

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет радіоелектроніки та телекомунікації
(повне найменування інституту, назва факультету)

Кафедра захисту інформації
(повна назва кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РЕАЛІЗАЦІЯ СМАРТ КОНТРАКТУ ДЛЯ ОПЛАТИ

КОМУНАЛЬНИХ ПОСЛУГ

IMPLEMENTATION OF A SMART CONTRACT FOR PAYING UTILITY

BILLS

Виконала: студент(ка) 4 курсу, групи РТ-819

Спеціальності 125 Кібербезпека

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Безпека інформаційних і комунікаційних

систем

ЗАЙЦЕВА А.О.

(прізвище та ініціали)

Керівник НЕЛАСА Г.В.

(прізвище та ініціали)

Рецензент САМОЙЛИК С.С.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет радіоелектроніки та телекомунікацій
Кафедра захисту інформації
Ступінь вищої освіти бакалавр
Спеціальність 125 Кібербезпека
(код і найменування)
Освітня програма (спеціалізація) Безпека інформаційних і телекомунікаційних систем
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ, к.т.н, доц.
В.О. ВОСКОБОЙНИК
«01» «06» 2023 року

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТКИ

ЗАЙЦЕВОЇ Аніти Олександрівни

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Реалізація смарт контракту для оплати комунальних послуг

Implementation of a smart contract for paying utility bills

керівник проєкту (роботи) к.т.н., доцент НЕЛАСА Ганна Вікторівна,

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові,)

затверджені наказом закладу вищої освіти від «04» квітня 2023 року №93

2. Строк подання студентом проєкту (роботи) 01 червня 2023 року

3. Вихідні дані до проєкту (роботи) створити платіжну систему на основі смарт-контракту, за допомогою якого можна буде розрахувати та сплатити кошти за спожиту електроенергію

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) платформа Ethereum на основі технології Blockchain, смарт-контракт. протидія кібератакам на смарт-контракти, програмна реалізація смарт-контракту для оплати послуг

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) презентація

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
Розділи 1-3	НЕЛАСА Г.В., к.т.н. доцент		
Нормоконтроль	КОРОЛЬКОВ Р.Ю., к.т.н. ст.викл.		

7. Дата видачі завдання « 01 » березня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Аналіз вразливостей смарт-контрактів	2 тиждень	Розділ 2
4	Програмна реалізація платіжної системи	3-4 тижні	Розділ 3
5	Оформлення пояснювальної записки та відповідної документації	6-7 тиждень	
			Додатки
7	Нормоконтроль та рецензування.	8 тиждень	
8	Захист дипломної роботи.	9 тиждень	

Студент(ка)

_____ Аніта ЗАЙЦЕВА
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Ганна НЕЛАСА
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
61 с., 3 табл., 5 рис., 3 дод., 24 джерел.

БЛОКЧЕЙН, БЛОКЧЕЙН-ПЛАТФОРМА ETHEREUM, СМАРТ
КОНТРАКТ, ДЕЦЕНТРАЛІЗОВАНІ ПРОГРАМИ, МОВА
ПРОГРАМУВАННЯ SOLIDITY

Об'єкт дослідження – система оплати комунальних послуг.

Предмет дослідження – смарт-контракт для оплати послуг.

Мета роботи – створити платіжну систему на основі смарт-контракту, за допомогою якого можна буде розрахувати та сплатити кошти за спожиту електроенергію.

Для реалізації програмної частини використовувалась мова програмування Solidity та середовище розробки Remix IDE.

В результаті ми отримали працюючий прототип системи оплати за спожиту електроенергію, яка реалізована із використанням смарт-контракту на Solidity, криптогаманця MetaMask та сайту, написаного із використанням html, css та java-script.

ABSTRACT

Explanatory note to the bachelor's thesis: 61 p., 3 tables, 5 figure, 3 appendix, 24 sources.

BLOCKCHAIN, ETHEREUM BLOCKCHAIN PLATFORM, SMART CONTRACT, DECENTRALIZED PROGRAMS, SOLIDITY PROGRAMMING LANGUAGE

The object of research is a utility payment system.

The subject of the study is a smart contract for payment for services.

The purpose of the work is to create a payment system based on a smart contract, which will allow you to calculate and pay for the consumed electricity.

The Solidity programming language and the Remix IDE were used to implement the software part.

As a result, we got a working prototype of a payment system for electricity consumed, which was implemented using a smart contract on Solidity, a MetaMask crypto wallet, and a website written using html, css, and java script.

ЗМІСТ

Перелік скорочень	7
Вступ.....	8
1 Платформа Ethereum на основі технології Blockchain	10
1.1 Технологія блокчейн.....	10
1.2 Блокчейн-платформа Ethereum.....	12
1.3 Алгоритми безпеки блокчейн	15
2 Смарт-контракт. Протидія кібератакам на смарт-контракти	18
2.1 Смарт-контракт	18
2.2 Різновид атак на смарт-контракти.....	21
2.3 Заходи щодо захисту смарт-контрактів.....	24
3 Програмна реалізація смарт-контракту для оплати послуг	31
3.1 Мова програмування Solidity.....	31
3.2 UML-діаграма системи оплати	32
3.3 Реалізація та опис коду смарт-контракту	34
Висновки	40
Перелік джерел посилання	41
ДОДАТОК А Код реалізації смарт-контракту для системи оплати	44
ДОДАТОК Б Код реалізації графічного інтерфейсу	46
ДОДАТОК В Презентація	54

ПЕРЕЛІК СКОРОЧЕНЬ

- DApp (Decentralized Application) – децентралізована програма;
DLT (Distributed Ledger Technology) – технологія розподіленої книги;
DoS (Denial of Service) – відмова в обслуговуванні;
ECDSA (Elliptic Curve Digital Signature Algorithm) – алгоритм цифрового підпису еліптичної кривої;
EVM (Ethereum Virtual Machine) – віртуальна машина Ethereum;
NFT (Non-Fungible Token) – незамінний токен;
PoS (Proof of Stake) – доказ ставки;
PoW (Proof of Work) – підтвердження роботи;
SHA (Secure Hash Algorithm) – алгоритм безпечного хешування

ВСТУП

Кожна компанія оточена паперовими контрактами, які, у більшості випадків, громіздкі та стають джерелом комерційних і судових суперечок. Рішенням для цієї проблеми можуть стати "розумні контракти", які спростять бізнес та торгівлю між анонімними ідентифікованими сторонами. Завдяки таким контрактам буде уникнення посередників, а також зменшення формальностей та витрат, пов'язаних з традиційними методами. При цьому всьому буде зберігатися надійність, безпека та автентичність.

Серед інших переваг смарт-контрактів можна виділити:

- забезпечення більшої безпеки завдяки використанню технології блокчейн, на якій базуються контракти;
- більша свобода для сторін, які укладають угоду, оскільки відсутня залежність від посередників;
- майже миттєве виконання угод для всіх сторін, як тільки необхідні умови будуть виконані;
- створення середовища прозорості та довіри, де всі учасники можуть переглядати інформацію про контракт у блокчейні;
- уникнення необхідності утримання великої компанії в ролі довіреної сторони, оскільки смарт-контракти розподілені між тисячами комп'ютерів. Це також допомагає уникнути бюрократії, цензури та пов'язаних з цим процесів великих витрат часу.

Актуальність теми даної дипломної роботи полягає в тому, що розумні контракти мають потенціал змінити різні галузі, такі як роздрібна торгівля, ланцюг постачання, промисловість та інші, шляхом прискорення транзакцій, зменшення паперової роботи та підвищення прибутковості.

Метою цієї дипломної роботи є створення платіжної системи на основі смарт-контрактів на Solidity, за допомогою якого можна спростити процес сплати комунальних платежів.

Завданнями дослідження є:

- розглянути та дослідити платформу Ethereum на основі технології Blockchain;
- проаналізувати, які вразливості мають смарт-контракти на платформі Ethereum;
- розглянути мову програмування смарт-контрактів Solidity;
- розробити платіжну систему на основі смарт-контракту для сплати комунальних послуг.

Методи дослідження включають аналітичний огляд наукової літератури та нормативних документів, теоретичний аналіз платформи блокчейн, та програмну реалізацію системи.

У цій дипломній роботі будуть висвітлені принципи роботи із смарт-контрактами на платформі Ethereum, а також спосіб реалізації онлайн-сервісів, які працюють із використанням платіжних систем на основі смарт-контрактів.

Структура дипломної роботи буде включати такі розділи: вступ, платформа Ethereum на основі технології Blockchain, смарт-контракт. Протидія кібератакам на смарт-контракти, програмна реалізація смарт-контракту для оплати послуг та висновки по результатам розробки.

В ході виконання роботи буде реалізовано смарт-контракт розрахунку та сплати за спожиту електроенергію, а також інтерфейс, через який можна буде взаємодіяти із смарт-контрактом на основі мови HTML та CSS.

Виконання цієї дипломної роботи допоможе зробити більш зручним і безпечним процес сплати комунальних послуг.

1 ПЛАТФОРМА ETHEREUM НА ОСНОВІ ТЕХНОЛОГІЇ BLOCKCHAIN

1.1 Технологія блокчейн

Блокчейн, який іноді називають технологією розподіленої книги (DLT), робить історію будь-якого цифрового активу незмінною та прозорою завдяки використанню децентралізованої мережі та криптографічного хешування [1].

Проведемо просту аналогію з документом Google Docs, щоб зрозуміти, як працює технологія блокчейн. Викладач створює документ Google, після чого ділиться ним із групою студентів. Цей документ не копіюється чи передається, він розповсюджується на всіх учасників. Таким чином створюється децентралізований ланцюжок розповсюдження, при якому кожному надається доступ до базового документа одночасно. Ніхто не блокується в очікуванні змін від іншої сторони, а усі зміни в документі записуються в режимі реального часу – це робить зміни абсолютно прозорими. Слід зазначити, що на відміну від Google Docs, оригінальний вміст і дані в блокчейні не можна змінювати після запису, що в свою чергу підвищує рівень безпеки.

