи системи, навіть якщо це призводить до певного дублювання даних.

Фізичну модель даних для нашої системи наведено на рис. 2.6.

Таким чином, фізична модель є технічною інструкцією для створення структури бази даних. Вона гарантує, що дані будуть зберігатися ефективно, з можливістю швидкого доступу до них і забезпеченням цілісності. Така модель є важливою основою для побудови системи, яка може працювати з великими обсягами даних та обслуговувати різноманітні запити користувачів.

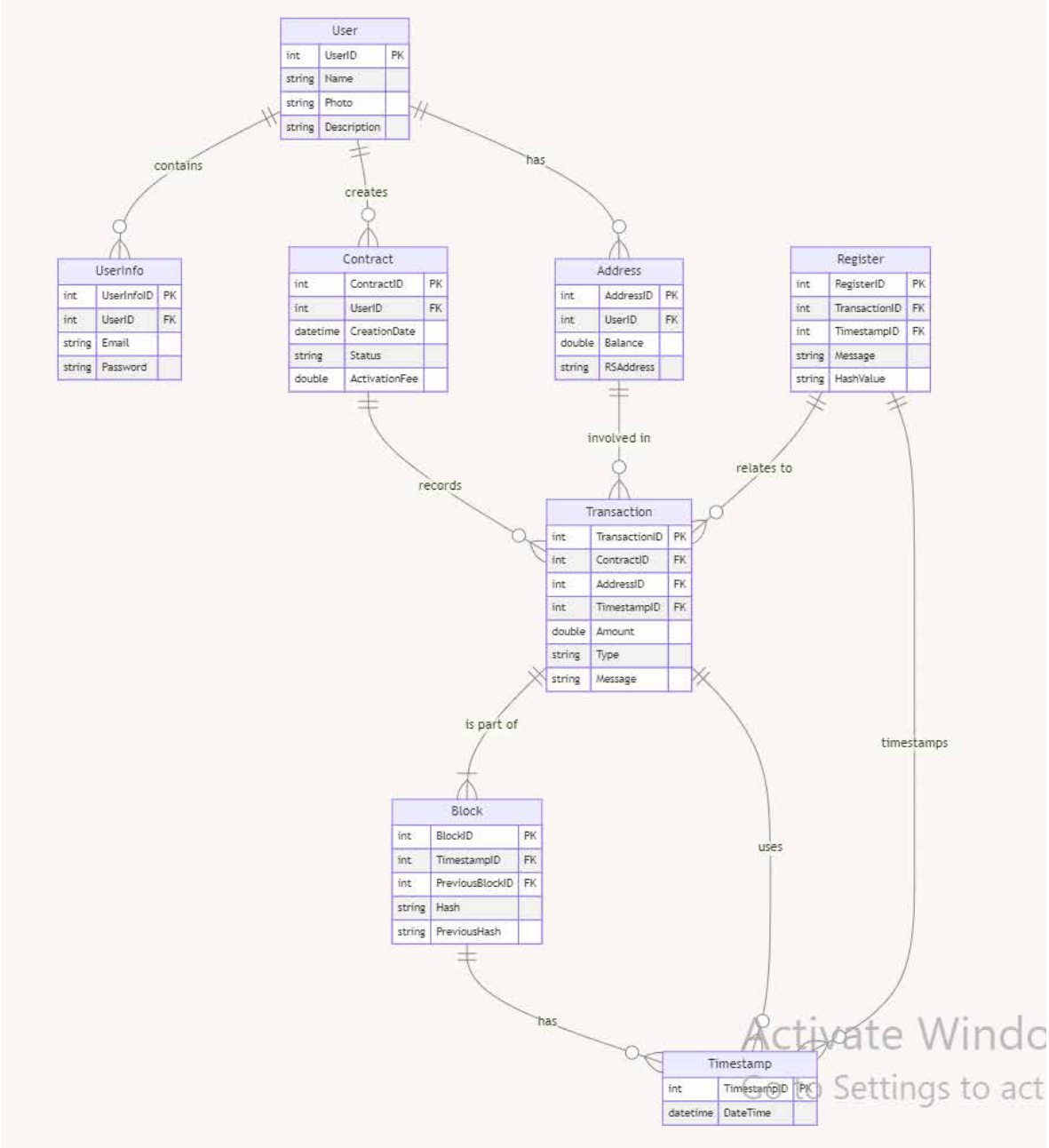

Рис.2.6 Фізична модель даних програмної системи

2.3 Опис архітектури системи на основі блокчейн

Архітектура системи управління смарт-контрактами, побудована на основі блокчейн-технологій, призначена для забезпечення надійного, безпечного та ефективного керування контрактами, їх виконанням, а також збереженням і обробкою даних. У її основі лежить децентралізований підхід, який дозволяє досягти високого рівня безпеки, масштабованості та відмовостійкості системи.

Компоненти архітектури функціонують на різних рівнях і тісно взаємодіють між собою, при цьому кожен рівень виконує окремі функціональні завдання в межах загальної системи. Детальніше структуру архітектури системи наведено на рисунку 2.7.

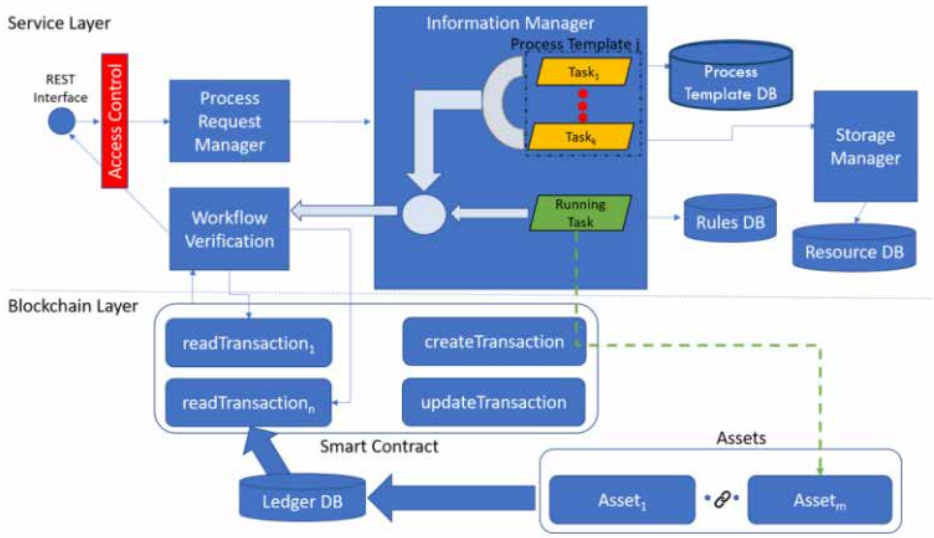

Рис.2.7 Архітектура системи на основі блокчейн

Архітектура системи складається з двох основних рівнів — сервісного та блокчейн-рівня, які спільно забезпечують управління, перевірку та зберігання даних у блокчейн-системі. У системі керування смарт-контрактами кожен із цих рівнів виконує важливі функції та забезпечує взаємодію користувачів із блокчейн-інфраструктурою.

На сервісному рівні обробка запитів користувачів здійснюється через менеджер обробки запитів, який працює за допомогою REST-інтерфейсу. Система контролю доступу перевіряє права користувачів і дозволяє працювати лише авторизованим користувачам. Після перевірки запити проходять етап валідації робочого процесу та передаються до менеджера інформації.

Менеджер інформації відповідає за керування процесами та взаємодіє з базою шаблонів процесів, де зберігаються шаблони для різних типів завдань. Ці шаблони визначають логіку та послідовність дій, які виконує смарт-контракт. Для роботи з даними використовується менеджер зберігання даних, який взаємодіє з базою правил і базою ресурсів. У цих базах містяться необхідні правила та ресурси для виконання контрактів. Усі завдання постійно

контролюються системою, щоб забезпечити правильне виконання процесів відповідно до заданої логіки.

Блокчейн-рівень відповідає за виконання основних операцій зі смарт-контрактами та транзакціями. На цьому рівні всі операції зі створення, оновлення та читання даних записуються в Ledger DB. Система підтримує різні типи транзакцій, зокрема createTransaction, updateTransaction та readTransaction. Вони взаємодіють із активами системи, які можуть представляти контракти, користувачів або інші об’єкти, що зберігаються в блокчейні. Смарт-контракти забезпечують запис історії змін і станів активів у Ledger DB.

У такій архітектурі смарт-контракти автоматизують перевірку та виконання операцій, що дозволяє забезпечити надійність, безпеку та цілісність даних. Ledger DB зберігає незмінну історію всіх транзакцій, яку можна перевірити в будь-який момент. Це створює захищену систему управління смарт-контрактами, у якій користувачі можуть безпечно взаємодіяти з блокчейн-активами, а всі виконані дії фіксуються та зберігаються.

Така архітектура особливо ефективна для систем, де важливими є прозорість, захист даних та автоматизація процесів у блокчейн-середовищі.

Діаграма компонентів використовується для моделювання фізичної структури системи та демонструє взаємозв’язки між її компонентами. У цьому випадку компонент розглядається як окрема фізична частина системи, яка забезпечує реалізацію її функцій і підтримує роботу всієї системи загалом.

Розглянемо детальніше діаграму компонентів для нашої системи на рис. 2.8.

Основним компонентом системи є файл SmartContract.java, який відповідає за реалізацію логіки роботи з блокчейном. Він виконує ключові операції, пов’язані зі смарт-контрактами, зокрема їх запуск та обробку транзакцій.

Компонент AuthService.java забезпечує автентифікацію користувачів і контролює безпечний доступ до системи. Для цього використовуються криптографічні механізми, реалізовані за допомогою зовнішніх бібліотек.

DataBaseService.java відповідає за взаємодію з базою даних. Цей компонент працює з файлом database_student.sql, забезпечуючи збереження, оновлення та отримання інформації, пов’язаної з користувачами та смарт-контрактами.

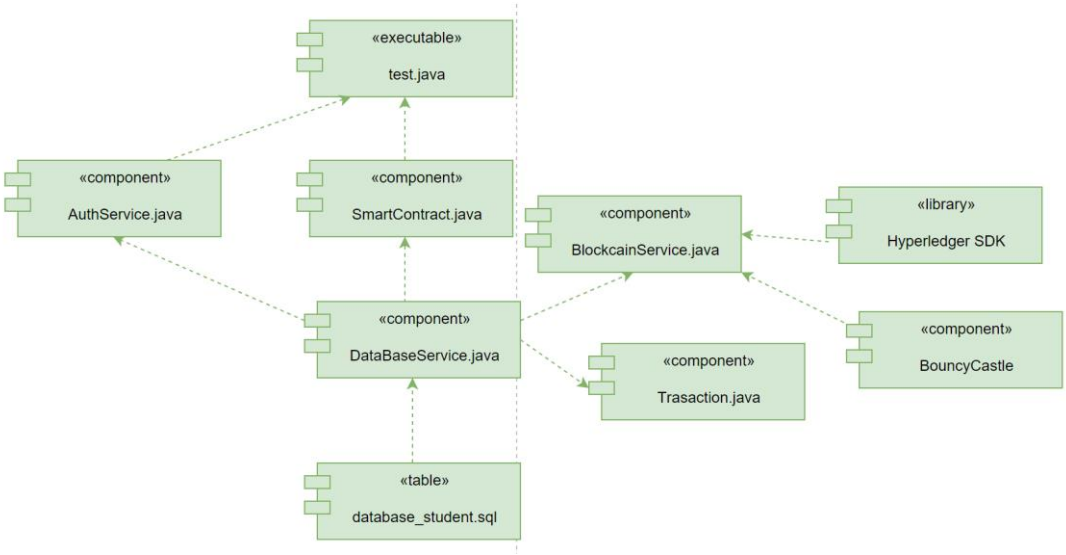

Рис.2.8 Діаграма компонентів нашої системи

BlockchainService.java виконує роль посередника між системою та блокчейн-рівнем. Він забезпечує можливість запису та зчитування даних із розподіленого реєстру. Для реалізації цієї взаємодії використовується Hyperledger SDK, який надає інструменти для розгортання, виклику та керування смарт-контрактами.

Компонент Transaction.java відповідає за обробку окремих транзакцій у системі. Його основне завдання – перевірка правильності формату транзакцій та відповідності встановленим правилам. Для забезпечення захисту даних і безпечної обробки транзакцій у системі також використовується криптографічна бібліотека BouncyCastle.

Нарешті, test.java виконує функцію виконуваного компонента, використовуючись для тестування функцій системи, імітуючи реальні сценарії для перевірки цілісності та ефективності процесів, пов’язаних з блокчейном. Зв’язки між компонентами вказують на залежності та взаємодії, які є необхідними для функціонування системи управління смарт-контрактами на основі блокчейну.

У контексті нашого завдання варто розглянути також діаграму пакетів системи. Розглянемо її детальніше на рис. 2.9.