Блокчейн – це цифровий список записів даних, який постійно зростає. Такий список складається з багатьох блоків даних, кожен з яких містить часову позначку, хеш попереднього блоку та дані транзакцій, подані як хеш-дерево [2].

Коли дані в блокчейні отримують доступ або змінюються, запис зберігається в «блоці» разом із записами інших транзакцій. Транзакції зберігаються у вигляді шифрованих записів з використанням унікальних незмінних хешів, наприклад, створених за допомогою алгоритму SHA-256. Нові блоки даних не перезаписують старі; вони додаються до ланцюжка, щоб можна було відстежувати будь-які зміни. Оскільки всі транзакції зашифровані,

записи є не змінними, тому будь-які зміни в книзі можуть бути виявлені мережею та відхилені.

Блоки зашифрованих даних назавжди «прив'язані» один до одного, а транзакції реєструються послідовно та безстроково. Таким чином створюється ідеальна історія аудиту, що дозволяє переглядати попередні версії блокчейну.

При додаванні нових даних до мережі більшість вузлів мають перевірити та підтвердити легітимність цих нових даних на основі правил або економічних стимулів, відомих також як механізми консенсусу. Після досягнення консенсусу створюється новий блок, який додається до ланцюжка. Потім усі вузли оновлюють свої дані, щоб відобразити реєстр блокчейну.

Більшість публічних блокчейнів досягають консенсусу за допомогою системи підтвердження роботи (PoW) або підтвердження частки (PoS). У системі Proof of Work перший вузол або учасник, який підтвердить додавання нових даних або транзакцію в цифровій книзі, отримує певну кількість токенів як винагороду. Щоб завершити процес перевірки, учасник або «майнер» повинен вирішити криптографічне запитання [3]. У системах Proof of Stake майнери оцінюються на основі кількості нативних монет протоколу, які вони мають у своїх цифрових гаманцях, і тривалості часу, протягом якого вони їх мали. Майнер із найбільшою кількістю монет має більше шансів бути обраним для підтвердження транзакції та отримання винагороди [3].

Щодо використання технології блокчейн, то в Україні 6 вересня 2017 року відбулася презентація першої в світі системи аукціонів, що використовує інноваційну криптографічну технологію Blockchain, яку впровадило державне підприємство «СЕТАМ» [4].

Користувачі в будь-який момент часу можуть перевірити будь-яку ставку в кожному з аукціонів у незалежного «експерта». Партнери з Фонду Східна Європа надаватимуть незалежний онлайн-сервіс, який є однією з частин блокчейн мережі і має всі дані необхідні для підтвердження перевірки. Усі бажаючі зможуть скопіювати спеціальний код ставки, перейти на сайт

blockchain.gov.ua та перевірити достовірність даних, записаних на Blockchain [4].

Існують платформи блокчейн, які описують середовище, де користувачі можуть взаємодіяти з блокчейном та його мережею. Прикладом є Ethereum – програмна платформа, яка містить криптовалюту Ethereum або ефір, де користувачі також можуть створювати програмовані токени та смарт-контракти, які побудовані безпосередньо на інфраструктурі блокчейну Ethereum.

1.2 Блокчейн-платформа Ethereum

Якщо біткойн (BTC) вважається майбутнім грошей, то що ж таке Ethereum? Це логічне запитання, особливо з урахуванням того, що Ethereum та його власну криптовалюту Ether (ETH) можна знайти поруч з біткойнами на біржах і в новинах. Однак Ethereum має різні цілі, особливості та технології.

Ethereum – це децентралізована розподілена обчислювальна платформа з відкритим кодом, яка дозволяє створювати смарт-контракти та децентралізовані програми, також відомі як DApps [5].

У 2013 році Віталік Бутерін опублікував технічний документ Ethereum, в якому детально описано смарт-контракти – автоматизовані незмінні оператори «якщо-тоді», що дозволяють розробляти децентралізовані програми [6]. Незважаючи на те, що розробка DApp вже існувала в просторі блокчейну, різні платформи не були сумісні між собою. Бутерін мав на меті об'єднати їх за допомогою Ethereum.

Так народився Ethereum 1.0. Для кращого розуміння давайте уявімо магазин додатків Apple: єдиний простір, де доступно десятки тисяч різноманітних програм, що дотримуються одних і тих самих правил. Цей набір правил жорстко закодовано в мережі та виконується автономно з розробниками, які можуть застосовувати власні правила в децентралізованих програмах. У цьому випадку немає центральної сторони, яка змінює та

впроваджує правила, як це відбувається у випадку з Apple. Замість цього влада знаходиться у руках спільноти людей, які діють разом.

Мережа Ethereum існує на тисячах комп'ютерів по всьому світу завдяки користувачам, які беруть участь як "вузли" і не являються централізованим сервером. Це надає мережі децентралізованості та захист від атак, і, по суті, робить її незаперечною. Якщо один комп'ютер вийде з ладу, це не має великого значення, оскільки тисячі інших продовжують підтримувати роботу мережі.

Ethereum – це, по суті, єдина децентралізована система, яка керує комп'ютером під назвою віртуальна машина Ethereum (EVM) [6]. Кожен вузол містить копію цієї віртуальної машини, що означає, що будь-яка взаємодія повинна бути перевірена, щоб кожен вузол зміг оновити свою власну копію.

В інших випадках мережеві взаємодії вважаються «транзакціями» та зберігаються в блоках блокчейну Ethereum [6]. Майнери перевіряють ці блоки перед передачею їх у мережу та використанням як історію транзакцій або цифрову книгу. Майнінг, необхідний для перевірки транзакцій, відомий як консенсусний метод Proof of Work. Кожен блок має унікальний 64-значний код, який його ідентифікує [6]. Майнери витрачають свої комп'ютерні потужності на пошук цього коду, що підтверджує його унікальність. Комп'ютерна потужність майнерів – «доказ» цієї роботи, за що вони отримують винагороду в ЕТН (eфір).

Усі транзакції Ethereum є повністю публічними. Майнери транслують завершені блоки решті мережі, підтверджуючи зміни та додаючи блоки до кожної копії розподіленого реєстру [6]. Підтверджені блоки неможливо підробити. Вони становлять ідеальну історію всіх мережевих транзакцій.

Але якщо майнерам платять за їхню роботу, звідки береться цей ЕТН? Кожна транзакція передбачає комісію, яка називається «газ», яку сплачує користувач, який ініціював цю транзакцію [6]. Ця комісія сплачується майнеру, який підтверджує транзакцію, і слугує стимулом для майбутнього майнінгу, забезпечуючи безпеку мережі. Газ фактично обмежує кількість

операцій, які користувач може виконати в межах транзакції. Він також призначений для запобігання мережевому спаму.

Для взаємодії з Ethereum необхідна криптовалюта, яка зберігається у гаманці. Цей гаманець підключається до децентралізованих додатків (DApps) і виступає як паспорт до екосистеми Ethereum. За його допомогою будь-хто може купувати товари, грати у ігри, надавати позики і займатися різними видами діяльності. Криптовалюта займає тут місце даних, тобто користувачі можуть вільно переглядати та взаємодіяти анонімно. Це також означає, що використання DApp є недискримінаційним. Наприклад, жоден DApp, який пропонує кредити чи банкінг, не може відмовити комусь на підставі раси чи фінансового стану. Посередник не може блокувати транзакції, які він вважає "підозрілими". Користувачі контролюють свої дії і спосіб їх здійснення, тому багато хто вважає Ethereum Web 3.0 – майбутнім веб-взаємодії.

Перехід Ethereum на протокол Proof of Stake (PoS), який дозволяє користувачам перевіряти транзакції та карбувати новий ETH на основі їхніх запасів ефіру, є частиною значного оновлення платформи Ethereum.

Останнім часом блокчейн Ethereum отримав сплеск популярності, оскільки розробники використовували його для створення низки децентралізованих фінансових проєктів і NFT. NFT – це токенизовані цифрові елементи, створені за допомогою Ethereum [7]. Загалом, токенизація полягає в присвоєнні певному цифровому активу цифрового токена, який ідентифікує його та зберігається в блокчейні. Це встановлює право власності, оскільки зашифровані дані зберігають адресу гаманця власника [7]. NFT розглядаються як транзакції в блокчейні, ними можна торгувати або продавати їх. Транзакція перевіряється мережею, і право власності передається [7]. NFT можуть бути створені для різних видів активів. Наприклад, прихильники спорту можуть придбати спортивні жетони, також відомі як фан-токени, своїх улюблених спортсменів, які можна порівняти з обмінними картками. Деякі з цих NFT є зображеннями, схожими на колекційні картки, тоді як інші містять відео пам'ятних чи історичних моментів у кар'єрі спортсмена.

Ethereum також впроваджується в ігри та віртуальну реальність. Decentraland [7] – це віртуальний світ, який використовує блокчейн Ethereum для захисту елементів, що містяться в цьому світі. Земля, аватари, пристрої, які можна носити, будівлі та середовище – усе це токенизовано через блокчейн, щоб створити право власності. CryptoKitties – онлайн-гра у формі децентралізованої програми (DApp), яка дозволяє гравцям купувати, продавати, збирати та розводити віртуальних кішок різних типів [8]. Гра використовує блокчейн Ethereum. Гравці можуть спарювати котів після покупки, за що вони платять комісію в Ethereum. Кожен кіт є унікальним і не може бути скопійованим або знищеним. Він має свої власні характеристики, які можуть бути успадковані його нащадками. При цьому використовуються стандарти технології NFT, що дозволяє експортувати об'єкти CryptoKitties до інших програм, що підтримують відповідний стандарт. Цінність кішки залежить від її характеристик і може змінюватись. Кішки з рідкісними генетичними ознаками або старістю особливо цінні. Мета гри – отримати кішок з рідкісними характеристиками.