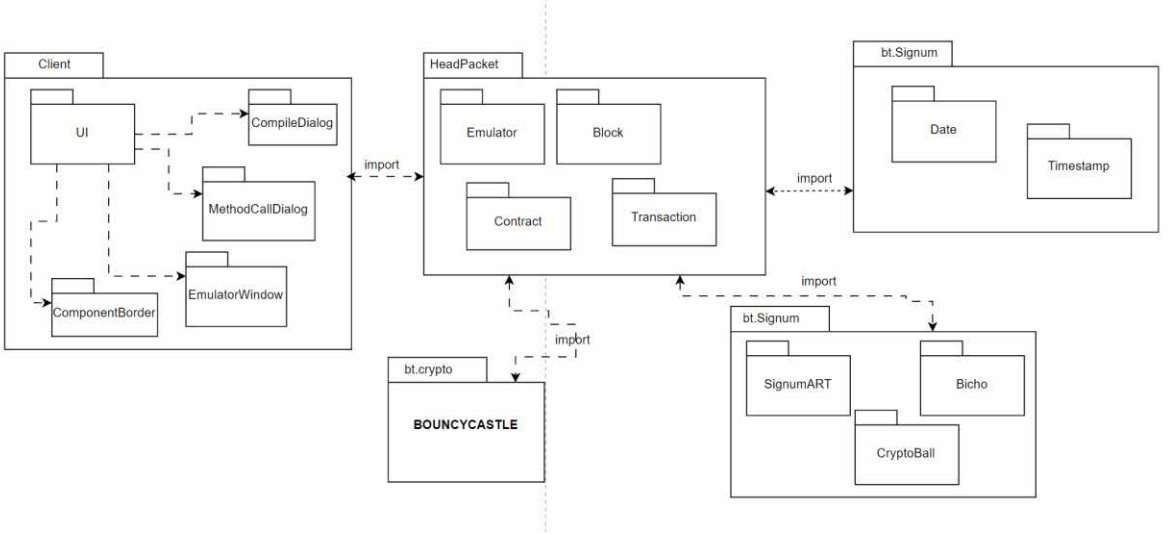

Рис.2.9 Діаграма пакетів програмної системи

Пакет Client реалізує інтерфейс користувача (UI) і містить набір компонентів для взаємодії з системою, зокрема UI, CompileDialog, MethodCallDialog, ComponentBorder та EmulatorWindow. Ці елементи забезпечують користувачу можливість працювати з емулятором, а також створювати, компілювати та запускати смарт-контракти.

Пакет HeadPacket виконує роль основного модуля, що відповідає за логіку блокчейн-середовища. До його складу входять ключові компоненти Emulator, Block, Contract і Transaction, які забезпечують моделювання блокчейну, керування блоками та транзакціями, а також взаємодію зі смарт-контрактами. Компонент Emulator дозволяє проводити тестування транзакцій і виконання контрактів у ізольованому середовищі без підключення до реальної блокчейн-мережі.

Пакет bt.Signum поділяється на два підпакети. Перший містить класи Date і Timestamp, які відповідають за роботу з часовими мітками, що є важливими для точного фіксування транзакцій у системі. Другий підпакет включає SignumART і CryptoBall, які представляють цифрові активи в блокчейні та дозволяють розширювати функціональність системи для роботи з токенами, NFT та іншими формами цифрової власності.

Пакет bt.crypto містить криптографічну бібліотеку BOUNCYCASTLE, яка забезпечує захист даних у блокчейн-системі. Вона використовується для реалізації криптографічних механізмів, таких як шифрування, створення цифрових підписів і хешування, що гарантує безпеку та цілісність інформації.

Взаємодія між пакетами організована таким чином: Client використовує функціонал HeadPacket для доступу до емулятора та роботи з блокчейн-системою. У свою чергу, HeadPacket залежить від bt.Signum і bt.crypto для обробки часових даних, криптографічного захисту та керування цифровими активами. Така архітектура забезпечує чітку модульність системи, де кожен пакет виконує окрему роль, що підвищує масштабованість і спрощує супровід програмного забезпечення.

2.4 Моделювання сценаріїв використання системи

Дослідження розпочинається з аналізу сценарію, у якому користувач ініціює процедуру розгортання смарт-контракту, здійснюючи при цьому конфігурацію широкого спектра вхідних параметрів. Цей процес є критично важливим етапом ініціалізації, оскільки саме на основі заданих значень формується логіка подальшого функціонування децентралізованого алгоритму.

Більш детально архітектоніку цього процесу та послідовність заповнення необхідних реквізитів відображено на рис. 2.10, де схематично зафіксовано етапи валідації введених даних та їхньої інтеграції у структуру створюваного контракту.

Рис.2.10 Сценарій ініціалізації смарт-контракту

Цей сценарій наочно показує обмежений функціонал системи, у вигляді створення смарт-контракту користувачем.

Далі розглянемо наступний сценарій який відповідає здійсненню транкзакцій у системі. Цей процес є дуже важливим у процесі виконання завдання кваліфікаційної роботи. Детальніше можемо побачити на рис. 2.11.

Рис. 2.11 Сценарій здійснення транкзакцій у системі

Варто акцентувати увагу на тому, що попереднє моделювання сценаріїв взаємодії є фундаментом для подальшої програмної реалізації системи, оскільки воно дозволяє візуалізувати архітектурні сутності та детермінувати характер зв’язків між ними. У процесі ініціалізації транзакції користувачем алгоритм системи здійснює верифікацію доступного балансу коштів, після чого забезпечує фіксацію операції у структурі блоку. Саме завдяки імплементації блокчейн-технологій досягається транспарентність процесів та гарантується високий рівень захищеності інформаційних активів.

Наступним етапом аналізу є розгляд сценарію, що стосується адміністрування та управління NFT (невзаємозамінними токенами). Аналогічно до попередніх етапів дослідження, детальна графічна інтерпретація та логіка виконання даної операції представлені на рис. 2.12.

Рис.2.12 Сценарій управління NFT

Користувач може створити і відповідно управляти цифровими активами, потім є можливість висталення на аукціон. Система обробляє цю транкзакцію і проводить збереження кожної зміни власності, що забезпечує прозорість. Ще два сценарії можемо побачити у таблиці 2.1.

Таблиця 2.1

Сценарії використання системи

Сценарій	Опис
Перевірка та аудит	Користувачі мають доступ до перевірки транзакцій, що забезпечує прозорість та контроль, відповідно вони можуть генерувати звіти про свої фінансові операції або статус конкретного контракту, де звіти базуються на даних, збережених у блокчейні.
Тестування контрактів на емуляторі	Перед випуском контракту в реальну блокчейн-мережу, користувачі можуть протестувати його на емуляторі, що дозволяє перевірити логіку виконання контракту та умови без фінансових ризиків, надаючи можливість поліпшити або налаштувати контракт перед реальним застосуванням.

Таблиця надає структурований огляд додаткових сценаріїв у системі, ці всі ключові взаємодіють дають розуміти про багатофункціональність системи.

3 РОЗРОБКА СИСТЕМИ УПРАВЛІННЯ СМАРТ-КОНТАКТАМИ

3.1 Технології та інструменти для реалізації системи

Платформа Signum була обрана для реалізації блокчейн-системи завдяки своїм унікальним характеристикам, що забезпечують високий рівень безпеки, децентралізації та економічної ефективності під час виконання смарт-контрактів. Додатковою перевагою є її орієнтація на енергоефективні рішення та стабільну роботу мережі навіть при значних навантаженнях.

Signum є однією з перших платформ, яка інтегрувала технологію Automated Transactions (AT), що дозволяє створювати самовиконувані смарт-контракти без необхідності постійного ручного контролю або втручання користувача. Завдяки цьому AT-контракти виконуються без додаткових зовнішніх сервісів чи надмірних обчислювальних витрат, що підвищує їхню надійність, передбачуваність і економічну доцільність. Ключовою перевагою Signum є також її децентралізована архітектура, яка забезпечує прозорість операцій і захист від несанкціонованого втручання або підробки даних.

Система Proof of Capacity (PoC), що використовується в Signum, базується на використанні вільного дискового простору користувачів для зберігання криптографічних даних, необхідних для майнінгу. Такий підхід значно знижує енергоспоживання у порівнянні з традиційними Proof of Work системами та робить мережу більш екологічно дружньою. Спрощену архітектуру принципу роботи платформи наведено на рисунку 3.1.

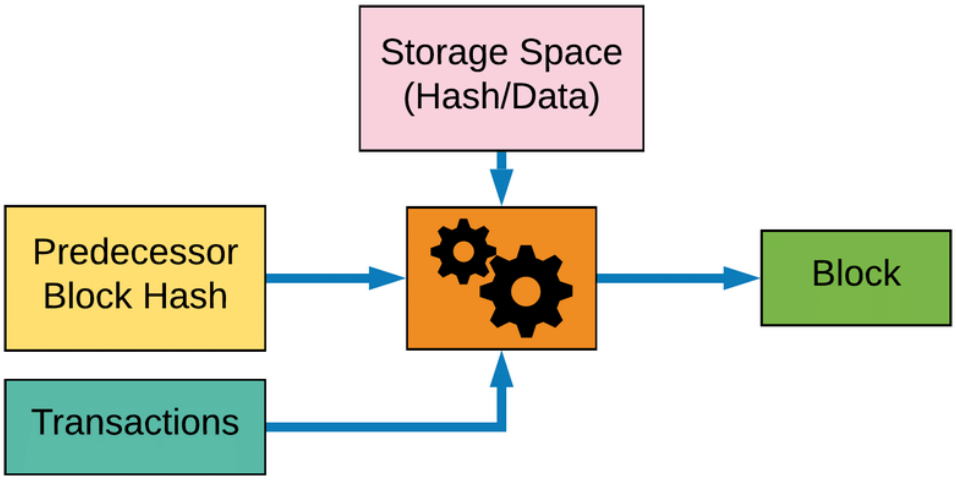

Рис.3.1 Спрощена архітекта PoC

Центральним елементом представленої моделі є процес формування блоку, що візуалізований за допомогою символічних механізмів (шестерень), які уособлюють обчислювальні операції, необхідні для інтеграції нової ланки в існуючий ланцюг. Технологічний цикл ініціюється отриманням хеш-суми попереднього блоку, що виконує роль криптографічного посилання. Саме цей ідентифікатор забезпечує детермінований зв'язок між блоками, що є фундаментальною умовою дотримання принципів цілісності та іммутабельності (незмінності) бази даних блокчейну. На вхідний етап створення блоку також надходять транзакції – агреговані дані про операції, які підлягають верифікації та подальшій архівації.

Результатом опрацювання хеша попередньої ланки та масиву транзакцій є генерація нового блоку. Він інкапсулює унікальний хеш, сформований на основі внутрішніх даних та посилання на попередній елемент, що у майбутньому стане базисом для наступної ітерації в ланцюзі. Структура діаграми передбачає спеціалізований рівень зберігання, що акцентує увагу на одночасній фіксації як криптографічних доказів, так і безпосередньо транзакційних даних у кожному сегменті реєстру. Такий підхід забезпечує високий рівень безпеки та надмірності даних, властивий розподіленим книгам.

Окремим аспектом дослідження є вибір платформи Signum, економічна ефективність якої зумовлена використанням унікального алгоритму консенсусу

Proof of Capacity (PoC). Дана технологія дозволяє учасникам мережі здійснювати підтримку інфраструктури (майнінг) без необхідності експлуатації високовартісного спеціалізованого обладнання та надмірного енергоспоживання. Це позиціонує Signum як екологічно стійку альтернативу традиційним блокчейн-платформам. Порівняльний аналіз енергетичної ефективності та операційних витрат наведено на графіку (рис. 3.2).

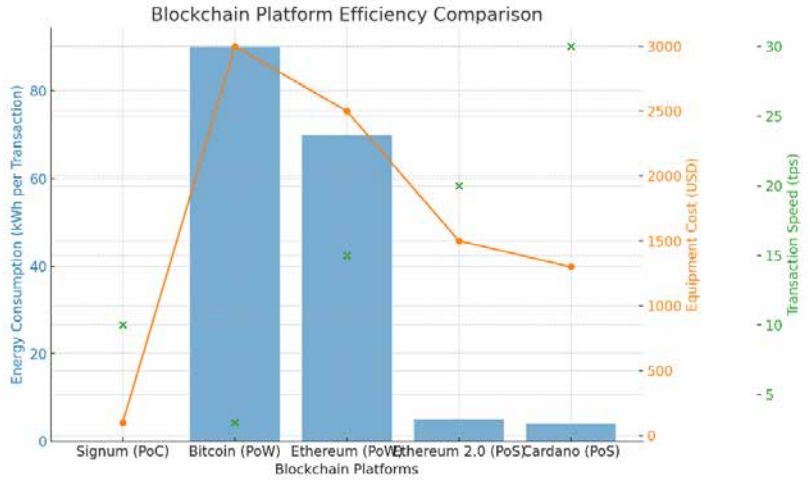

Рис. 3.2 Порівняння ефективності блокчейн-платформ

Платформа Signum, яка працює на основі алгоритму Proof of Capacity, характеризується дуже низьким рівнем енергоспоживання — приблизно 0,1 кВт·год на одну транзакцію. Це суттєво менше порівняно з платформами, що використовують алгоритм Proof of Work, такими як Bitcoin та Ethereum, де витрати енергії становлять близько 90 і 70 кВт·год відповідно. Крім того, для роботи Signum потрібне значно дешевше обладнання — орієнтовно 100 доларів, тоді як для Bitcoin вартість обладнання може досягати 3000 доларів. Хоча швидкість обробки транзакцій у Signum становить близько 10 транзакцій за секунду і є нижчою, ніж у деяких сучасних платформ, наприклад Cardano, цього достатньо для більшості практичних застосувань.