1.3 Алгоритми безпеки блокчейн

Важливо розглянути питання безпеки блокчейну, так як смарт-контракти працюють на платформі Ethereum, яка в свою чергу заснована на технології блокчейн.

Криптографія – одна з основних вимог до блокчейну. Двома найпоширенішими типами алгоритмів, які використовуються для безпеки в блокчейні, є цифрові підписи та хешування.

Цифровий підпис – криптографічний алгоритм, працюючий на основі асиметричного шифрування [9]. Користувач вводить свій закритий (секретний) ключ для певної транзакції, щоб зашифрувати її. Одержувач міг розшифрувати цю транзакцію за допомогою відкритого ключа, який надає відправник. За рахунок цього неможливо витратити кошти користувача без

його відома, оскільки без його закритого ключа транзакція не отримає правильний підпис і внаслідок цього не буде прийнята мережею нод, що беруть участь у блокчейні [9].

Хешування – це процес перетворення вхідного масиву даних будь-якої довжини в вихідний бітовий рядок фіксованої довжини. Перед тим, як об'єднати транзакції в блоки, кожна транзакція має хешуватися. Хеш, який додається в блок, формується на основі даних, записаних у даному блоці. До хешу самого блоку додається ще й хеш попереднього блоку, в результаті чого утворюється зв'язаний ланцюжок блоків. Алгоритми хешування, які використовуються в блокчейні, гарантують так званий ефект лавини – навіть мінімальна зміна даних, що хешуються, призводить до суттєвої зміни хеша. Тому будь-яка зміна, внесена в уже виконану транзакцію, генеруватиме зовсім інший хеш, який потім змінить хеш усіх подальших блоків.

Деякі важливі риси алгоритмів хешування, завдяки яким захист блокчейну ефективен даним способом [10].

1. Алгоритми хешування не дозволяють обчислювати або виводити вхідні дані на основі вихідних даних. Важливо зазначити, що немає жодного способу повернути процес хешування, щоб переглянути вихідний набір даних.

2. Будь-які зміни у вхідних даних можуть призвести до зовсім інших результатів за допомогою алгоритмів хешування. Найменша зміна регістру певного символу в наборі даних може призвести до зовсім іншого хешу.

3. Той самий вхід завжди генерує той самий вихід із алгоритмом хешування. Незалежно від частоти проходження даних через алгоритм хешування, він завжди може створити той самий хеш з однаковими символами в рядку.

Протоколи однорангової мережі (Peer-to-Peer Network) будуть корисними для крупних корпорацій. В даному випадку величезні обсяги особистих даних або користувачів зберігаються на окремих централізованих пристроях.

Таким чином, чітко видно єдину точку відмови в разі будь-якої загрози безпеці, неправильного керування або втрати даних. Блокчейн розв'язує проблему єдиної точки відмови, чим значною мірою сприяє забезпеченню безпеки. Структура блокчейну надає можливість вузлам у системі блокчейну перевіряти автентичність транзакцій, а не третій стороні.

Кожен вузол мережі блокчейн отримує інформацію про транзакції між клієнтами. Алгоритм однорангової мережі гарантує, що кожен вузол може перевірити достовірність транзакцій перед їх записом у вигляді блоків у мережі [10].

Цей алгоритм також передбачає перегляд минулих транзакцій відправника, з метою виявлення можливих проблем з подвійними витратами. Алгоритми однорангової мережі можуть приймати протоколи угод від майнерів і досягати згоди щодо порядку та суми різних транзакцій. Після перевірки транзакція розподіляється по блокчейну у вигляді блоку [10].

Найбільш поширеними алгоритмами безпеки блокчейну є алгоритми консенсусу. Алгоритми Proof of Work і Proof of Stake розглянуті у попередніх пунктах.

2 СМАРТ-КОНТРАКТ. ПРОТИДІЯ КІБЕРАТАКАМ НА СМАРТ-КОНТРАКТИ

2.1 Смарт-контракт

Термін «розумний контракт» вперше ввів вчений в галузі інформатики та криптографії Нік Сабо на початку 1990-х років. За словами Сабо: «Я називаю ці нові контракти «розумними», тому що вони набагато функціональніші, ніж їхні неживі паперові предки. Не мається на увазі використання штучного інтелекту. Смарт-контракт – це набір обіцянок, визначених у цифровій формі, включаючи протоколи, в рамках яких сторони виконують ці обіцянки.» [11]

Смарт-контракти можуть бути «розумнішими», ніж паперові, оскільки можуть автоматично виконувати певні заздалегідь запрограмовані кроки, але їх не слід розглядати як інтелектуальні інструменти, які можуть аналізувати більш суб'єктивні вимоги контракту. Класичним прикладом «розумного контракту», який запропонував Сабо, є торговий автомат. Коли покупець виконує умови «контракту» (тобто вставляє гроші в автомат), автомат автоматично дотримується умов неписаної угоди та доставляє закуску.

Смарт-контракти – набір функцій і даних, який зберігається в блокчейні [12]. Основною метою є автоматизація виконання угоди без посередників, забезпечуючи можливість миттєвого підтвердження угоди всіма сторонами.

«Розумні контракти» це як цифрові заяви «якщо-тоді» між двома (чи більше) сторонами. Якщо потреби однієї групи задовольняються, угода може бути виконана, і контракт вважається завершеним [13].

Ідею смарт-контракту можна розбити на такі кроки.

1. Визначення сторін та встановлення умов угоди. Умови контракту, зобов'язання кожної сторони та стандарти виконання контракту описані в цій угоді.

2. Визначення умов, які мають бути виконані для виконання контракту. Ці умови, як правило, виражаються у вигляді набору правил або критеріїв, які мають бути виконані, щоб договір вважався дійсним.

3. Написання коду смарт-контракту. У ньому будуть визначені точні кроки, які необхідно виконати для виконання контракту, коли зазначені умови будуть виконані.

4. Розгортання смарт-контракту на платформі блокчейн. Це передбачає перевірку дійсності контракту шляхом завантаження коду в мережу блокчейн.

5. Автоматичний запуск виконання контракту. Коли заздалегідь визначені обставини задовольняються, контракт автоматично виконується, і мережа блокчейну активує його.

6. Інформація про контракт вводиться в мережу блокчейн під час його виконання. Це охоплює умови контракту, передумови для виконання, а також дату й час виконання. Особливості контракту залишаються незмінними після того, як вони введені в реєстр блокчейну, тобто їх не можна змінити або видалити.

Реалізація смарт-контрактів можлива в будь-якій сфері діяльності.

Платежі. Скажімо, ринок просить у фермера 5 кг полуниці. Перший зафіксує кошти в смарт-контракті, який потім можна буде схвалити, коли другий буде виконано. Коли фермер виконує свої зобов'язання, кошти будуть негайно вивільнені – тобто після виконання юридичного контракту. Однак договір скасовується, а кошти повертаються клієнту, якщо фермер пропускає свій термін.

Цифрова ідентичність. «Розумні контракти» цифрової ідентичності, побудовані на технологіях розподіленого реєстру (децентралізовані), дають людям повний контроль над своїми даними та дозволяють їм ділитися вмістом своїх даних за своїм бажанням, забезпечуючи високий рівень безпеки та знижуючи ризик неправильного управління даними або їх порушення.

Нерухомість. Уявіть ситуацію, коли документ на ваш будинок токенований в блокчейні Ethereum. Якщо ви готові продати його, ви повинні створити смарт-контракт з покупцем. Згідно цього контракту, угода буде зберігатися в умовному депонуванні до моменту, коли кошти від покупця будуть належним чином передані. Таким чином, продавець економить кошти, оскільки він не має платити посереднику, а покупець отримує доступ до будинку набагато швидше, ніж за традиційних умов.

Страховання. При укладанні страхового поліса користувач угодиться на смарт-контракт зі страховою компанією. Усі умови поліса будуть викладені в контракті, який користувач прочитає та підпише, якщо погодиться з ними. Цей договір залишатиметься відкритим, доки він не знадобиться відповідальній стороні. Потім вони просто завантажують необхідні форми, які підтверджують їхню потребу в страховій виплаті, і кошти будуть вивільнені. Такий тип договору усуває необхідність взаємодії зі страховими компаніями та окремими представниками. Хоча користувачеві все одно знадобляться документи, щоб підтвердити свої вимоги, подальше подання та процес фінансування буде майже миттєвим.

Ланцюг поставок. Припустимо, компанія в Україні хоче придбати партію товарів у постачальника в Японії. Вона може автоматизувати кожен крок транзакції, від замовлення до доставки, використовуючи смарт-контракт. Уся відповідна інформація, така як характеристики продукту, інформація про доставку, умови оплати та кінцеві терміни виконання, буде включена в контракт. Щоб гарантувати, що товари відповідають очікуванням покупця, смарт-контракт також включатиме умови щодо якості та кількості товару. Після підписання контракту гроші будуть зберігатися на умовному депонуванні, доки постачальник не підтвердить, що продукти були доставлені. Блокчейн буде відстежувати та зберігати графіки доставки та інформацію про відправлення, надаючи обом сторонам повну видимість і прозорість. Коли товар буде доставлено і покупець підтвердить, що він відповідає погодженим параметрам, смарт-контракт миттєво звільнить платежі постачальнику.

2.2 Різновид атак на смарт-контракти

Перед тим як детальніше розглянути питання захисту смарт-контрактів, проаналізуємо найтипівіші атаки, які можуть відбуватися в них.

Повторна атака. Атаки повторного входу використовують можливість смарт-контрактів в Ethereum бути викликаними іншими смарт-контрактами, а також асинхронність цих викликів [14]. Якщо функція смарт-контракту не захищена від повторного входу, зловмисник може викликати цю функцію кілька разів до завершення першого виклику, що може призвести до повторного виконання функції і потенційно змінити стан контракту непередбачуваним чином.