Отже, вибір Signum як основи для розроблюваної системи пояснюється її економічністю та можливістю ефективної роботи зі смарт-контрактами. Платформа добре підходить для організації аукціонів і виконання фінансових операцій. На відміну від Ethereum, де комісії можуть значно збільшуватися залежно від навантаження мережі, Signum забезпечує стабільні та невисокі

транзакційні витрати, що є важливим для систем із великою кількістю активних користувачів і регулярними операціями.

Для коректної роботи з блокчейн-платформою необхідно використовувати Java 17 або новішу версію. Це забезпечує стабільну інтеграцію Java з Gradle та підтримку сучасних бібліотек і технологій, що використовуються в блокчейн-розробці. Java 17 також містить оновлені механізми безпеки та нові функціональні можливості, необхідні для надійної взаємодії зі смарт-контрактами та стабільної роботи системи.

Web3j є легкою Java-бібліотекою для роботи з блокчейном Ethereum. Вона дозволяє інтегрувати смарт-контракти в Java-застосунки, надсилати транзакції, отримувати дані з блокчейну та керувати криптогаманцями без потреби працювати з низькорівневими протоколами. Завдяки цьому Web3j значно спрощує процес створення блокчейн-застосунків на Java.

Бібліотека BouncyCastle забезпечує набір криптографічних API для Java та використовується для підвищення рівня безпеки системи. Вона підтримує алгоритми шифрування, хешування та перевірки цифрових підписів, що особливо важливо під час роботи з конфіденційними даними та фінансовими транзакціями. Використання BouncyCastle дозволяє гарантувати цілісність і захист даних у блокчейн-системі.

Gradle застосовується як інструмент для керування залежностями проєкту, автоматизації збірки, компіляції та тестування застосунку. Його використання спрощує процес розробки, дозволяючи автоматизувати перевірку залежностей і запуск тестів. Версія Gradle 8.5 забезпечує повну сумісність із Java 17, що допомагає уникнути технічних проблем під час розробки. Крім того, Gradle спрощує інтеграцію сторонніх бібліотек, зокрема Web3j та BouncyCastle. Фрагмент діаграми розгортання системи наведено на рис. 3.3.

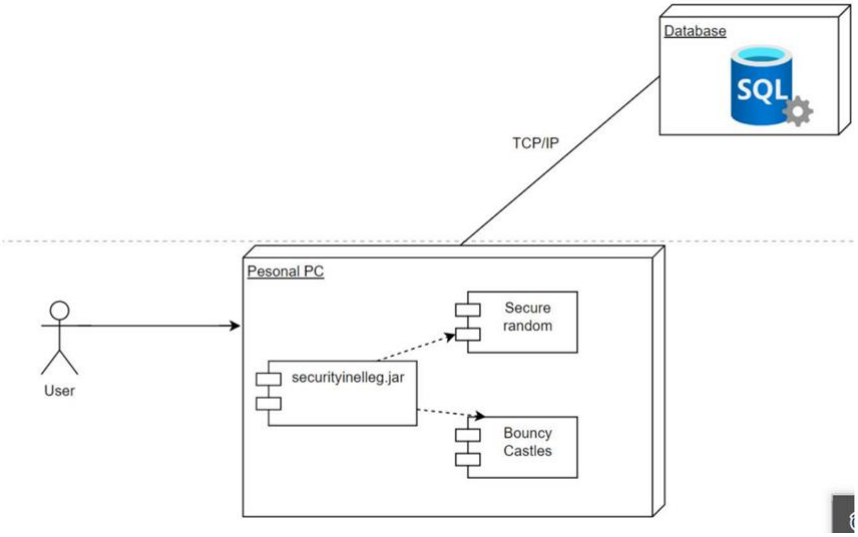

Рис. 3.3 Діаграма розгортання програмної системи

Для тестування та налагодження логіки смарт-контрактів у контрольованому середовищі в коді застосовуються класи Emulator та EmulatorWindow. Вони забезпечують можливість моделювання основних процесів блокчейну без необхідності фактичного розгортання контрактів у тестовій або основній мережі. Такий підхід дозволяє створити ізольоване середовище, у якому можна безпечно перевіряти різні сценарії роботи системи.

Завдяки використанню цих інструментів розробники мають змогу оперативно виявляти помилки в логіці контрактів, аналізувати їхню поведінку при різних типах транзакцій та перевіряти коректність виконання умов. Крім того, емуляція процесів дозволяє оцінити стабільність і передбачуваність роботи смарт-контрактів ще до їх розгортання у production-середовищі, що суттєво знижує ризики виникнення критичних помилок.

3.2 Криптографічні алгоритми, реалізація алгоритмів смарт контрактів

Основною бібліотекой яка використовувалась у нашій програмі є BouncyCastle для забезпечення криптографічних функцій, таких як хешування і підпис, що є дуже важливими аспектами для захисту транзакцій і валідації даних у блокчейні.

Першочергово розглянемо алгоритм хешування SHA-256 який є основним криптографічним алгоритмом, що використовується для генерації унікального хешу для блоків та транзакцій. Метод performSHA256_64 реалізує SHA-256 хешування, яке генерує хеш із 64-бітного значення. Розглянемо фрагмент реалізації на рис. 3.4.

```
protected long performSHA256_64(long input1, long input2) {
    Register input = new Register();
    input.value[0] = input1;
    input.value[1] = input2;

    Register ret = performSHA256_(input);
    return ret.getValue1();
}
```

Рис. 3.4 Фрагмент реалізації SHA-256

Даний метод приймає як вхідні параметри два числових значення великої розрядності, здійснює їх криптографічне перетворення та повертає результуючий хеш. Цей інструментарій відіграє ключову роль у забезпеченні цілісності структурних одиниць блоку та є фундаментом для верифікації транзакційних записів.

Стосовно алгоритму Proof of Capacity (PoC), важливо розмежувати рівні його імплементації. Він не інтегрований безпосередньо у програмний код смарт-контрактів, оскільки функціонує на базовому протокольному рівні мережі Signum. Концепція PoC базується на використанні попередньо обчислених даних (плотів), які верифікуються майнерами під час формування нових блоків. Таким чином, смарт-контракт делегує завдання досягнення консенсусу нижньому рівню мережевої інфраструктури.

Програмна реалізація зосереджена на функціональній логіці прикладних контрактів, таких як Bicho, Cryptoball та SignumArt. Ці класи оперують високорівневими функціями платформи: ініціацією транзакцій, генерацією хеш-ідентифікаторів та симуляцією блокчейн-середовища для проведення налагоджувального тестування. Вони використовують безпеку, яку гарантує PoC, не втручаючись у механіку підтвердження блоків.

Програмна реалізація та модульна структура

Архітектура системи базується на модульному підході, де кожен клас відповідає за специфічну бізнес-логіку:

- SignumArt: керує життєвим циклом цифрових активів та NFT.
- Bicho та Cryptoball: інкапсулюють алгоритми ігрових механік та розподілу виграшів.

Взаємодія між цими компонентами забезпечується через ієрархію методів, що дозволяє гнучко масштабувати систему. Деталізований фрагмент програмного коду, що ілюструє реалізацію цих методів, наведено на рис. 3.5.

```
public void putForAuction(int priceNQT, int timeout) {
    if (highestBidder == null && owner.equals(this.getCurrentTxSender())) {
        status = STATUS_FOR_AUCTION;
        auctionTimeout = getBlockTimestamp().addMinutes(timeout);
        currentPrice = priceNQT;
        sendMessage(status, owner.getId(), currentPrice,
auctionTimeout.getValue(), tracker);
    }
}
```

Рис. 3.5 Виставлення активу на аукціон

Він встановлює статус контракту на значення «в аукціоні» та фіксує часовий інтервал, протягом якого відбувається прийом ставок і визначається момент завершення аукціону. Це дозволяє чітко регламентувати часові межі роботи смарт-контракту та забезпечити коректність його виконання.

Також реалізація смарт-контракту Bicho передбачає комплексну логіку для обробки ставок учасників, автоматичного розрахунку результатів аукціону та визначення переможців. Окрім цього, реалізовано механізм автоматичного здійснення виплат, що гарантує прозорість і незмінність фінансових операцій без участі посередників. Завдяки цьому транзакції виконуються у повністю

децентралізованому режимі, що підвищує надійність і довіру до системи. Фрагмент коду, що ілюструє логіку здійснення виплат, наведено на рисунку 3.6.

```
private void pay() {
    amountToPlatform = currentPrice * platformFee / THOUSAND;
    amountToRoyalties = currentPrice * royaltiesFee / THOUSAND;

    totalPlatformFee += amountToPlatform;
    totalRoyaltiesFee += amountToRoyalties;
    totalTimesSold++;

    sendAmount(amountToRoyalties, royaltiesOwner);
    sendAmount(currentPrice - amountToPlatform - amountToRoyalties, owner);
}
```

Рис. 3.6 Логіка для ставок

Метод здійснює розподіл фінансових коштів між платформою, правовласником та власником активу відповідно до заздалегідь визначених правил смарт-контракту. Він забезпечує автоматизоване виконання виплат усім зацікавленим сторонам, що гарантує прозорість і коректність фінансових операцій без необхідності ручного втручання.

Крім того, у програмній реалізації використовуються класи Emulator та EmulatorWindow, які дозволяють проводити тестування смарт-контракту в середовищі, що імітує роботу блокчейну. Це дає змогу перевірити правильність логіки виконання контракту ще до його розгортання в реальній мережі, зменшуючи ймовірність помилок та підвищуючи надійність розробки.

Також у системі реалізовано алгоритми шифрування даних, які мають доволі складну та багаторівневу структуру. Для кращого розуміння архітектури програмного забезпечення детальніше розглянемо її на рисунку 3.7.

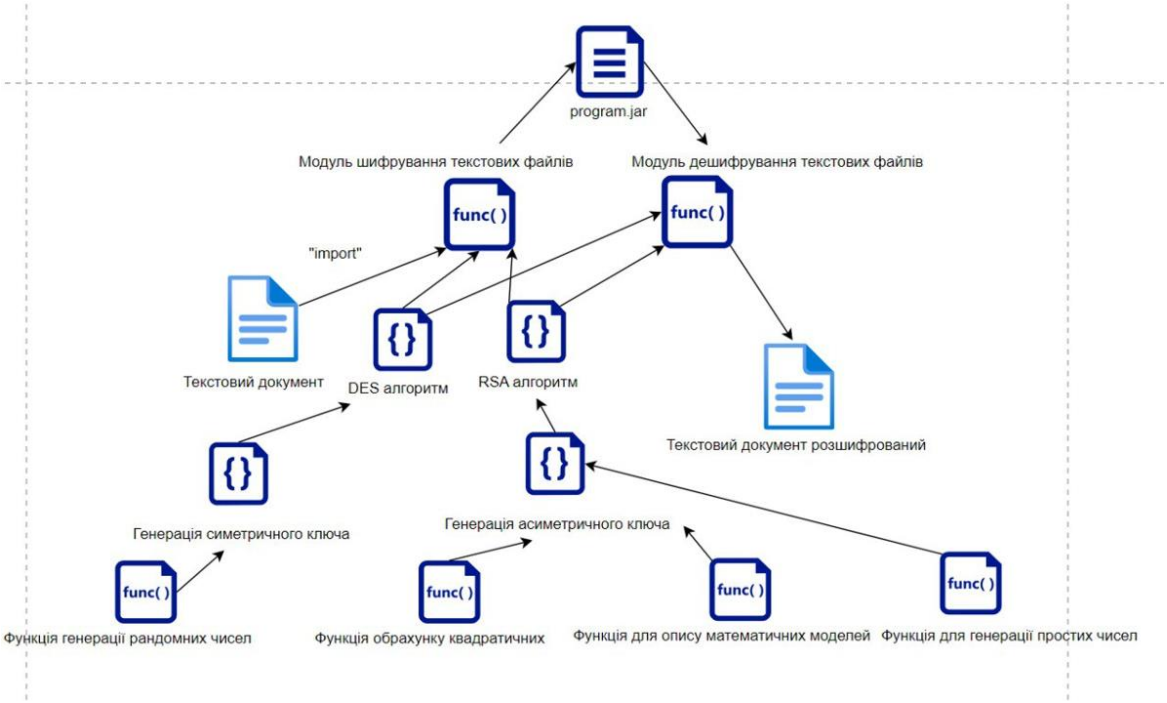

Рис.3.8 Структура коду шифрування інформації

Узагальнена схема алгоритму DES застосовується для шифрування блоків даних розміром 64 біти та базується на багаторазовому виконанні операцій підстановки, перестановки й перемішування бітів, які повторюються в декількох раундах із використанням ключа шифрування. Така структура забезпечує поступове ускладнення залежності між відкритим текстом і шифротекстом, підвищуючи криптографічну стійкість на рівні окремого блоку даних.