Наприклад, смарт-контракт, який керує балансом коштів, може бути вразливим до атак повторного входу, якщо не перевіряє стан контракту перед виконанням переказу коштів. Якщо зловмисник зможе викликати функцію переказу кілька разів до завершення першого виклику, контракт може передати більше коштів, ніж планувалося, що може призвести до втрати коштів для контракту або інших користувачів контракту.

Атака умов гонки. Умови гонки відносяться до проблеми, пов'язаної з паралелізмом, яка може виникнути в смарт-контрактах. Гонка виникає, коли дві або більше транзакції намагаються одночасно отримати доступ і змінити один і той же стан, що може призвести до непередбачуваних результатів [14].

Наприклад, розглянемо смарт-контракт, який відстежує баланс для кожного користувача. Якщо дві транзакції намагаються одночасно переказати кошти від одного користувача до іншого, і обидві транзакції зчитують поточний баланс першого користувача перед зміною, існує ризик того, що обидві транзакції використовуватимуть той самий початковий баланс. У результаті одна з транзакцій може перезаписати зміну балансу, зроблену іншою транзакцією, що призведе до некоректних балансів.

Цілочисельна атака переповнення/недоповнення.

Переповнення/недоповнення цілого числа стосується типу помилки, яка може виникнути в смарт-контрактах під час виконання арифметичних операцій над цілими числами [14,16]. Коли здійснюється операція над цілим числом, результатом якої є значення, що перевищує максимально допустиме значення, виникає переповнення. Так само, коли виконується операція над цілим числом, що призводить до значення, меншого за мінімально допустиме значення, виникає недоповнення.

Наприклад, переповнення в контракті, який керує балансом, може призвести до збереження неправильного балансу, що створює вразливість і може призвести до втрати коштів. Недостатність у контракті, який керує поставкою токенів, може призвести до створення негативних токенів, що призведе до неочікуваної поведінки та пошкодження даних.

Атака виклику делегата/проксі-дзвінка. Виклик делегування (також відомий як виклик проксі) – це функція у віртуальній машині Ethereum (EVM), яка дозволяє контракту викликати інший контракт, при цьому контракт, що викликає, фактично передає свій контекст виконання викликаному контракту [15,16]. Атака виклику делегата відбувається, коли зловмисник користується функцією виклику делегата для зміни поведінки смарт-контракту. Зловмисник може змінити поведінку викликаного контракту, надаючи шкідливі вхідні дані або змінюючи код викликаного контракту.

Наприклад, розглянемо контракт, який керує системою голосування. Якщо контракт використовує виклики делегатів для виклику іншого контракту для підрахунку голосів, зловмисник може ввести шкідливі вхідні дані в виклик делегатів, щоб маніпулювати підрахунком голосів. Як наслідок, підробка результатів системи голосування та втрата довіри до системи.

Відмова в обслуговуванні (DoS). Тип кібератаки, при якій зловмисник намагається навмисно перевантажити або вивести з ладу систему або мережу, зробивши їх недоступними для користувачів. Метою DoS-атаки є заважання законним користувачам отримати доступ до цільової системи або мережі

шляхом перевантаження їх надмірним трафіком або використанням вразливостей у системі.

У контексті смарт-контрактів DoS-атака на контракт може бути здійснена шляхом повторного надсилання транзакцій [15], які вимагають великих обчислювальних ресурсів, що призводить до перевантаження мережі. Також атака може бути здійснена шляхом використання вразливостей у коді контракту, що дозволяють зловмиснику зруйнувати контракт.

Атака випередження. Тип атаки, яка використовує переваги децентралізованої та прозорої природи систем блокчейн. Це стає можливим, коли зловмисник перехоплює транзакцію, що підтверджується мережею, і швидко виконує власну транзакцію з тим самим активом, використовуючи зміну ринкових умов, що виникла в результаті початкової транзакції [16].

У контексті смарт-контрактів, така атака може статися, коли зловмисник перехоплює транзакцію, яку смарт-контракт планує виконати, і швидко виконує свою власну транзакцію з тими ж активами, використовуючи умови, створені початковою транзакцією.

Наприклад, якщо користувач планує укласти угоду на децентралізованій біржі, зловмисник може перехопити транзакцію, виконати власну угоду з тими ж активами і скористатися зміною ринкових умов, що виникла в результаті початкової угоди.

Викрадення права власності на контракт. Тип атаки на смарт-контракт, у якій зловмисник бере на себе право власності на контракт, фактично надаючи йому контроль над його операціями [16]. Це можливо, якщо код контракту містить уразливості, що дозволяють зловмиснику змінити адресу, що зберігається як власник контракту, надаючи йому можливість виконувати будь-які функції в межах контракту.

Після того, як зловмисник отримає право власності на контракт, він може виконати шкідливий код, викрасти збережені в контракті кошти або іншим чином маніпулювати функціями контракту. У деяких випадках вони

також можуть змінити код контракту, щоб видалити будь-які докази атаки та ускладнити виявлення такої атаки.

Атака з витоків інформації. Ці атаки можуть відбуватися, коли код контракту містить уразливості, які дозволяють зловмиснику отримати доступ і викрасти конфіденційну інформацію, таку як особисті ключі, паролі або конфіденційні бізнес-дані [16]. Серед поширених способів витоків інформації:

- витоків журналів: зловмисники можуть використовувати функцію журналювання контракту, щоб отримати конфіденційну інформацію;
- виконання коду: зловмисники можуть запустити зловмисний код у контракті, що дозволяє їм отримати доступ до конфіденційної інформації та отримати її;
- часові атаки: зловмисники можуть використовувати інформацію про час, щоб отримати конфіденційну інформацію, наприклад закриті ключі.

Атаки на залежність від упорядкування транзакцій. Це тип атаки на блокчейн, який використовує залежність деяких смарт-контрактів від порядку обробки транзакцій [16,17]. Під час такої атаки зловмисник маніпулює порядком обробки транзакцій, щоб досягти зловмисного результату.

Наприклад, розглянемо смарт-контракт, який дозволяє користувачам робити ставки на майбутню подію. Якщо контракт обробляє транзакції в певному порядку, зловмисник може зробити ставку після того, як результат події стане відомим, і маніпулювати контрактом, щоб виплатити більше, ніж вони повинні були.

2.3 Заходи щодо захисту смарт-контрактів

Публічні блокчейни, такі як Ethereum, створюють складнощі в області захисту смарт-контрактів. Код контракту, який розгортається, зазвичай не можна змінити для виправлення потенційних уразливостей безпеки. Водночас, активи, які можуть бути викрадені зі смарт-контрактів, дуже складно

відстежити, і часто неможливо повністю відновити через характер незмінності блокчейну. Створення безпечних смарт-контрактів Ethereum виконується наступними засобами.

Належний контроль доступу. У смарт-контрактах функції можуть мати різні рівні видимості, такі як публічний або зовнішній, і вони можуть бути викликані зовнішніми обліковими записами або контрактними обліковими записами. Встановлення правильного рівня видимості для функцій є необхідним, якщо потрібно, щоб інші взаємодіяли з вашим контрактом. Однак функції, що позначені як приватні, можуть бути викликані лише іншими функціями в межах смарт-контракту. Надання доступу кожному учаснику мережі до всіх контрактних функцій може створювати проблеми, особливо якщо це означає, що будь-хто може виконувати конфіденційні операції без необхідних дозволів.

Щоб запобігти несанкціонованому використанню функцій смарт-контракту, необхідно впровадити безпечний контроль доступу [18,22].

Модель власності. Під час створення контракту адреса встановлюється як "власник". Захищеним функціям присвоюється модифікатор `OnlyOwner`, який гарантує, що контракт перевіряє автентичність адреси, що викликає функцію, перед її виконанням. Виклики захищених функцій з інших адрес, крім власника контракту, завжди повертаються, що запобігає небажаному доступу.

Контроль доступу на основі ролей. При управлінні доступом на основі ролей, доступ до конфіденційних функцій розподіляється між набором довірених учасників. Наприклад, один обліковий запис може бути відповідальним за видобуток токенів, а інший – за оновлення або призупинення контракту. Децентралізація контролю доступу таким чином усуває окремі точки відмов та зменшує необхідність довіри до користувачів.

Використання гаманців з кількома підписами. Облікові записи з декількома підписами належать різним організаціям і для виконання транзакцій потрібні підписи мінімальної кількості облікових записів,

наприклад, 4 з 6. Використання гаманців з кількома підписами для контролю доступу вводить додатковий рівень безпеки, оскільки дії з цільовим контрактом вимагають згоди кількох сторін.

Оператори `require()`, `assert()` і `revert()` для захисту контрактних операцій. Як вже згадувалося, після розгортання смарт-контракту в блокчейні будь-хто може викликати його загальнодоступні функції. Оскільки ми не можемо передбачити, як зовнішні облікові записи будуть взаємодіяти з контрактом, було б ідеально мати внутрішні засоби захисту від проблемних операцій ще до розгортання контракту. Можна забезпечити правильну поведінку в смарт-контрактах за допомогою операторів `require()`, `assert()` і `revert()`, щоб ініціювати винятки та скасовувати зміни стану, якщо виконання не відповідає певним вимогам:

- `require()`: вимоги визначаються на початку функцій і забезпечують виконання попередньо визначених умов перед виконанням викликаної функції [19];
- `assert()`: використовується для виявлення внутрішніх помилок і перевірки на порушення «інваріантів» у коді [19]. Інваріант – це логічне твердження про стан контракту, яке має бути істинним для виконання всіх функцій [22]. Прикладом інваріанта є максимальна загальна пропозиція або баланс контракту на токени;
- `revert()`: можна використовувати в операторі `if-else`, який запускає виняток, якщо необхідна умова не задовольняється.

Тестування смарт-контракту та перевірка правильності коду. Один зі стандартних методів – написання невеликих модульних тестів, використовуючи фальшиві дані, які контракт повинен отримати від користувачів. Модульне тестування добре підходить для перевірки функціональності конкретних функцій та забезпечення правильної роботи смарт-контракту.

Формальна верифікація – ще один метод перевірки безпекових властивостей у смарт-контрактах. У порівнянні зі звичайним тестуванням, формальна верифікація може остаточно довести відсутність помилок у смарт-контракті. Це досягається шляхом створення формальної специфікації, яка фіксує необхідні безпекові властивості, та доведенням того, що формальна модель контракту відповідає цій специфікації [20].

Після перевірки контракту можна попросити інших перевірити вихідний код на наявність проблем із безпекою. Замовлення аудиту смарт-контракту є одним із способів проведення незалежної перевірки коду.

Налаштування програми винагороди за помилки є ще одним підходом для реалізації зовнішніх перевірок коду [22]. Винагорода за помилки – це фінансова винагорода, яка надається окремим особам (зазвичай хакерам), які виявляють вразливі місця в програмі.

Надійні плани аварійного відновлення. Належний план аварійного відновлення включатиме деякі або всі наступні компоненти [21]:

- оновлення контракту;
- аварійні зупинки.

Оновлення контракту можливе навіть у незмінних за замовчуванням смарт-контрактах Ethereum шляхом використання шаблонів оновлення. Механізми оновлення контрактів можуть відрізнятися, але одним з найпоширеніших підходів є використання шаблону проксі. [22] У шаблоні проксі стан та логіка контракту розділяються між двома контрактами. Перший контракт, відомий як "проксі-контракт", зберігає змінні стану (наприклад, баланси користувачів), а другий контракт, відомий як "логічний контракт", містить код для виконання функцій контракту.

Облікові записи взаємодіють з проксі-контрактом, який передає всі виклики функцій до логічного контракту за допомогою `delegatecall()`, низькорівневої функції. [21,22] На відміну від звичайного виклику повідомлення, `delegatecall()` гарантує, що код, який виконується за адресою

логічного контракту, виконується в контексті проксі-контракту. Це означає, що логічний контракт завжди записується в сховище проксі-контракту (замість власного сховища), а значення `msg.sender` і `msg.value` зберігаються.

Для делегування викликів логічному контракту, адресу останнього потрібно зберігати в сховищі проксі-контракту [21]. Тому оновлення логіки контракту полягає лише у розгортанні нового логічного контракту та збереженні його нової адреси в проксі-контракті. Оскільки наступні виклики проксі-контракту автоматично направляються до нового логічного контракту, ви фактично "оновлюєте" контракт без зміни його коду.

Аварійні зупинки. Як уже згадувалося, обширний аудит і тестування не можуть виявити всі помилки в смарт-контракті. Якщо після розгортання в контракті виявляється вразливість, то виправити її неможливо, оскільки код, який виконується, не можна змінити. Крім того, реалізація механізмів оновлення (наприклад, шаблонів проксі) може зайняти час, оскільки вони часто потребують схвалення з різних сторін. Це лише дає зловмисникам більше часу для завдання шкоди і ускладнює ситуацію.

Аварійна зупинка зазвичай складається з таких компонентів [22].

1. Глобальна логічна змінна, що вказує, чи смарт-контракт перебуває в зупиненому стані чи ні. Ця змінна має значення `false` під час налаштування контракту, але повернеться до значення `true` після припинення контракту.

2. Функції, які під час виконання посилаються на логічну змінну. Такі функції доступні, коли смарт-контракт не зупинено, і стають недоступними, коли спрацьовує функція аварійної зупинки.

3. Сутність, яка має доступ до функції аварійної зупинки, яка встановлює для логічної змінної значення `true`. Щоб запобігти зловмисним діям, виклики цієї функції можна обмежити довіреною адресою (наприклад, власником контракту).

Після того, як контракт активує аварійну зупинку, певні функції будуть недоступні. Це досягається шляхом обгортання функцій `select` у модифікатор, який посилається на глобальну змінну.

Моніторинг подій. Події дозволяють відстежувати виклики функцій смарт-контракту та відстежувати зміни змінних стану [22]. Рекомендується програмувати свій смарт-контракт так, щоб він відправляв подію кожного разу, коли сторона здійснює критично важливу дію з погляду безпеки (наприклад, виведення коштів).

Реєстрація подій та їх моніторинг поза ланцюгом дозволяють аналізувати контрактні операції та швидше виявляти зловмисну активність. Це дозволяє швидко реагувати на хакерські атаки та вживати заходи для мінімізації впливу на користувачів, наприклад, призупиняти функції або здійснювати оновлення.

Також можна скористатися готовими інструментами моніторингу, які автоматично надсилають сповіщення при взаємодії з контрактами. Ці інструменти дозволяють створювати спеціальні повідомлення на основі різних тригерів, таких як обсяг транзакцій, частота викликів функцій або конкретні функції, що використовуються. Наприклад, можна налаштувати сповіщення, яке буде активуватися, коли сума, вилучена в одній транзакції, перевищує певний поріг.

Зменшення складності коду до мінімуму. Збереження простоти є особливо важливим аспектом при написанні смарт-контрактів, особливо з урахуванням того, що вони мають потенційно керувати значними обсягами цінності. Рекомендацією для досягнення простоти в написанні смарт-контрактів є використання наявних бібліотек, таких як контракти OpenZeppelin, де це можливо. Оскільки ці бібліотеки були ретельно перевірені та протестовані розробниками, використання їх допомагає зменшити ймовірність помилок, що виникають при створенні нових функцій з нуля.

Ще однією поширеною порадою є написання невеликих функцій та збереження модульності контрактів шляхом розділення бізнес-логіки на декілька контрактів. Написання простого коду не лише зменшує потенційні вразливості смарт-контракту, але також полегшує оцінку правильності загальної системи та раннє виявлення можливих помилок дизайну.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СМАРТ-КОНТРАКТУ ДЛЯ ОПЛАТИ ПОСЛУГ

3.1 Мова програмування Solidity

Для реалізації смарт-контракту обираємо мову програмування Solidity. Solidity – це об'єктно-орієнтована мова, за допомогою якої здійснюється програмування смарт-контрактів для блокчейн-платформи Ethereum [23].

Під час роботи над блокчейном, команда Ethereum почала розробку мови Solidity. Створюючи її, розробники були натхненні вже відомими мовами програмування, а саме Python, C++ і Javascript. Solidity була розроблена з урахуванням унікальних можливостей, які надає блокчейн Ethereum, включаючи можливість створення цифрових активів, використання токенів та управління децентралізованими програмами [24].

Мова є статично типізованою. Це означає, що перевірка типів виконується під час компіляції, а не під час виконання, як у випадку з динамічно типізованими мовами. Працюючи зі статично типізованою мовою необхідно вказувати тип кожної змінної.

Solidity підтримує успадкування, завдяки чому функції, змінні та інші властивості, що є в одному контракті, можна використовувати в інших.

Мова підтримує складні типи, що визначаються користувачем, такі як структури і перерахування, за допомогою яких можна групувати зв'язані типи даних.

Solidity – це мова програмування з відкритим вихідним кодом з спільнотою учасників спільної роботи, що активно зростає.

Контракти Solidity виконуються на віртуальній машині Ethereum (EVM). Така машина є повністю ізольованим середовищем. Вона працює тільки з контрактами, що виконуються на ній, і не здійснює доступ до будь-яких інших мережних ресурсів.

3.2 UML-діаграма системи оплати

Розглядається система оплати, яка має двох акторів, один з яких є споживачем електроенергії, інший – підприємство, яке надає відповідні послуги. Кожен з цих акторів взаємодіє з системою.

Базовими варіантами використання є "Надання послуг населенню" та "Оплата за електроенергію". Для виконання другого варіанту використання, система звертається до додаткових сервісів "Розрахунок платежу за показаннями" та "Проведення платежу". На рисунку 3.1 наведено варіанти використання системи оплати.



Рисунок 3.1 – Діаграма варіантів користування для системи оплати послуг

На наступному етапі розробки моделі варіантів використання системи, що розглядається, напишемо текстовий сценарій. Цей сценарій буде доповнювати діаграму, розкриваючи зміст окремих дій, які виконуються системою та акторами в процесі оплати послуг. В даному випадку сценарій зручно подати у вигляді трьох окремих таблиць, кожна з яких описує окремий розділ.

В головному розділі сценарію (табл. 3.1) вказується ім'я варіанту використання, який розглядається, імена взаємопов'язаних з ним акторів, ціль виконання варіанта, посилання на інші варіанти використання.

Таблиця 3.1 – Головний розділ

Варіант використання	Оплата за електроенергію
Актори	Споживач, Підприємство
Ціль	Проведення успішного платежу
Короткий опис	Споживач робить запит на отримання послуг. Після сплатує за обсяг послуг, які використав.
Посилання на інші варіанти використання	Включає в себе варіанти використання: – розрахунок платежу за показаннями; – проведення платежу.

В наступному розділі сценарію (табл. 3.2) йде опис послідовності дій, що приводить до успішного виконання варіанту використання, що розглядається. Ініціатором дій виступає один з акторів, в даному випадку – Споживач. Для зручності наступних посилань кожна дія позначається порядковим номером в послідовності.

Таблиця 3.2 – Розділ "Типовий хід подій"

Типовий хід подій	
Дія акторів	Відгук системи
1. Запит на отримання послуг в сфері електроенергії.	2. Обробка запиту. 3. Надання послуги в сфері електроенергії.
4. Користування послугою.	5. Обробка значень, отриманих від користувача. 6. Розрахунок платежу за значеннями.
7. Проведення платежу. Виняток №1: Недостатньо грошей на рахунку споживача.	8. Обробка отриманого платежу.

В третьому розділі сценарію (табл. 3.3) йде опис послідовності дій, які виконуються при виняткових ситуаціях (винятків).

Таблиця 3.3 – Розділ "Винятки"

Дія актора	Відгук системи
Виняток №1. Недостатньо грошей на рахунку споживача	
1. Платіж не проводиться через недостатню кількість грошей на рахунку споживача.	2. Обробка даних. 3. Внесення суми боргу в наступний платіж.
4. Проведення платежу з погашенням боргу.	5. Обробка отриманого платежу.