Алгоритм DES свого часу був широко поширеним і активно використовувався в різних системах захисту інформації, однак з розвитком обчислювальних потужностей та криптоаналітичних методів було виявлено його суттєві обмеження. Основною проблемою стала недостатня довжина ключа, що зробило можливим перебір варіантів і, відповідно, знизило рівень безпеки. Це призвело до поступового витіснення DES більш сучасними та стійкими алгоритмами шифрування, такими як AES.

Детальнішу структуру та принцип роботи алгоритму DES розглянемо на рисунку 3.9.

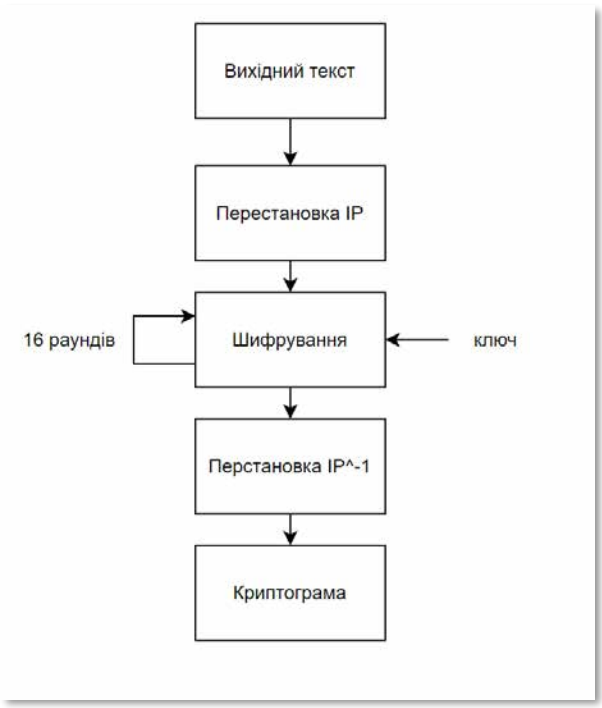

Рис.3.9 - Узагальнена схема шифру DES

Процедура шифрування за стандартом DES базується на ітераційному перетворенні 64-бітних блоків даних. Кожен етап алгоритму суворо регламентований специфікаціями стандарту, що включають фіксовані таблиці перестановок (IP та FP) та S-блоки заміни. Ці таблиці є фундаментальними константами, які забезпечують детермінованість та стійкість алгоритму до певних видів криптоаналізу.

Центральним елементом архітектури є мережа Фейстеля, яка складається з 16 ідентичних циклів (раундів). У кожному раунді блок даних розділяється на дві рівні частини, одна з яких піддається перетворенню за допомогою раундової функції F.

Процес у межах кожного раунду включає наступні кроки:

- Розширення (Expansion): 32-бітний напівблок розширюється до 48 біт для подальшого змішування з ключем.
- Накладання ключа: виконання операції XOR між розширеним блоком та 48-бітним раундовим підключем.
- S-перетворення (Substitution): стиснення даних назад до 32 біт за допомогою таблиць замін, що вносить нелінійність у шифр.

- Перестановка (Permutation): фінальне змішування бітів у межах напівблоку.

Графічне представлення структури окремого раунду та логіку взаємодії елементів функції F наведено на рис. 3.10. Ця циклічність дозволяє досягти ефекту лавини, де зміна одного біта вхідних даних призводить до кардинальної зміни всього шифротексту.

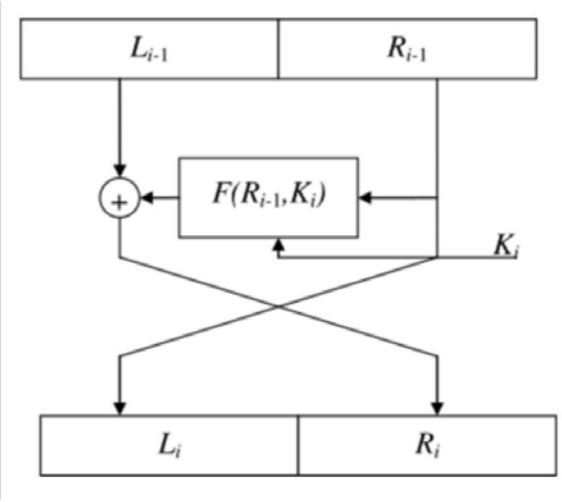

Рис.3.10 - Пряме перетворення за схемою Фейстеля

На етапі розширення даних правий півблок вхідного тексту збільшується з 32 до 48 біт за допомогою спеціальної функції розширення E. Ця функція реалізує перестановку та дублювання окремих бітів, у результаті чого формується розширена структура блоку з більшою довжиною. Таке перетворення необхідне для подальшого поєднання даних із раундовим ключем і підвищення криптографічної стійкості.

Після завершення операції розширення отриманий 48-бітний блок використовується у наступних етапах обчислень у межах одного раунду алгоритму шифрування DES. Детальну схему функції розширення наведено на рисунку 3.11.

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Рис.3.11 - Функція E- розширення блоку

Після етапу розширення отриманий 48-бітний блок даних піддається побітовій операції XOR (виключне АБО) з раундовим ключем, який формується відповідно до алгоритму генерації ключів і буде детально розглянутий у наступних підрозділах.

Далі виконується етап підстановки за допомогою S-блоків (S_i), у межах якого кожна 6-бітова частина вхідного блоку замінюється відповідним 4-бітовим значенням згідно з визначеними таблицями. У результаті такої трансформації з початкових 48 біт отримується 32-бітний вихідний блок. S-блоки є стандартизованими елементами алгоритму DES і представлені у вигляді таблиць розмірністю 4 рядки на 16 стовпців, що містять заздалегідь визначені значення підстановки.

Таким чином, у процесі виконання цих операцій відбувається зменшення розміру даних з 48 до 32 біт за рахунок комбінованого застосування XOR-операції та нелінійної підстановки через S-блоки, що є ключовим елементом криптографічної стійкості алгоритму DES.

3.3 Алгоритми обробки транзакцій та захисту даних.

Основна обробка транзакцій у програмі відбувається за допомогою методів, що відповідають за прийом, обробку та управління транзакціями. Наприклад, у класах Bicho, Cryptoball та SignumArt використовується метод

txReceived(), який обробляє транзакцію при її отриманні. Для прикладу розглянемо програмний код у класі SignumArt на рис. 3.12.

Програмна реалізація смарт-контрактів у системі демонструє високий рівень спеціалізації логіки обробки даних залежно від типу цифрового активу або сервісу. Кожен клас-спадкоємець базового контракту імплементує власні протоколи взаємодії, що дозволяє гнучко адаптувати систему під різні бізнес-моделі.

```
public void txReceived() {
    if (status == STATUS_FOR_SALE) {
       if (duchStartHeight > ZERO) {
          // Duch auction style, calculate the current price
          currentPrice = startPrice - (getBlockHeight() - duchStartHeight) *
priceDropPerBlock;
          if(currentPrice < reservePrice) {
             currentPrice = reservePrice;
          }
       }
       if (getCurrentTxAmount() >= currentPrice) {
          // Conditions match, let's execute the sale
          pay(); // pay the current owner
          owner = getCurrentTxSender(); // new owner
          status = STATUS_NOT_FOR_SALE;
          sendMessage(owner.getId(), getCurrentTxAmount(), trackNewOwner);

          cancelOfferIfPresent();
          return;
       }
    }
```

Рис.3.12 Реалізація txReceived()

У межах класу SignumArt ключову роль відіграє метод txReceived(). Він виступає диспетчером подій для операцій з NFT, автоматизуючи процес передачі прав власності. Алгоритм перевіряє вхідний фінансовий потік: при досягненні суми, що відповідає встановленій вартості активу (або переможній ставці на аукціоні), система ініціює зміну власника в реєстрі. Це гарантує атомарність угоди – передача токена відбувається виключно за умови повного виконання фінансових зобов'язань.

Логіка класу Bicho орієнтована на ігрові механіки. Тут транзакції акумулюються протягом раунду, після чого система звертається до хешу останнього сформованого блоку для детермінації переможного номера. Таке використання блокчейн-даних як джерела ентропії забезпечує прозорість

розіграшу, оскільки результат неможливо спрогнозувати або підробити до моменту закриття блоку.

Фундаментом безпеки всієї інфраструктури є методи криптографічного перетворення даних. Використання функцій performSHA256 та спеціалізованої performSHA256_64 дозволяє створити цифровий відбиток будь-якої операції. Зокрема, метод performSHA256_64 у базовому класі Contract забезпечує хешування парних 64-бітних значень, що критично важливо для верифікації зв’язків між блоками та унікальної ідентифікації транзакцій.

Ці криптографічні методи утворюють захисний бар'єр, який унеможливлює ретроспективну зміну даних або несанкціоновану модифікацію смарт-контрактів. Детальна схема взаємодії методів хешування з транзакційним рівнем системи представлена на рис. 3.13.

```
protected long performSHA256_64(long input1, long input2) {
    Register input = new Register();
    input.value[0] = input1;
    input.value[1] = input2;

    Register ret = performSHA256_(input);
    return ret.getValue1();
}
```

Рис.3.13 Алгоритм захисту даних

Ця функція застосовується у процесах, пов’язаних із визначенням результатів лотерей або аукціонів, де необхідно забезпечити випадковість і мінімізувати можливість зовнішнього втручання чи маніпуляцій. У нашій програмній реалізації також використовується підхід, заснований на хешах попередніх блоків, що дозволяє генерувати псевдовипадкові значення та підвищує стійкість системи до передбачуваності результатів.

Зокрема, у класі Bicho визначення виграшних номерів здійснюється на основі хешу попередніх блоків, що значно ускладнює можливість прогнозування результатів і підвищує рівень захисту від потенційних атак або маніпуляцій. Детальнішу реалізацію цього механізму наведено на рисунку 3.14.

```
private void roundRun() {
    hash1 = getPrevBlockHash1();
    hash = hash1;
    hashCounter = ONE;
    while (hashCounter < HASH_BLOCKS) {
        sleep(ONE);
        hash *= TWO; // зрушення на один біт
        hash += (getPrevBlockHash1() & ONE);
        hashCounter += ONE;
    }
    hash = performSHA256_64(hash, hash1);
}
```

Рис.3.14 Генерація випадкових чисел

Цей фрагмент коду генерує випадкове значення на основі хешу попереднього блоку, що суттєво ускладнює прогнозування наступних результатів. Такий підхід забезпечує більшу справедливість у випадкових процесах, зокрема в аукціонах або лотереях, і підвищує рівень захисту системи від можливих шахрайських дій.

Крім того, кожна транзакція, яка надходить до смарт-контракту, проходить перевірку на відповідність заданим умовам. Наприклад, у випадку аукціону система додатково контролює, чи відповідає сума транзакції встановленому мінімальному розміру ставки, що гарантує коректність участі користувачів у торгах.

3.4 Опис елементів інтерфейсу системи

У нашому проекті реалізовано 5 класів для реалізації UI програмної системи, що можемо побачити на рис. 3.15.

Рис.3.15 Класи системи які відповідають за інтерфейс користувача

Розглянемо опис кожного класу детальніше у таблиці 3.1.

Таблиця 3.1

Опис класів які відповідають за інтерфейс програмної системи

Клас	Опис
CompileDialog	Реалізує діалогове вікно для компіляції, яке може включати функціонал вибору файлів, перевірки синтаксису та запуску процесу компіляції з відображенням помилок.
ComponentBorder	Відповідає за відображення меж компонентів інтерфейсу, налаштовує зовнішній вигляд меж (колір, товщина, стиль).
EmulatorWindow	Основне вікно емулятора, яке надає доступ до інструментів тестування і налагодження смарт-контрактів, візуалізації транзакцій, запуску тестів і відображення результатів.
HintTextField	Реалізовує текстове поле з підказкою, що зникає при введенні даних, допомагає користувачам зрозуміти призначення полів.
MethodCallDialog	Створює діалогове вікно для виклику методів або функцій, дозволяє вибирати методи, вводити параметри та переглядати результати виконання.

Перейдемо до огляду самого інтерфейсу. Розглянемо рисунок 3.16.