3.3 Реалізація та опис коду смарт-контракту

Маємо сценарій, де є споживач, який користується послугами в сфері електроенергії, і є підприємство, яке надає відповідні послуги. Треба, щоб споживач здійснив оплату за надані послуги. Для цього створимо смарт-контракт для системи оплати послуг.

Створюємо смарт-контракт, який має назву `EnergyPayment.sol`. Для початку вказуємо ліцензію та версію Solidity, яку буде використовувати компілятор.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;
```

```
contract EnergyPayment {
```

```
    address private owner;
    uint public rate;
```

Оголошуємо контракт під назвою `"EnergyPayment"`. В ньому створюємо змінні стану. Змінна `rate` буде зберігати вартість одиниці енергії. Ця змінна має видимість `public`, так як нам треба, щоб обидві сторони могли бачити вартість за одиницю енергії.

Також ми визначаємо змінну стану, `owner`, яка буде зберігати адресу власника контракту (підприємства). Ця змінна має видимість `private`, тобто доступ до неї можна отримати тільки з поточного смарт-контракту.

Оголошуємо структуру "Consumption", яка містить поля для зберігання інформації про спожиту енергію.

```
struct Consumption {
    uint amount;
    uint cost;
    bool paid;
}
```

Структура містить наступні поля:

- "amount" – кількість спожитої енергії;
- "cost" – вартість спожитої енергії;
- "paid" – прапорець, який вказує, чи було оплачено спожиту енергію.

Далі прописуємо рядок,

```
mapping(address => Consumption[]) public consumptions;
```

де оголошуємо mapping з назвою "consumptions", який відображає адреси користувачів на масиви структур "Consumption"; mapping використовується для збереження записів про споживання енергії кожного користувача.

Далі створюємо конструктор контракту.

```
constructor(uint _rate) {
    owner = msg.sender;
    rate = _rate;
}
```

Конструктор встановлює власника контракту (адресу того, хто розгорнув контракт) і початкове значення "rate" (вартість одиниці енергії).

Щоб запобігти несанкціонованому використанню функцій смарт-контракту, впроваджуємо безпечний контроль доступу. Використаємо модифікатор OnlyOwner. Він гарантує, що контракт перевіряє автентичність адреси виклику перед виконанням функції.

```

modifier onlyOwner {
    require(msg.sender == owner, "Caller is not owner");
    _;
}

```

В модифікаторі виконуємо операцію, яка є перевіркою того, що саме власник контракту викликає функцію. Для цього ми використовуємо функцію `require`, яка перевіряє наявність умов та видає виняток, якщо умова не виконується.

Визначаємо функцію `changeRate`. Вона дозволяє власнику контракту змінювати вартість одиниці енергії.

```

function changeRate(uint _rate) public onlyOwner {
    rate = _rate;
}

```

Далі визначаємо функцію `recordConsumption`, яка записує інформацію про споживання енергії.

```

function recordConsumption(uint _amount) public {
    Consumption memory newConsumption;
    newConsumption.amount = _amount;
    newConsumption.cost = _amount * rate;
    newConsumption.paid = false;

    consumptions[msg.sender].push(newConsumption);
}

```

Вона приймає параметр `"_amount"` (кількість спожитої енергії) і створює новий об'єкт типу `"Consumption"` з відповідними значеннями полів. Потім цей об'єкт додається до масиву `"consumptions"` для відповідного користувача (адреси виклику).

Визначаємо наступну функцію `payConsumption`. Вона дозволяє споживачу оплатити енергію, яку він спожив.

```

function payConsumption(uint _index) public payable {
    require(!consumptions[msg.sender][_index].paid,
"Consumption already paid");
    require(msg.value ==
consumptions[msg.sender][_index].cost, "Incorrect value transferred");

    consumptions[msg.sender][_index].paid = true;
}

```

Вона приймає параметр "_index", який вказує на індекс споживання в масиві "consumptions" для відповідного користувача. Функція перевіряє, чи споживання не було вже оплачено і чи була передана правильна сума грошей. Якщо перевірка пройдена, то відповідне поле "paid" у структурі "Consumption" встановлюється в значення "true".

Визначаємо останню функцію withdrawFunds.

```

function withdrawFunds(uint _amount) public onlyOwner {
    payable(owner).transfer(_amount);
}
}

```

Ця функція дозволяє власнику контракту зняти кошти з контракту. Вона приймає параметр "_amount", який вказує на кількість коштів, що мають бути зняті. Функція переказує вказану суму коштів з контракту на адресу власника контракту.

Даний смарт-контракт дозволяє вести облік споживаної енергії і оплачувати її. Користувачі (споживачі) можуть побачити вартість за одиницю енергії, записати показання (скільки вони використали енергії), а потім оплатити. Власник контракту (підприємство) може змінювати вартість одиниці енергії і знімати накопичені кошти з контракту.

Для того, щоб реалізувати смарт-контракт, треба його розгорнути. В Remix IDE заходимо в Deploy & run transactions і натискаємо на кнопку Deploy (рис. 3.2). Також вказуємо вартість одиниці енергії.

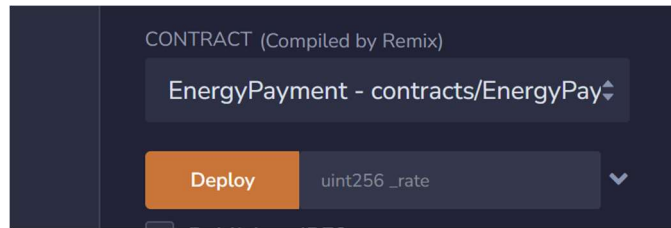


Рисунок 3.2 – Розгортка смарт-контракту

Після чого переходимо на онлайн-сервіс (рис.3.3), де бачимо вартість одиниці енергії, а також поле, де можемо ввести кількість спожитої енергії. Після введення натискаємо "Записати споживання" і бачимо вартість енергії, яку спожили. Проводимо оплату, натискаючи кнопку "Оплатити".

Рисунок 3.3 – Інтерфейс системи оплати

Заходимо в крипто гаманець, де бачимо суму спожитої енергії, а також комісію за дану транзакцію (рис. 3.4). Підтверджуємо платіж та очікуємо результату.

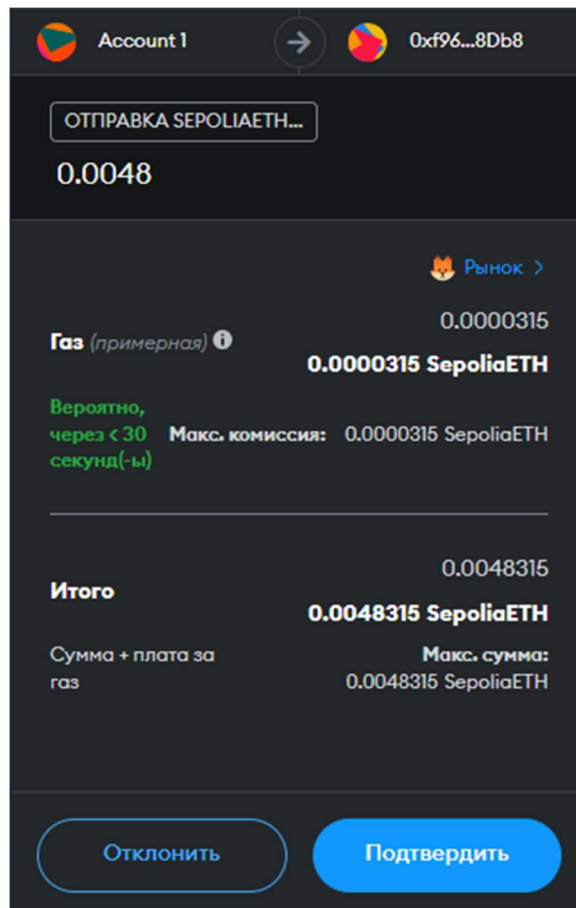


Рисунок 3.4 – Крипто гаманець MetaMask

Після чого на сторінці з'являється вікно про успішну оплату послуг (рис.3.5).

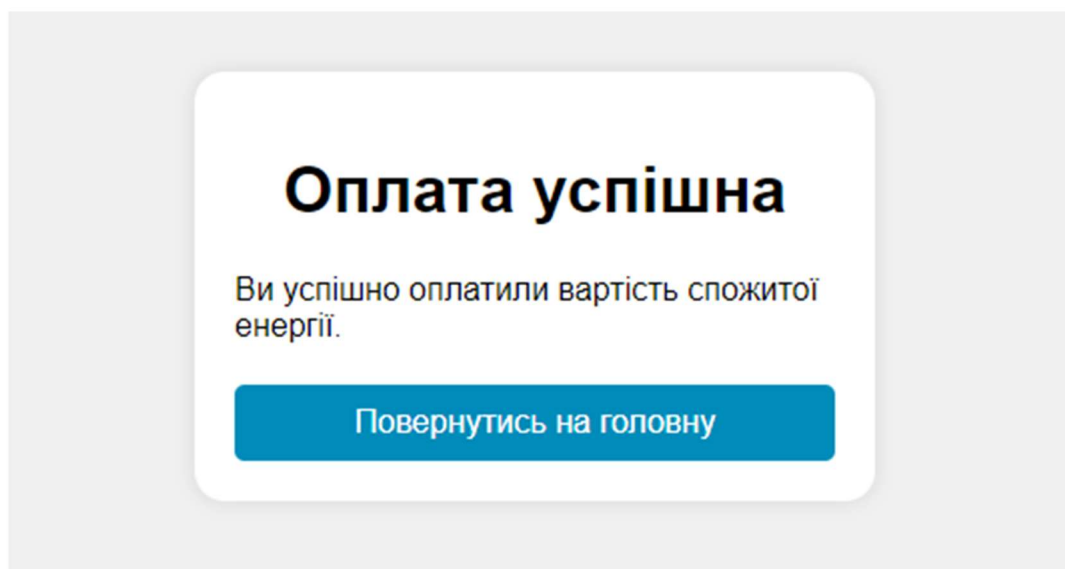


Рисунок 3.5 – Результат платежу

ВИСНОВКИ

В ході дипломної роботи було розглянуто технологію Blockchain та платформу Ethereum, яка на ній побудована. Було розглянуто різні алгоритми безпеки блокчейн, а саме цифровий підпис та хешування; протоколи однорангової мережі (Peer-to-Peer Network); та алгоритми консенсусу, а саме доказ роботи (Proof of Work) та доказ частки (Proof of Stake).

Було розглянуто та досліджено поняття та ідею смарт-контрактів. Проведено аналіз типових кібератак, які можуть виникнути під час роботи з розумними контрактами. Після чого розглянуто заходи протидії вразливостям притаманним смарт-контрактам.

Для написання контракту було обрано мову програмування Solidity та середовище розробки Remix IDE. Було розроблено систему оплати комунальних послуг на основі смарт-контракту. Також було розроблено інтерфейс, через який можна буде взаємодіяти із смарт-контрактом на основі мови HTML та CSS. Завдяки розумному контракту споживачі бачать вартість за одиницю енергії, записують свої показання, а потім сплачують. Підприємство за потреби змінює вартість одиниці енергії і знімає накопичені кошти з контракту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. «Все, що вам потрібно знати про технологію Block Chain» Finance de Demain Consulting – Електронний ресурс – Режим доступу: <https://financededemain.com/uk/tout-savoir-sur-la-technologie-block-chain/?noamp=available> (дата звернення 14.04.2023)
2. «Blockchain» Synopsys – Електронний ресурс – Режим доступу: <https://www.synopsys.com/glossary/what-is-blockchain.html> (дата звернення 14.04.2023)
3. «What is proof of stake?» McKinsey & Company – Електронний ресурс – Режим доступу: <https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-proof-of-stake> (дата звернення 14.04.2023)
4. «СЕТАМ СТАВ ПЕРШИМ У СВІТІ АУКЦІОНОМ НА BLOCKCHAIN ТА ЗМІНИВ НАЗВУ НА OPENMARKET» СЕТАМ – Електронний ресурс – Режим доступу: <https://setam.gov.ua/article/setam-stav-pershim-u-sviti-auksionom-na-blockchain-ta-zminiv-nazvu-na-openmarket> (дата звернення 15.04.2023)
5. «What is Ethereum?» ConsenSys – Електронний ресурс – Режим доступу: <https://consensys.net/knowledge-base/ethereum/> (дата звернення 20.04.2023)
6. «What is Ethereum and how does it work?» COINTELEGRAPH – Електронний ресурс – Режим доступу: <https://cointelegraph.com/learn/what-is-ethereum-a-beginners-guide-to-eth-cryptocurrency> (дата звернення 20.04.2023)
7. «What Is Ethereum and How Does It Work?» Investopedia – Електронний ресурс – Режим доступу: <https://www.investopedia.com/terms/e/ethereum.asp> (дата звернення 22.04.2023)
8. «CryptoKitties» Вікіпедія – Електронний ресурс – Режим доступу: <https://uk.wikipedia.org/wiki/CryptoKitties> (дата звернення 22.04.2023)

9. «Алгоритмы поддержки безопасности блокчейн» Habr – Электронный ресурс – Режим доступа: <https://habr.com/ru/articles/595621/> (дата звернения 22.05.2023)
10. «Blockchain Security Algorithms Used To Protect The Blockchain Security» 101 Blockchains – Электронный ресурс – Режим доступа: <https://101blockchains.com/blockchain-security-algorithms/> (дата звернения 22.05.2023)
11. Nick Szabo «Smart Contracts: Building Blocks for Digital Markets» – Электронный ресурс – Режим доступа: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html (дата звернения 28.04.2023)
12. «Що таке смарт-контракти та як вони працюють» ForkLog – Электронный ресурс – Режим доступа: <https://forklog.com/sp/dev-on-tezos/ua/tezos-introduction/> (дата звернения 28.04.2023)
13. «What is a smart contract, and how does it work?» Cointelegraph – Электронный ресурс – Режим доступа: <https://cointelegraph.com/learn/what-are-smart-contracts-a-beginners-guide-to-automated-agreements> (дата звернения 28.04.2023)
14. «Most Common Smart Contract Attacks» Hacken – Электронный ресурс – Режим доступа: <https://hacken.io/discover/most-common-smart-contract-attacks/> (дата звернения 03.05.2023)
15. «Security Issues with Delegate Calls» Medium – Электронный ресурс – Режим доступа: <https://blog.solidityscan.com/security-issues-with-delegate-calls-4ae64d775b76> (дата звернения 03.05.2023)
16. «Various types of attacks on Smart Contracts» Linkedin – Электронный ресурс – Режим доступа: <https://www.linkedin.com/pulse/various-types-attacks-smart-contracts-amit-nadiger> (дата звернения 06.05.2023)
17. «Known Attacks» GitHub – Электронный ресурс – Режим доступа: <https://ethereum-contract-security-techniques-and->

tips.readthedocs.io/en/latest/known_attacks/#transaction-ordering-dependence-tod-front-running (дата звернення 10.05.2023)

18. «Access Restriction Patterns» Medium – Електронний ресурс – Режим доступу: <https://medium.com/coinmonks/access-restriction-patterns-ee1736628a44> (дата звернення 10.05.2023)

19. «Require vs Assert in Solidity» DEV – Електронний ресурс – Режим доступу: <https://dev.to/tawseef/require-vs-assert-in-solidity-5e9d> (дата звернення 10.05.2023)

20. «How to Test Smart Contracts on the Blockchain» interexy – Електронний ресурс – Режим доступу: <https://interexy.com/how-to-test-smart-contracts-on-the-blockchain/> (дата звернення 10.05.2023)

21. «6 Solidity Smart Contract Security Best Practices» alchemy – Електронний ресурс – Режим доступу: <https://www.alchemy.com/overviews/smart-contract-security-best-practices> (дата звернення 10.05.2023)

22. «SMART CONTRACT SECURITY» – Електронний ресурс – Режим доступу: <https://ethereum.org/en/developers/docs/smart-contracts/security/> (дата звернення 10.05.2023)

23. «Что такое Solidity» Microsoft – Електронний ресурс – Режим доступу: <https://learn.microsoft.com/ru-ru/training/modules/blockchain-learning-solidity/2-what-is-solidity> (дата звернення 19.05.2023)

24. «Язык программирования Solidity» Solidity University – Електронний ресурс – Режим доступу: <https://solidity.university/solidity/> (дата звернення 19.05.2023)

ДОДАТОК А

Код реалізації смарт-контракту для системи оплати

EnergyPayment.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;

contract EnergyPayment {

    address private owner;
    uint public rate; // вартість одиниці енергії

    struct Consumption {
        uint amount; // кількість спожитої енергії
        uint cost; // вартість спожитої енергії
        bool paid; // чи оплачено споживання
    }

    mapping(address => Consumption[]) public consumptions;

    constructor(uint _rate) {
        owner = msg.sender;
        rate = _rate;
    }

    modifier onlyOwner {
        require(msg.sender == owner, "Caller is not owner");
        _;
    }

    // Зміна вартості одиниці енергії
    function changeRate(uint _rate) public onlyOwner {
        rate = _rate;
    }

    // Запис спожитої енергії
    function recordConsumption(uint _amount) public {
        Consumption memory newConsumption;
        newConsumption.amount = _amount;
        newConsumption.cost = _amount * rate;
        newConsumption.paid = false;

        consumptions[msg.sender].push(newConsumption);
    }

    // Оплата спожитої енергії
    function payConsumption(uint _index) public payable {
        require(!consumptions[msg.sender][_index].paid, "Consumption
already paid");
    }
}
```

```
        require(msg.value == consumptions[msg.sender][_index].cost,  
"Incorrect value transferred");  
  
        consumptions[msg.sender][_index].paid = true;  
    }  
  
    // Вивести кошти власнику  
    function withdrawFunds(uint _amount) public onlyOwner {  
        payable(owner).transfer(_amount);  
    }  
}
```

ДОДАТОК Б

Код реалізації графічного інтерфейсу

Index.html (основна сторінка)

```
<!DOCTYPE html>
<html>
<head>
  <title>Енергетична платіжна система</title>
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
  <div class="container">
    <h1>Енергетична платіжна система</h1>
    <p>Вартість одиниці енергії: <span id="rate"></span> wei</p>
    <p>Введіть кількість спожитої енергії:</p>
    <input id="amount" type="number" min="0">
    <p id="cost"></p>
    <button id="submit">Записати споживання</button>
    <button id="pay">Оплатити</button>
  </div>
  <script
src="https://cdn.jsdelivr.net/npm/web3@1.2.9/dist/web3.min.js"></script>
  <script src="app.js"></script>
</body>
</html>
```

Styles.css (файл стилів)

```
body {
  font-family: Arial, sans-serif;
  background-color: #f0f0f0;
  padding: 30px;
}
```

```
.container {
  width: 300px;
  margin: auto;
  background-color: white;
  padding: 20px;
  border-radius: 15px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
```

```
h1 {
  text-align: center;
}
```

```
button {
  display: block;
  width: 100%;
  padding: 10px;
  margin-top: 20px;
  font-size: 16px;
  background-color: #008CBA;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}
```

```
button:hover {
  background-color: #007B9A;
}
```

App.js (головний скрипт роботи сторінки)

```
// Replace with your contract's ABI
var abi = [
  {
    "inputs": [
```

```

        {
            "internalType": "uint256",
            "name": "_rate",
            "type": "uint256"
        }
    ],
    "stateMutability": "nonpayable",
    "type": "constructor"
},
{
    "inputs": [
        {
            "internalType": "uint256",
            "name": "_rate",
            "type": "uint256"
        }
    ],
    "name": "changeRate",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "address",
            "name": "",
            "type": "address"
        },
        {
            "internalType": "uint256",
            "name": "",
            "type": "uint256"
        }
    ],
    "name": "consumptions",
    "outputs": [