Графічна оболонка програмного комплексу «BlockTalk Emulator» спроектована за принципом модульності, що дозволяє забезпечити інтуїтивно зрозумілу навігацію та оперативний доступ до ключових інструментів блокчейн-моделювання. Архітектура інтерфейсу чітко розділена на три функціональні зони, кожна з яких відповідає за окремий аспект роботи з децентралізованим середовищем.

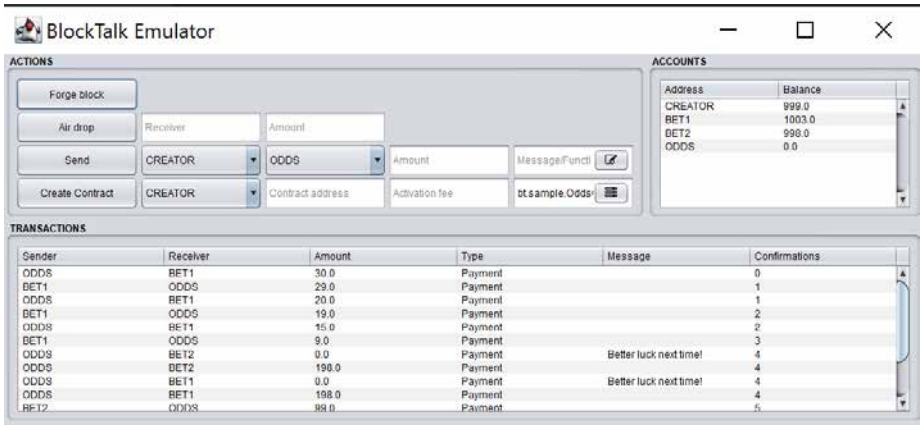

Рис.3.16 Інтерфейс системи

Верхній сегмент інтерфейсу займає панель «Actions» (Дії), яка виступає командним центром системи. Тут сконцентровані інструменти ініціації системних процесів, зокрема:

- Forge block: функція примусової генерації нового блоку для фіксації поточних операцій у реєстрі.
- Air drop: механізм масового або тестового розподілу активів для наповнення балансів акаунтів.
- Send: стандартний інструмент для проведення фінансових переказів між існуючими адресами.
- Create Contract: спеціалізований модуль для розгортання смарт-контрактів, що дозволяє гнучко налаштовувати реквізити сторін та параметри активаційної комісії.

Праворуч розміщено інформаційний блок «Accounts» (Акаунти). Він представлений у вигляді динамічної таблиці, що в реальному часі відображає стан усіх зареєстрованих у системі адрес. Структурований вигляд (адреса та актуальний баланс) дозволяє розробнику або користувачу миттєво оцінювати платоспроможність вузлів та коректність виконання транзакційних операцій.

Нижню частину робочого простору відведено під блок «Transactions» (Транзакції), який виконує роль деталізованого журналу подій (ledger). Кожен запис у цій секції містить вичерпну інформацію про рух активів: від реквізитів відправника та отримувача до суми переказу, типу транзакції (наприклад, платіж або виклик контракту) та супровідних метаданих. Важливим елементом є індикатор підтверджень, що демонструє статус фіналізації кожної операції в ієрархії блоків.

Цей комплексний підхід до побудови інтерфейсу трансформує складні процеси блокчейн-емуляції у наочний та керований формат, мінімізуючи поріг входження для тестування смарт-контрактів та аналізу транзакційних потоків. Детальніше ознайомитися з візуальною структурою та логікою розміщення елементів керування можна на рис. 3.17.

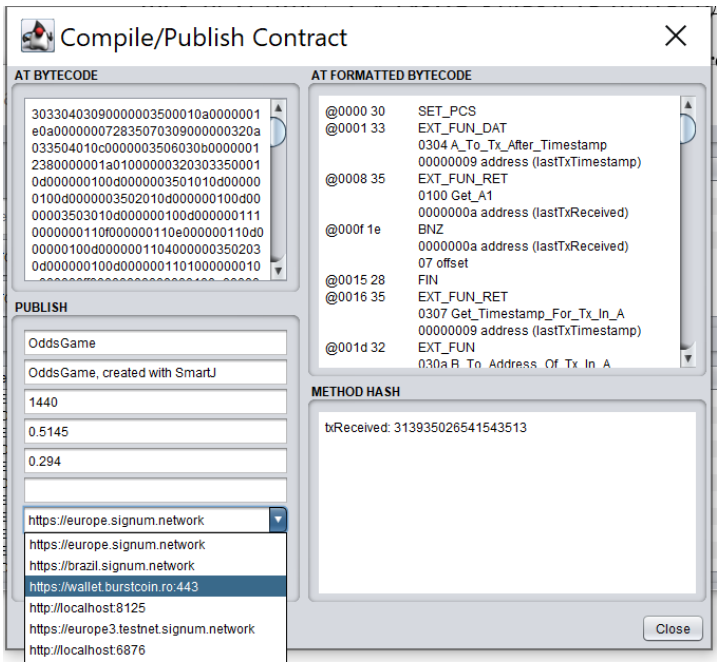

Рис.3.17 Компіляція і публікація смарт-контракту

Наступна панель інтерфейсу «BlockTalk Emulator» зосереджена на інструментарії розгортання та низькорівневого аналізу програмного коду смарт-контрактів. Ця частина робочого простору забезпечує прозорість процесу трансформації високорівневої логіки у машинний код, що безпосередньо виконується віртуальною машиною блокчейну.

У лівій верхній секції розташоване поле «AT Bytecode», яке відображає скомпільований байткод смарт-контракту. Це репрезентація коду у вигляді шістнадцяткової послідовності, яка є «сирим» матеріалом для мережі. Саме цей масив даних завантажується в розподілений реєстр і стає незмінною частиною блокчейну після підтвердження транзакції.

Для полегшення аудиту та налагодження коду передбачена панель «AT Formatted Bytecode», що розташована праворуч. Вона виконує роль декомпілятора в реальному часі, трансклюючи масив чисел у зрозумілий формат інструкцій (mnemonics). Тут кожна операція відображається як окрема команда з відповідними параметрами та адресами, що дозволяє розробнику верифікувати коректність скомпільованого коду перед його публікацією.

Нижній сегмент інтерфейсу відведено під блок «Publish», який є фінальним етапом ініціалізації контракту в мережі. Він містить наступні ключові елементи:

- Метадані контракту: поля для введення назви (наприклад, «OddsGame») та розширеного опису («OddsGame, created with SmartJ»), що допомагає ідентифікувати призначення контракту в загальному реєстрі.
- Конфігураційні параметри: встановлення часу життєвого циклу (TTL — Time To Live, наприклад, 1440 блоків) та визначення фінансових лімітів для виконання коду.
- Селектор вузла: випадаюче меню зі списком URL-адрес блокчейн-нод. Цей функціонал дозволяє гнучко перемикатися між локальними симуляторами, тестовими мережами (Testnet) та основними мережами (Mainnet).

Окремим важливим елементом, розташованим праворуч від панелі публікації, є поле «Method Hash». У ньому відображається криптографічний відбиток (хеш) конкретної функції, як-от txReceived. Оскільки блокчейн оперує адресами та хешами, а не іменами функцій у текстовому форматі, наявність цього значення є критично важливою для коректного виклику методів смарт-контракту зовнішніми транзакціями.

Цей інструментарій дозволяє розробнику повністю контролювати процес переходу від написання коду до його фінального розгортання, забезпечуючи високу надійність децентралізованих додатків.

4 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ СИСТЕМИ

4.1 Тестування функціональності системи.

Для перевірки коректності роботи системи було розроблено велику кількість тестових сценаріїв, які охоплюють різні аспекти її функціонування. Кожен із тестових класів, наведених у списку, відповідає за окремий функціональний або технічний модуль системи та дозволяє перевірити його поведінку в різних умовах.

Зокрема, такі класи, як OddsGame, Crowdfund, Auction, PaymentChannel, NFT2 та інші, використовуються для моделювання різноманітних сценаріїв використання, що забезпечує всебічну перевірку стабільності роботи смарт-контрактів. Основною метою кожного тестового випадку є виявлення можливих помилок, оптимізація логіки виконання, а також оцінка рівня безпеки та продуктивності системи в цілому. Перелік тестових класів представлено на рисунку 4.1.

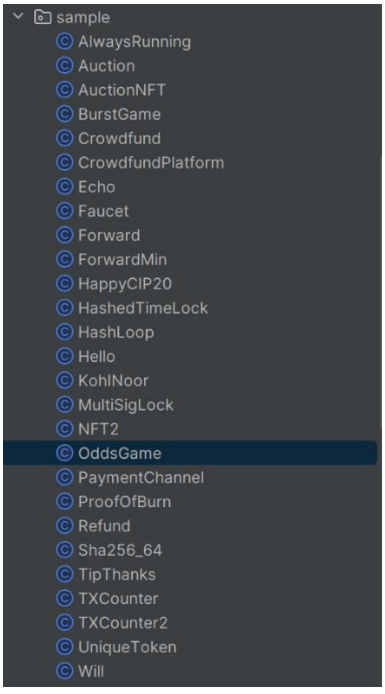

Рис.4.1 Класи з тестовими випадками

Для прикладу розглянемо кілька класів для тестування. Почнемо з прикладу BurstGame. Програмний код тестового випадку розглянемо на рис.4.2.

```java
@TargetCompilerVersion(CompilerVersion.v0_0_0)
public class BurstGame extends Contract {

    Address challenger;

    long challengerAmount;
    long creatorAmount;
    long balance;
    long blockHash;

    static final String DEV_ADDRESS = "BURST-JJQ8-MMA4-GHB4-4ZNZU";
    public void txReceived(){
        challenger = getCurrentTx().getSenderAddress();
        if(challenger == getCreator()){
            if(getCurrentTx().getAmount() == 0){
                // If creator sends a message with 0 amount (exactly the activation fee)
                // we withdraw the current balance
                sendBalance(challenger);
            }
            // do nothing, creator is just increasing his amount
            return;
        }
        challengerAmount = getCurrentTx().getAmount();
        balance = getCurrentBalance();
        creatorAmount = balance - challengerAmount;
        // sleep two blocks
        sleep(2);
        blockHash = getPrevBlockHash().getValue1();
        blockHash &= 0x0FFFFFFFFFFFFFFFL; // avoid negative values
        blockHash %= balance;

        if(blockHash < creatorAmount){
            // tip developer with 0.5 percent
            sendAmount(balance/200, parseAddress(DEV_ADDRESS));
            // creator wins
            sendBalance(getCreator());
        }
        else {
            // tip developer with 1 percent
            sendAmount(balance/100, parseAddress(DEV_ADDRESS));
            // challenger wins
            sendBalance(challenger);
        }
    }
    public static void main(String[] args) throws Exception {
        // some initialization code to make things easier to debug
        Emulator emu = Emulator.getInstance();
        Address creator = Emulator.getInstance().getAddress("CREATOR");
        Address challenger = Emulator.getInstance().getAddress("CHALLENGER");
        emu.airDrop(creator, 1000 * ONE_BURST);
        emu.airDrop(challenger, 1000 * ONE_BURST);
        Address odds = Emulator.getInstance().getAddress("GAME");
        emu.createConctract(creator, odds, BurstGame.class, ONE_BURST);
        emu.forgeBlock();
        emu.send(challenger, odds, 100*ONE_BURST);
        emu.send(challenger, odds, 100*ONE_BURST);
        emu.send(challenger, odds, 100*ONE_BURST);
        emu.send(challenger, odds, 100*ONE_BURST);
        emu.send(challenger, odds, 100*ONE_BURST);
        emu.send(challenger, odds, 100*ONE_BURST);
        emu.send(challenger, odds, 100*ONE_BURST);
        emu.send(challenger, odds, 100*ONE_BURST);
        emu.send(challenger, odds, 100*ONE_BURST);
        emu.send(challenger, odds, 100*ONE_BURST);
        emu.send(challenger, odds, 100*ONE_BURST);
        emu.forgeBlock();
        emu.forgeBlock();
        emu.send(challenger, odds, 10*ONE_BURST);
        emu.forgeBlock();
        emu.send(challenger, odds, 20*ONE_BURST);
        emu.forgeBlock();
        emu.forgeBlock();
        emu.send(challenger, odds, 30*ONE_BURST);
        emu.forgeBlock();
        emu.forgeBlock();
        new EmulatorWindow(BurstGame.class);
    }
}
```

Рис.4.2 Програмний код тестового класу BurstGame

Нас зустрічає програма з даними, що можемо побачити на рис. 4.3.

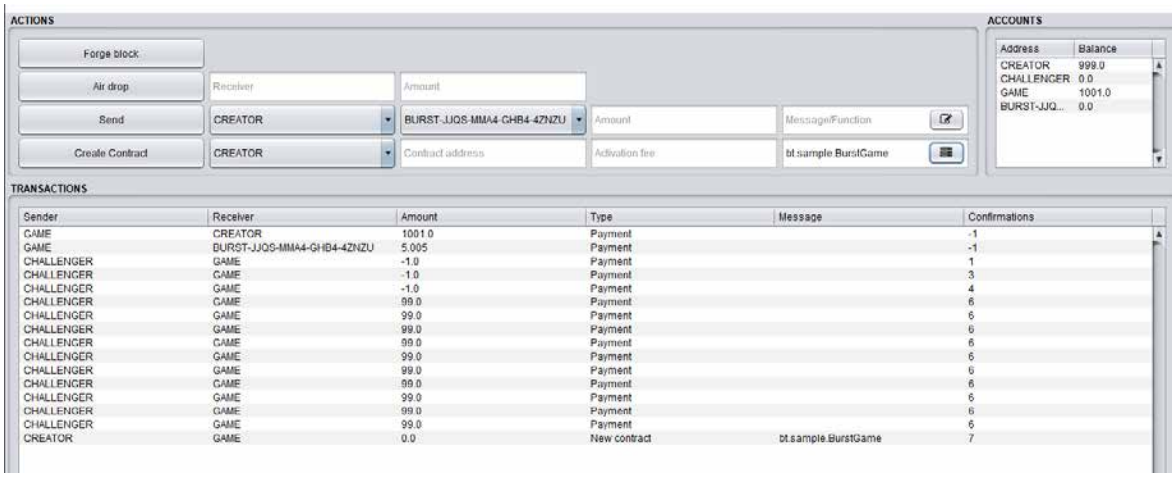

Рис.4.3 Програмна реалізація BurstGame

Верхній сегмент призначений для оперативного управління станом мережі та включає панель дій з критично важливими функціями: «Forge block» (формування блоку), «Air drop» (тестовий розподіл активів), «Send» (прямий переказ) та «Create Contract» (ініціалізація нового смарт-контракту). З правого боку розгорнуто динамічний перелік активних облікових записів, де чітко простежуються ролі учасників: системний акаунт CREATOR, опонент CHALLENGER, цільовий контракт GAME, а також специфічна адреса формату Signum/Burst — BURST-JJQS-MMA4-GHB4-4ZNZU.

Нижня частина інтерфейсу відведена під транзакційний лог, що фіксує інтенсивний обмін даними та активами між суб'єктами. Зокрема, зафіксовано потік операцій від CHALLENGER до GAME. Більшість записів класифіковані як «Payment», що вказує на виконання фінансових зобов'язань або здійснення ігрових ставок. Важливим технічним індикатором є стовпець «Confirmations», який дозволяє в реальному часі відстежувати глибину занурення транзакції в блокчейн та її остаточну валідацію.

Аналіз представленого стану системи свідчить про проведення комплексного стрес-тестування контракту BurstGame. Цей сценарій дозволяє верифікувати поведінку смарт-контракту в умовах активної взаємодії, зокрема:

- коректність ідентифікації вхідних платежів від гравців;
- точність алгоритмів перерахунку балансів сторін;

- швидкість та надійність обробки підтверджень у межах циклу формування блоків.

Результати цієї взаємодії та логічна структура самого контракту детально представлені на рис. 4.4, що дозволяє співвіднести візуальні зміни в інтерфейсі з програмним кодом алгоритму.

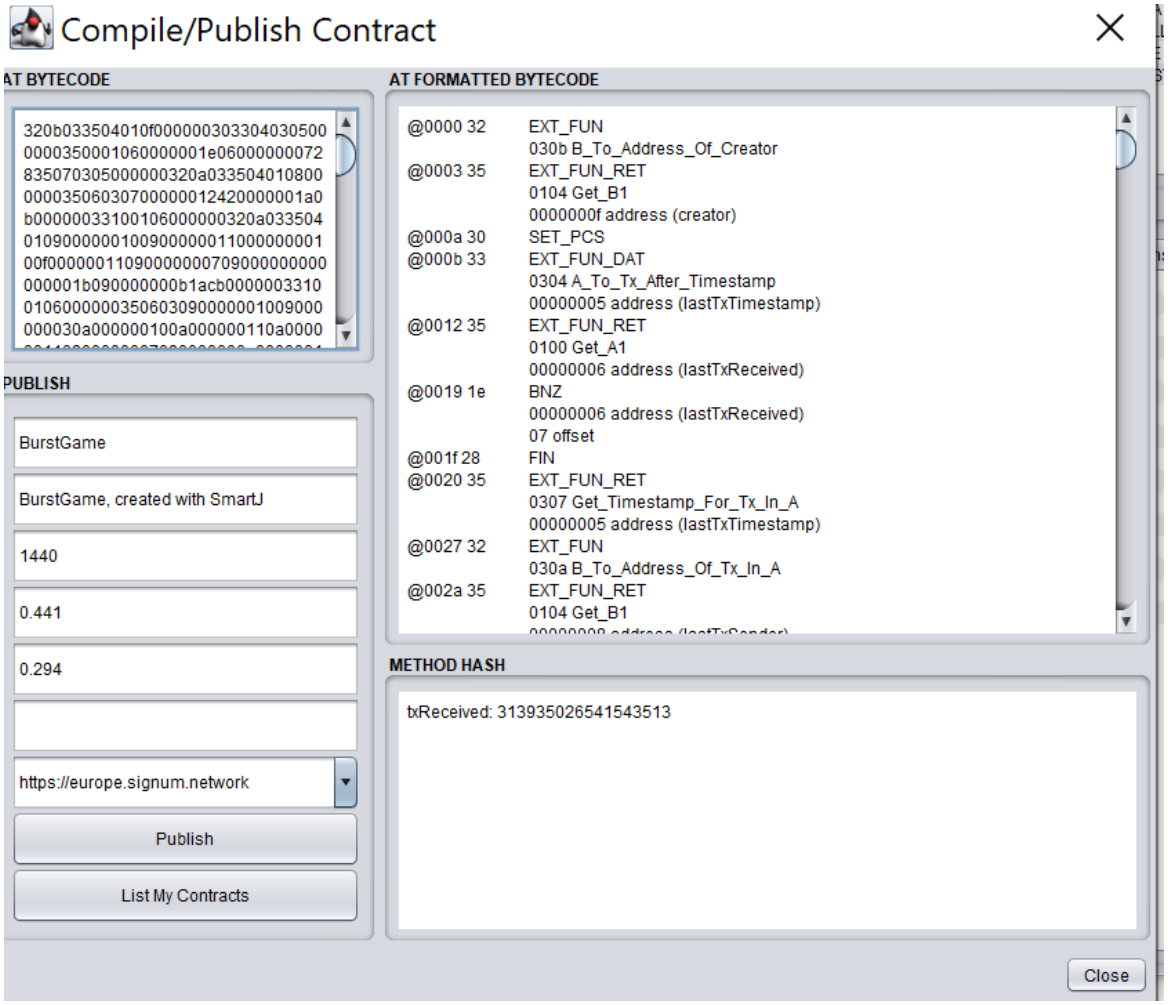

Рис.4.4 Інтерфейс вкладки Publish Contract

У лівій частині інтерфейсу розташовано блок «AT BYTECODE», де представлено скомпільований байт-код смарт-контракту. Ця послідовність є низькорівневою інструкцією для віртуальної машини блокчейну, що забезпечує виконання програмної логіки контракту в децентралізованому середовищі.

Центральну частину вікна займає секція «AT FORMATTED BYTECODE», яка транслює байт-код у зручний для аналізу вигляд асемблерних інструкцій. Тут відображені конкретні команди, такі як EXT_FUN (виклик зовнішньої функції), SET_PCS (встановлення лічильника команд) та FIN (завершення операції). Таке

представлення дозволяє детально простежити механіку роботи контракту з адресами та внутрішніми змінними блокчейну.

Розділ «PUBLISH» призначений для фінальної конфігурації та розгортання контракту. Користувач має можливість визначити назву («OddsGame»), короткий опис, а також технічні та фінансові параметри: термін дії (1440 блоків), комісію за публікацію (0.441) та операційні збори (0.294). Випадаючий список дозволяє обрати цільову мережу для публікації, зокрема Signum Network.

У правому нижньому куті, в секції «METHOD HASH», зафіксовано унікальний ідентифікатор методу txReceived. Цей хеш є критично важливим для точної адресації та ініціації конкретних функцій контракту під час виконання транзакцій.

Окрему увагу варто приділити тестовому випадку NFT2, який вирізняється найбільш складною архітектурною реалізацією (програмний код наведено у ДОДАТКУ Г). Візуалізація інтерфейсу під час запуску цього сценарію представлена на рис. 4.5.

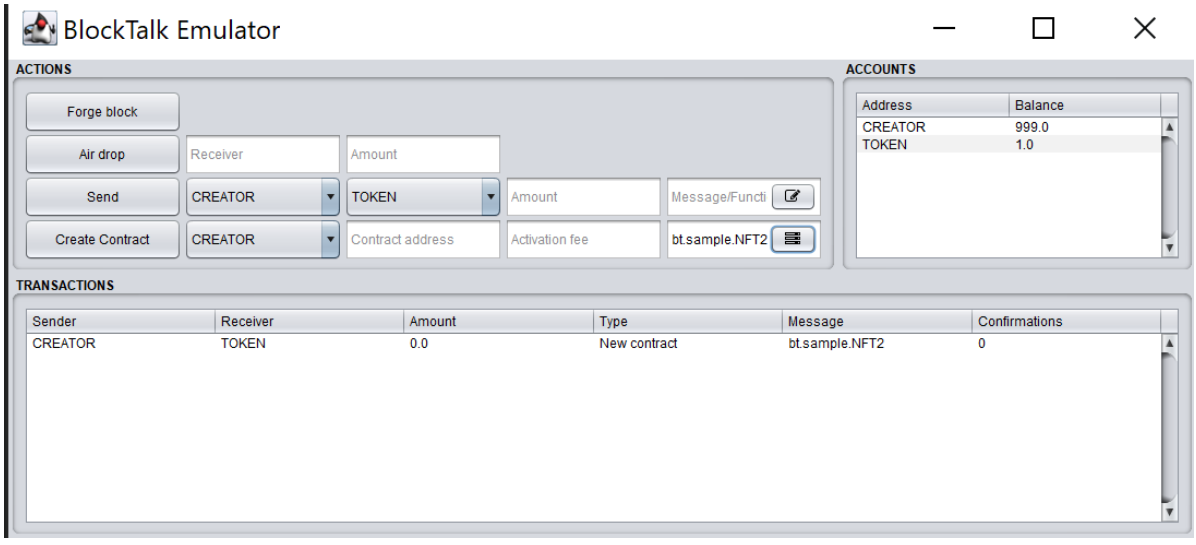

Рис. 4.5 Інтерфейс програмної системи тестового випадку NFT2

У цьому прикладі набагато цікавішим є самий контракт який можемо побачити на рис. 4.6.

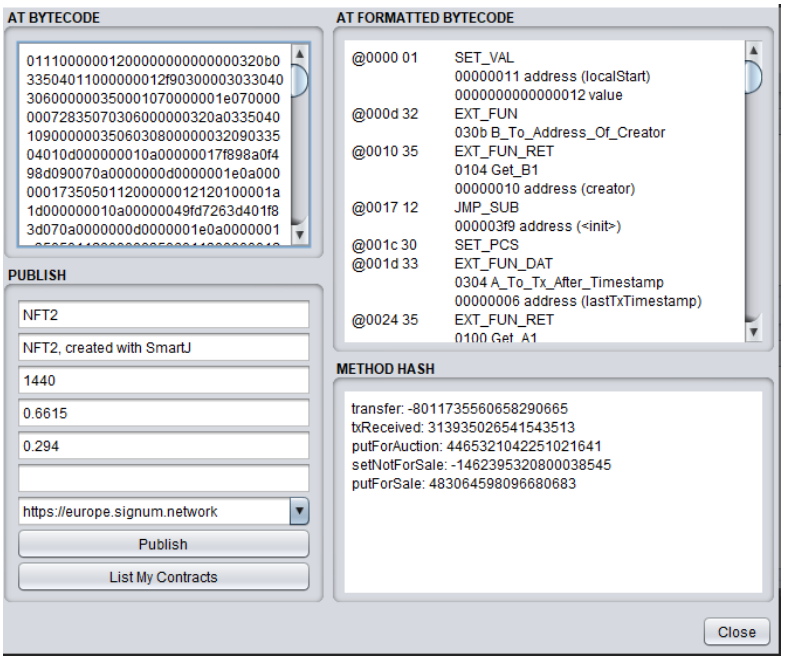

Рис.4.6 Інтерфейс і публікація контракту NFT2

У правій частині інтерфейсу, в секції «AT FORMATTED BYTECODE», реалізовано представлення байт-коду у вигляді асемблерних команд, що значно спрощує аналіз етапів виконання алгоритму. Команди на кшталт SET_VAL (присвоєння значення), EXT_FUN (виклик зовнішньої функції) та JMP_SUB (перехід до підпрограми) розкривають внутрішню логіку контракту, демонструючи механіку взаємодії з адресами та системними змінними. Це забезпечує можливість детального аудиту процесів, що відбуваються безпосередньо під час виконання коду віртуальною машиною.

Розділ «PUBLISH» надає інструментарій для фінальної налаштування параметрів розгортання. Користувач може визначити ідентифікаційні дані контракту (назву та опис), встановити часовий регламент підтвердження (1440 блоків), а також задати фінансові параметри: комісію за публікацію (0.6615) та транзакційні збори за виконання (0.294). Вибір цільового середовища для інсталяції коду здійснюється через селектор мереж, де за замовчуванням встановлено Signum Network.

Нижче, у спеціалізованому блоці «METHOD HASH», згенеровано унікальні криптографічні відбитки для ключових методів контракту, серед яких: transfer, txReceived, putForAuction, setNotForSale та putForSale. Ці хеш-значення виконують роль точних покажчиків, що дозволяють системі ідентифікувати та

активувати відповідну функціональну логіку під час зовнішніх викликів або в процесі тестування в блокчейн-середовищі.

Завершальний етап взаємодії з інтерфейсом передбачає використання кнопки «Publish» для розгортання скомпільованого контракту в обраній мережі. Крім того, функція «List My Contracts» надає користувачеві можливість зручного перегляду та моніторингу всіх раніше опублікованих ним смарт-контрактів.

4.2 Аналіз тестування і оптимізація

Аналіз результатів тестування та подальша оптимізація є фундаментальними етапами розробки програмного забезпечення, що безпосередньо впливають на його відмовостійкість, швидкість роботи та зручність для кінцевого користувача. Після завершення фази програмування система проходить через цикл комплексних перевірок, мета яких — не лише виявлення програмних помилок та вразливих місць, а й глибока верифікація функціоналу на відповідність технічним вимогам та очікуваній логіці поведінки в реальних умовах експлуатації.

На цьому етапі досліджується стійкість архітектури під різними типами навантажень, у межах специфічних користувацьких сценаріїв та за умов обмеженості апаратних ресурсів. Процес оптимізації, своєю чергою, спрямований на досягнення максимальної операційної ефективності, скорочення часу відгуку системи та прискорення обробки потоків даних. Водночас цей етап вимагає високої точності та ітераційного підходу, щоб внесені покращення не порушили цілісність і стабільність уже існуючих функцій.

Для систематичного огляду проведених досліджень перейдемо до розгляду конкретних тестових випадків, детальні параметри яких наведені в таблиці 4.1.

Процес оптимізації системи спрямований на інтенсифікацію її продуктивності, мінімізацію експлуатаційних витрат та підтримання стабільної роботи в умовах масштабування. Першочерговим етапом є аудит архітектурних компонентів для виявлення «вузьких місць» — критичних вузлів, що обмежують

швидкість роботи при пікових навантаженнях. Це передбачає верифікацію часових інтервалів обробки транзакцій, оцінку ефективності криптографічних протоколів, а також моніторинг споживання оперативної пам’яті та обчислювальних потужностей процесора.

Таблиця 4.1

Тестування функціональності створення контракту

Тестовий випадок	Опис	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
Створення нового контракту	Тестування створення контракту	Тип контракту: NFT2, адреса платформи, комісія	Контракт успішно створено	+	Pass/Fail
Створення контракту з помилковими даними	Перевірка, якщо вхідні дані некоректні	Тип контракту: NFT2, комісія: негативна	Повертається помилка про некоректні дані	+	Pass/Fail
Перевірка ідентифікації контракту	Перевірка унікальності контракту при створенні	Ідентифікатор: унікальний	Контракт отримує унікальний ID	+	Pass/Fail

Перейдемо до наступного тестового випадку представленого у таблиці 4.2.

Таблиця 4.2

Тестування функціональності транкзакцій

Тестовий випадок	Опис	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
Відправка транзакції	Тестування відправки транзакції між рахунками	Відправник: "CREATOR", отримувач: "GAME", сума: 100	Транзакція успішно надіслана та оброблена	+	Pass/Fail
Неправильний формат даних	Відправка транзакції з некоректними даними	Отримувач: "НевірнийID", сума: -50	Помилка в обробці транзакції	+	Pass/Fail
Перевірка балансу	Перевірка оновлення балансу після транзакції	Відправник: "CREATOR", отримувач: "GAME", сума: 200	Баланси обох рахунків оновлено коректно	+	Pass/Fail

Наступною буде перевірка сценаріїв аукціонів представлена у таблиці 4.3.

Таблиця 4.3

Тестування сценаріїв аукціонів

Тестовий випадок	Опис	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
Старт аукціону	Створення аукціону з мінімальною ставкою	Початкова ставка: 50, тайм-аут: 1 год	Аукціон успішно створено	+	Pass/Fail
Збільшення ставки	Тестування підвищення ставки	Попередня ставка: 50, нова ставка: 60	Ставку успішно підвищено	+	Pass/Fail
Завершення аукціону	Перевірка завершення аукціону після тайм-ауту	Тайм-аут: пройдено	Власник змінюється, фінали завершені	+	Pass/Fail

І останнім сценарієм, буде тестування функціональності NFT-токенів, що представлено у таблиці 4.4.

Таблиця 4.4

Тестування функціональності NFT-токенів

Тестовий випадок	Опис	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
Створення NFT	Перевірка створення NFT-токену	Назва: "ArtPiece", ціна: 100 SIGNA	NFT успішно створено	+	Pass/Fail
Трансфер NFT	Передача NFT іншому користувачу	Власник: "USER1", новий власник: "USER2"	NFT успішно передано новому власнику	+	Pass/Fail
Встановлення продажу	Встановлення ціни на продаж NFT	NFT ID: 1, нова ціна: 200 SIGNA	Ціна на NFT встановлена	+	Pass/Fail

Фундаментальним аспектом є вдосконалення алгоритмів обробки транзакцій. Використання механізму Proof of Capacity (PoC) на платформі Signum дозволяє суттєво знизити енергоспоживання та операційні витрати на підтримку мережі. Оптимізація тут полягає у раціоналізації використання дискового простору для зберігання пре-обчислених хешів та зниженні складності динамічних обчислень. Такий підхід нівелює потребу у постійній

експлуатації високовартісного обладнання, що робить систему екологічно стійкою та економічно вигідною.

На рівні смарт-контрактів оптимізація фокусується на рефакторингу байт-коду. Зменшення фізичного розміру коду та мінімізація кількості методів і вхідних параметрів дозволяють економити системні ресурси під час їх виконання в блокчейні. Оскільки кожна операція потребує певних обчислювальних витрат на обробку та архівацію, лаконічність коду безпосередньо впливає на пропускну здатність мережі.

Для покращення користувацького досвіду впроваджено асинхронну обробку запитів в інтерфейсі системи. Це унеможливлює блокування робочого простору при одночасній взаємодії багатьох користувачів, забезпечуючи високу швидкість відгуку та відсутність візуальних затримок.

Додатковим інструментом підвищення ефективності є кешування даних. Агрегація часто запитуваної інформації, як-от актуальні баланси рахунків або останні транзакційні записи, у проміжному сховищі дозволяє суттєво розвантажити основну базу даних. Це прискорює доступ до інформації та гарантує стабільність системи під час інтенсивної експлуатації.

4.3 Оцінка ефективності системи

Оцінка ефективності системи ґрунтується на комплексному аналізі її продуктивності, стійкості до інтенсивних навантажень та експлуатаційної зручності. Ключові аспекти результативності платформи можна структурувати наступним чином:

- Енергоефективність та стійкість: Завдяки впровадженню алгоритму консенсусу Proof of Capacity (PoC), платформа Signum демонструє радикально вищі показники енергоефективності порівняно з традиційними системами Proof of Work (PoW). Відмова від енергомістких обчислень на користь використання дискового простору дозволяє підтримувати працездатність мережі на базовому апаратному забезпеченні. Це не лише знижує екологічний вплив, а й суттєво

зменшує бар'єр входу для нових учасників та операційні витрати на обслуговування інфраструктури.

- Продуктивність та масштабованість: Система демонструє високу швидкість обробки транзакцій завдяки впровадженню асинхронних методів обробки даних. Це мінімізує латентність при взаємодії користувачів зі смарт-контрактами. Додаткова оптимізація через багаторівневе кешування та раціоналізацію алгоритмів зберігання забезпечує миттєвий відгук на типові запити (перевірка балансів, моніторинг статусів), що критично важливо для підтримки безперервного користувацького досвіду під час пікових навантажень.
- Криптографічний захист: Безпека системи базується на інтеграції перевіреної часом бібліотеки BouncyCastle. Це гарантує найвищий рівень надійності при створенні цифрових підписів, верифікації транзакційних пакетів та шифруванні даних. Використання промислових криптографічних стандартів унеможливлює несанкціоновану модифікацію записів та забезпечує абсолютну цілісність розподіленої книги.

У підсумку, оцінка ефективності підтверджує досягнення оптимального балансу між обчислювальною потужністю, безпекою та ресурсомісткістю. Система спроможна підтримувати стабільну роботу в динамічному середовищі, що позиціонує її як надійне та масштабоване рішення для широкого спектра децентралізованих додатків.

ВИСНОВКИ

У дипломній роботі проведено всебічне дослідження сучасних екосистем смарт-контрактів, у результаті якого ідентифіковано критичні вразливості та функціональні обмеження існуючих рішень. Це обґрунтувало необхідність створення вдосконаленої системи управління, адаптованої до специфіки децентралізованих середовищ із високими вимогами до безпеки та операційної ефективності. На основі отриманих даних сформовано технічні вимоги до розробки платформи, що базується на блокчейн-технологіях, забезпечуючи прозорість операцій, надійне зберігання даних та мінімізацію ризиків несанкціонованого втручання.

Розроблена архітектура системи враховує фундаментальні принципи децентралізації та потреби користувачів у конфіденційності. Вона інтегрує комплекс криптографічних інструментів для захисту транзакцій та забезпечує цілісність даних у межах розподіленого реєстру.

Апробація реалізованого прототипу підтвердила повну відповідність системи встановленим вимогам. Тестування зафіксувало високу ефективність проведення транзакцій, надійність механізмів захисту та стійкість до специфічних загроз блокчейн-середовища. Результати аналізу підтверджують готовність платформи до практичного розгортання та її здатність виконувати поставлені завдання.

Визначено стратегічні напрямки для інтеграції системи в реальні сектори економіки, де автоматизація через смарт-контракти може суттєво підвищити ефективність бізнес-процесів. Також окреслено вектори подальшої модернізації для нарощування продуктивності та розширення функціоналу. Розроблена система на базі блокчейну Signum та алгоритму PoC є перспективним інструментом для розвитку цифрової комерції, управління активами та сфери децентралізованих фінансів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бутерин В. Ефіріум: наступне покоління криптовалюти та децентралізованого додатка. 2014. URL: https://ethereum.org/whitepaper.

2. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. URL: https://bitcoin.org/bitcoin.pdf.

3. Szabo N. Smart Contracts: Building Blocks for Digital Markets. 1996. URL: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html.

4. Стегній А. В. Дослідження сучасних підходів до побудови блокчейн-систем // Інформаційні технології та комп'ютерна інженерія. 2021. № 2. С. 45–53.

5. Antonopoulos A. Mastering Bitcoin: Unlocking Digital Cryptocurrencies. O'Reilly Media, 2017. 298 p.

6. Мартинов І. І. Блокчейн та смарт-контракти: нові можливості для бізнесу // Бізнес Інформ. 2020. № 9. С. 66–71.

7. Tikhomirov S. Security Analysis of Ethereum Smart Contracts. 2017. URL: https://arxiv.org/abs/1703.03994.

8. Харченко В. С. Розробка та впровадження блокчейн технологій в Україні // Науковий вісник ХНТУ. 2021. № 4. С. 90–98.

9. Mougayar W. The Business Blockchain: Promise, Practice, and the Application of the Next Internet Technology. Wiley, 2016. 208 p.

10. Кравець В. В., Мартинов А. О. Смарт-контракти: концепція, технологія та перспективи застосування // Вісник ХНУ імені В.Н. Каразіна. 2019. № 10. С. 121–128.

11. Popov S. The Tangle. 2018. URL: https://iota.org/IOTA_Whitepaper.pdf.

12. Кузнєцов О. М. Енергоефективність та безпека блокчейн-алгоритмів // Вісник Національного технічного університету «ХПІ». 2022. № 1. С. 45–51.

13. Greenspan G. Multichain Private Blockchain: White Paper. 2015. URL: https://www.multichain.com/download/MultiChain-White-Paper.pdf.

14. Шахов Д. В., Павленко М. П. Блокчейн та кібербезпека: перспективи застосування в Україні // Інформаційні технології та безпека. 2020. № 5. С. 112–119.

15. Wood G. Ethereum: A Secure Decentralised Generalised Transaction Ledger. Yellow Paper, 2014. URL: https://ethereum.github.io/yellowpaper/paper.pdf.

16. Singh A., Singh K. Blockchain Basics. Springer, 2019. 90 p.

17. Шевченко О. О. Актуальні питання розробки смарт-контрактів у децентралізованих системах // Вісник Київського національного університету ім. Тараса Шевченка. 2021. № 3. С. 70–78.

18. Pilkington M. Blockchain Technology: Principles and Applications. Research Handbook on Digital Transformations. Edward Elgar Publishing, 2016. P. 225–253.

19. Мірошниченко Д. В. Блокчейн-технології в електронному управлінні: проблеми і перспективи розвитку // Державне управління: удосконалення та розвиток. 2021. № 10. С. 38–44.

20. Bashir I. Mastering Blockchain: Unlocking the Power of Cryptocurrencies, Smart Contracts, and Decentralized Applications. Packt Publishing, 2020. 586 p.

21. Петров Д. І., Іванов О. П. Смарт-контракти у фінансових технологіях: застосування та виклики // Фінансовий ринок. 2022. № 2. С. 54–61.

22. Gupta M. Blockchain for Dummies. Wiley, 2017. 212 p.

23. Климчук С. В., Шевченко А. В. Проблеми реалізації смарт-контрактів на базі блокчейн-технологій // Економіка та право. 2020. № 6. С. 99–106.

24. Risius M., Spohrer K. A Blockchain Research Framework. Business & Information Systems Engineering, 2017. Vol. 59. No. 6. P. 385–409.

25. Романюк А. І., Павлова М. В. Вплив блокчейн-технології на розвиток фінансових ринків // Інноваційна економіка. 2021. № 12. С. 103–109.

26. Buterin V., Griffith V. Casper the Friendly Finality Gadget. 2017. URL: https://arxiv.org/abs/1710.09437.

27. Ткаченко В. М., Кириченко Л. О. Можливості використання блокчейн технологій у сфері медицини // Здоров'я нації. 2022. № 1. С. 45–51.

28. Yaga D., Mell P., Roby N., Scarfone K. Blockchain Technology Overview. NIST, 2018. URL: https://nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8202.pdf.

29. Drescher D. Blockchain Basics: A Non-Technical Introduction in 25 Steps. Apress, 2017. 255 p.

30. Байда С. В. Проблеми та перспективи впровадження блокчейн-технологій в Україні // Наукові записки. 2021. № 3. С. 60–67.

ДОДАТОК А

ЛІСТИНІНГ КОДУ СМАРТ-КОНТРАКТУ

```
public abstract class Contract {

    public final static long ONE_BURST = 100000000L;
    public final static long FEE_QUANT = 735000L;
    public final static long STEP_FEE = FEE_QUANT / 10L; // After AT2 fork,
othewise ONE_BURST/10

    Address address;
    Address creator;
    Timestamp creation;

    Transaction currentTx;
    long activationFee;

    // Thread stuff to emulate sleep functions
    Semaphore semaphore = new Semaphore(1);
    boolean running;
    Timestamp sleepUntil;

    protected Contract() {
       Emulator emu = Emulator.getInstance();
       setInitialVars(emu.curTx, new
Timestamp(emu.getCurrentBlock().getHeight(), 0));
    }

    /**
     * Utility function that return the address of a given BURST-account
     *
     * @param account
     * @return
     */
    protected Address parseAddress(String rs) {
       return Emulator.getInstance().getAddress(rs);
    }

    /**
     * Utility function that return the address of a given ID
     * @param id the signed long id
     * @return the address
     */
    protected Address getAddress(long id) {
       return
Emulator.getInstance().getAddress(SignumCrypto.getInstance().rsEncode(SignumID
.fromLong(id)));
    }

    /**
     * Send the entire balance to the given address.
     *
     * Care should be exercised with this function since a contract with no
balance
     * cannot continue to run!
     *
     * @param ad the address
     */
```

```
    protected void sendBalance(Address ad) {
        sendAmount(this.address.balance, ad);
    }

    /**
     * Send the given amount to the receiver address
     *
     * @param amount
     * @param receiver
     */
    protected void sendAmount(long amount, Address receiver) {
        Emulator.getInstance().send(address, receiver, amount);
    }

    /**
     * Send the given message to the given address.
     *
     * The message has the isText flag set as false, the hexadecimal values
are
     * converted to characters when shown in BRS wallet. Message is always
     * unencrypted.
     *
     * @param message  the message, truncated in 4*sizeof(long)
     * @param amount   the amount or 0 to send the message only
     * @param receiver the address
     */
    protected void sendMessage(String message, Address receiver) {
        Emulator.getInstance().send(address, receiver, 0, message);
    }

    /**
     * Send the given message to the given address.
     *
     * The message has the isText flag set as false, the hexadecimal values
are
     * converted to characters when shown in BRS wallet. Message is always
     * unencrypted.
     *
     * @param message  the message in form of a 4 longs register
     * @param receiver the address
     */
    protected void sendMessage(Register message, Address receiver) {
        Emulator.getInstance().send(address, receiver, 0, message);
    }

    /**
     * Send the given message to the given address.
     *
     * The message has the isText flag set as false, the given hexadecimal
value is
     * converted to characters when shown in BRS wallet. Message is always
     * unencrypted.
     *
     * @param message  the message in form of a long number
     * @param receiver the address
     */
    protected void sendMessage(long message, Address receiver) {
        Emulator.getInstance().send(address, receiver, 0,
Register.newInstance(message, 0, 0, 0));
    }

    /**
     * Send the given message to the given address.
     *
```

```
     * The message has the isText flag set as false, the given hexadecimal
values are
     * converted to characters when shown in BRS wallet. Message is always
     * unencrypted.
     *
     * @param message  the message in form of a long number
     * @param message2  the message in form of a long number
     * @param receiver the address
     */
    protected void sendMessage(long message, long message2, Address receiver)
{
       Emulator.getInstance().send(address, receiver, 0,
Register.newInstance(message, message2, 0, 0));
    }

    /**
     * Send the given message to the given address.
     *
     * The message has the isText flag set as false, the given hexadecimal
values are
     * converted to characters when shown in BRS wallet. Message is always
     * unencrypted.
     *
     * @param message  the message in form of a long number
     * @param message2  the message in form of a long number
     * @param message3  the message in form of a long number
     * @param message4  the message in form of a long number
     * @param receiver the address
     */
    protected void sendMessage(long message, long message2, long message3,
long message4, Address receiver) {
       Emulator.getInstance().send(address, receiver, 0,
Register.newInstance(message, message2, message3, message4));
    }

    /**
     * Function to be called before processing the transactions of the current
     * block.
     *
     * Users should overload this function if there is some action to be taken
     * before processing the first {@link #txReceived()} for the current
block.
     *
     * Use this function carefully, since #{@link #getCurrentTx()} and similar
     * functions are still unavalable inside this function.
     */
    protected void blockStarted() {
    }

    /**
     * Function to be called when all transactions on the current block were
     * processed.
     *
     * Users should overload this function if there is some action to be taken
after
     * the last {@link #txReceived()} was called for the current block.
     */
    protected void blockFinished() {
    }

    /**
     * Get the first transaction received after the given timestamp
     *
     * @param ts
```

```
     * @return
     */
    protected Transaction getTxAfterTimestamp(Timestamp ts) {
        return Emulator.getInstance().getTxAfter(address, ts);
    }

    /**
     * @return the transaction being processed at this moment
     */
    protected Transaction getCurrentTx() {
        return currentTx;
    }

    /**
     * @return the current transaction timestamp
     */
    protected Timestamp getCurrentTxTimestamp() {
        return getCurrentTx().getTimestamp();
    }

    /**
     * @return the current transaction sender address
     */
    protected Address getCurrentTxSender() {
        return getCurrentTx().getSenderAddress();
    }

    /**
     * @return the current transaction amount
     */
    protected long getCurrentTxAmount() {
        return getCurrentTx().getAmount();
    }

    /**
     * @return the block hash of the previous block (part 1 of 4)
     */
    protected Register getPrevBlockHash() {
        return Emulator.getInstance().getPrevBlock().hash;
    }

    /**
     * @return the first part of the previous block hash
     */
    protected long getPrevBlockHash1() {
        return Emulator.getInstance().getPrevBlock().hash.getValue1();
    }

    /**
     * @return the timestamp of the previous block
     */
    protected Timestamp getPrevBlockTimestamp() {
        return new Timestamp(Emulator.getInstance().getPrevBlock().getHeight(),
0);
    }

    /**
     * @return the timestamp of the block being processed
     */
    protected Timestamp getBlockTimestamp() {
        return new
Timestamp(Emulator.getInstance().getCurrentBlock().getHeight(), 0);
    }
```

```
    /**
     * @return the timestamp of the block being processed
     */
    protected long getBlockHeight() {
       return Emulator.getInstance().getCurrentBlock().getHeight();
    }

    /**
     * @return the address of the creator of this contract
     */
    protected Address getCreator() {
       return creator;
    }

    /**
     * @return the creation timestamp of this contract
     */
    protected Timestamp getCreationTimestamp() {
       return creation;
    }

    /**
     * @return the current balance of this contract
     */
    protected long getCurrentBalance() {
       return address.balance;
    }

    @EmulatorWarning
    public static Register performSHA256_(Register input) {
       Register ret = new Register();

       ByteBuffer b = ByteBuffer.allocate(32);
       b.order(ByteOrder.LITTLE_ENDIAN);

       for (int i = 0; i < 4; i++) {
          b.putLong(input.value[i]);
       }

       try {
          MessageDigest sha256 = MessageDigest.getInstance("SHA-256");
          ByteBuffer shab = ByteBuffer.wrap(sha256.digest(b.array()));
          shab.order(ByteOrder.LITTLE_ENDIAN);

          for (int i = 0; i < 4; i++) {
             ret.value[i] = shab.getLong(i * 8);
          }
       } catch (NoSuchAlgorithmException e) {
          // not expected to reach that point
          e.printStackTrace();
       }
       return ret;
    }

    /**
     * @return a SHA256 hash of the given input
     */
    protected Register performSHA256(Register input) {
       return performSHA256_(input);
    }

    /**
     * A utility function returning only the first 64 bits of a SHA256 for two
long
```

```
     * inputs.
     *
     * @return the first 64 bits SHA256 hash of the given input
     */
    protected long performSHA256_64(long input1, long input2) {
       Register input = new Register();
       input.value[0] = input1;
       input.value[1] = input2;

       Register ret = performSHA256_(input);
       return ret.getValue1();
    }

    /**
     * Sleeps until the contract receives a new transaction.
     */
    protected void sleepUntilNextTx() {
       // FIXME: on emulator we will sleep for one block
       sleep(0);
    }

    /**
     * Sleeps for the given number of blocks.
     *
     * @param nblocks number of blocks to sleep
     */
    protected void sleep(long nblocks) {
       if(nblocks <= 0)
          sleepUntil = null;
       else {
          sleepUntil = new
Timestamp(Emulator.getInstance().getCurrentBlock().height + nblocks, 0);
          address.setSleeping(true);
          try {
             semaphore.acquire();
          } catch (InterruptedException e) {
             e.printStackTrace();
          }
       }
       address.setSleeping(false);
       sleepUntil = null;
    }

    /**
     * A new transaction was received.
     *
     * Overload this function to implement your contract.
     */
    public abstract void txReceived();

    // ============================================================
    // Functions not visible, will be used by the virtual
    // machine when in debug mode but not exported to the
    // AT blockchain machine code

    void setInitialVars(Transaction tx, Timestamp creation) {
       this.creator = tx.sender;
       this.address = tx.receiver;
       this.creation = creation;
       this.activationFee = tx.getAmount();
       this.address.contract = this;

       // we acquire the semaphore in the creation process, it is released
       // when finished by the emulator
```

```
        try {
            running = true;
            semaphore.acquire();
        } catch (InterruptedException e) {
            running =false;
        }
    }

    void setCurrentTx(Transaction current) {
        this.currentTx = current;
    }

    @EmulatorWarning
    public String getFieldValues() {
        String ret = "<html>";
        Field[] fields = this.getClass().getDeclaredFields();
        for (Field f : fields) {
            try {
                f.setAccessible(true);
                Object v = f.get(this);
                ret += "<b>" + f.getName() + "</b> = " + (v != null ?
v.toString() : "null") + "<br>";
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return ret;
    }
}
```

ДОДАТОК Б

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Стецюк Софія Сергіївна, здобувачка НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота (проєкт) на тему «Розробка смарт-контракту для управління ланцюгами постачання» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.

- Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
  - [ ] інструменти генеративного ШІ не використовувалися.
  - [x] інструменти ШІ ChatGPT, DeepL були використані для:
    - [x] перекладу іншомовних джерел;
    - [ ] стилістичного редагування та перевірки граматики;
    - [x] допомоги у написанні/відлагодженні програмного коду;
    - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«__» _____ 20__р. _____ **Софія СТЕЦЮК**