```

```

        {
            "internalType": "uint256",
            "name": "amount",
            "type": "uint256"
        },
        {
            "internalType": "uint256",
            "name": "cost",
            "type": "uint256"
        },
        {
            "internalType": "bool",
            "name": "paid",
            "type": "bool"
        }
    ],
    "stateMutability": "view",
    "type": "function"
},
{
    "inputs": [],
    "name": "owner",
    "outputs": [
        {
            "internalType": "address",
            "name": "",
            "type": "address"
        }
    ],
    "stateMutability": "view",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "uint256",
            "name": "_index",

```

```

        "type": "uint256"
    }
],
"name": "payConsumption",
"outputs": [],
"stateMutability": "payable",
"type": "function"
},
{
    "inputs": [],
    "name": "rate",
    "outputs": [
        {
            "internalType": "uint256",
            "name": "",
            "type": "uint256"
        }
    ],
    "stateMutability": "view",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "uint256",
            "name": "_amount",
            "type": "uint256"
        }
    ],
    "name": "recordConsumption",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {

```

```

        "internalType": "uint256",
        "name": "_amount",
        "type": "uint256"
    }
],
"name": "withdrawFunds",
"outputs": [],
"stateMutability": "nonpayable",
"type": "function"
}
];

// Replace with your contract's address
var contractAddress = '0xdaCdF1E7C90CA1732991A60Ca93851a3397b6Ec5';

var web3 = new Web3(Web3.givenProvider ||
"https://mainnet.infura.io/v3/08b2a484451e4635a28b3d8234f24332");

var contract = new web3.eth.Contract(abi, contractAddress);

var rate;

contract.methods.rate().call().then(function(result) {
    rate = result;
    document.getElementById('rate').innerHTML = rate;
});

document.getElementById('amount').addEventListener('input', function()
{
    var amount = document.getElementById('amount').value;
    document.getElementById('cost').innerHTML = 'Вартість спожитої
енергії: ' + (amount * rate) + ' wei';
});

document.getElementById('submit').addEventListener('click', function()
{
    var amount = document.getElementById('amount').value;

```

```

        contract.methods.recordConsumption(amount).send({from:
web3.eth.defaultAccount})
        .on('receipt', function(receipt) {
            document.getElementById('cost').innerHTML = 'Вартість спожитої
енергії: ' + (amount * rate) + ' wei';
        })
        .on('error', console.error);
    });

    document.getElementById('pay').addEventListener('click', function() {
        var cost = document.getElementById('cost').innerHTML.split(' ')[3];

        contract.methods.payConsumption(0).send({from:
web3.eth.defaultAccount, value: cost})
        .on('receipt', function(receipt) {
            console.log(receipt);
            window.location.href = 'success.html';
        })
        .on('error', console.error);
    });

```

Success.html (сторінка успішної оплати)

```

<!DOCTYPE html>
<html>
<head>
    <title>Оплата успішна</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
    <div class="container">
        <h1>Оплата успішна</h1>
        <p>Ви успішно оплатили вартість спожитої енергії.</p>
        <button id="home">Повернутись на головну</button>
    </div>

```

```
        <script src="success.js"></script>
    </body>
</html>
```

Success.js (скрипт переходу на головну сторінку)

```
document.getElementById('home').addEventListener('click', function() {
    window.location.href = 'index.html';
});
```

ДОДАТОК В

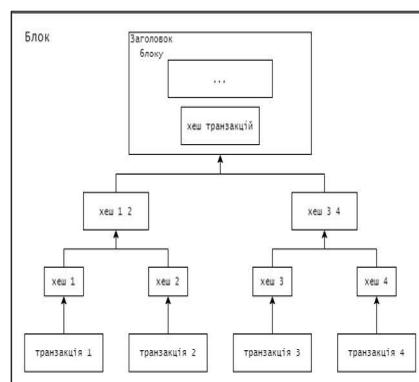
ПРЕЗЕНТАЦІЯ

Реалізація смарт контракту для оплати комунальних послуг

ПІДГОТУВАЛА:
СТУДЕНТКА ГРУПИ РТ-819
ЗАЙЦЕВА А.О.
КЕРІВНИК:
НЕЛАСА Г.В.

Рисунок В.1 – Титульний слайд

Технологія блокчейн



Блокчейн – це цифровий список записів даних, який постійно зростає. Такий список складається з багатьох блоків даних, кожен з яких містить часову позначку, хеш попереднього блоку та дані транзакцій, подані як хеш-дерево.

2/15

Рисунок В.2 – Слайд 2

Блокчейн-платформа Ethereum



Ethereum – це децентралізована розподілена обчислювальна платформа з відкритим кодом, яка дозволяє створювати смарт-контракти та децентралізовані програми, також відомі як DApps.

Ethereum – це, по суті, єдина децентралізована система, яка керує комп'ютером під назвою **віртуальна машина Ethereum (EVM)**. В інших випадках мережеві взаємодії вважаються «**транзакціями**» та зберігаються в блоках блокчейну Ethereum.

3/15

Рисунок В.3 – Слайд 3

Алгоритми безпеки блокчейн

Proof of Work

Перший вузол або учасник, який підтвердить додавання нових даних або транзакцію в цифровій книзі, отримує певну кількість токенів як винагороду. Щоб завершити процес перевірки, учасник або «майнер» повинен вирішити криптографічне запитання.

Proof of Stake

Майнери оцінюються на основі кількості нативних монет протоколу, які вони мають у своїх цифрових гаманцях, і тривалості часу, протягом якого вони їх мали. Майнер із найбільшою кількістю монет має більше шансів бути обраним для підтвердження транзакції та отримання винагороди.

4/15

Рисунок В.4 – Слайд 4

Смарт-контракт

Смарт-контракти – набір функцій і даних, який зберігається в блокчейні.

«Розумні контракти» це як **цифрові заяви «якщо-тоді»** між двома (чи більше) сторонами. Якщо потреби однієї групи задовольняються, угода може бути виконана, і контракт вважається завершеним.

Ідею смарт-контракту можна розбити на такі кроки:

1. Визначення сторін та встановлення умов угоди.
2. Визначення умов, які мають бути виконані для виконання контракту.
3. Написання коду смарт-контракту.
4. Розгортання смарт-контракту на платформі блокчейн.
5. Автоматичний запуск виконання контракту.
6. Інформація про контракт вводиться в мережу блокчейн під час його виконання.

5/15

Рисунок В.5 – Слайд 5

Мова програмування Solidity



Для реалізації смарт-контракту обираємо мову програмування Solidity.

- | | |
|--|--|
| <ul style="list-style-type: none"> ☐ відкритий вихідний код; ☐ код виконується на віртуальній машині Ethereum (EVM); | <ul style="list-style-type: none"> ☐ статична типізація; ☐ підтримка успадкування; ☐ підтримка складних типів, що визначаються користувачем. |
|--|--|

6/15

Рисунок В.6 – Слайд 6

UML-діаграма системи оплати



Варіант використання: Оплата за електроенергію

Актори: Споживач, Підприємство

Ціль: Проведення успішного платежу

Включає в себе інші варіанти використання:

- ☐ розрахунок платежу за показаннями;
- ☐ проведення платежу;
- ☐ оплата заборгованості (виняток).

7/15

Рисунок В.7 – Слайд 7

Реалізація смарт-контракту

```

4 contract EnergyPayment {
5
6   address private owner;
7   uint public rate; // вартість одиниці енергії
8
9   struct Consumption {
10    uint amount; // кількість спожитої енергії
11    uint cost; // вартість спожитої енергії
12    bool paid; // чи оплачено споживання
13  }
14
15  mapping(address => Consumption[]) public consumptions;
16
17  constructor(uint _rate) {
18    owner = msg.sender;
19    rate = _rate;
20  }
21
22  modifier onlyOwner {
23    require(msg.sender == owner, "caller is not owner");
24    _;
25  }
26
27  // Зміна вартості одиниці енергії
28  function changeRate(uint _rate) public onlyOwner {
29    rate = _rate;
30  }
31
32  // Запис спожитої енергії
33  function recordConsumption(uint _amount) public {
34    Consumption memory newConsumption;
35    newConsumption.amount = _amount;
  
```

Треба створити систему оплати послуг на основі смарт-контракту. Є дві сторони даної угоди: підприємство та споживач.

Підприємство в даному випадку виставляє вартість одиниці енергії, робить розрахунок за значеннями від споживача, отримує гроші.

Споживач повинен надсилати дані, про спожиту енергію, та оплачувати її.

Код будемо писати на платформі Remix IDE.

8/15

Рисунок В.8 – Слайд 8

Реалізація смарт-контракту

```
modifier onlyOwner {
    require(msg.sender==owner,
    "Caller is not owner");
    _;
}
```

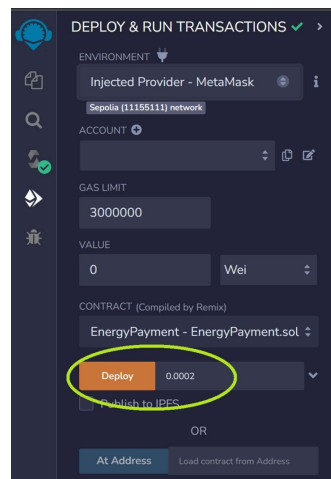
Щоб запобігти несанкціонованому використанню функцій смарт-контракту, впроваджуємо безпечний контроль доступу. Використаємо модифікатор OnlyOwner.

В модифікаторі виконуємо операцію, яка є перевіркою того, що саме власник контракту викликає функцію. Для цього ми використовуємо функцію require, яка перевіряє наявність умов та видає виняток, якщо умова не виконується.

9/15

Рисунок В.9 – Слайд 9

Реалізація смарт-контракту

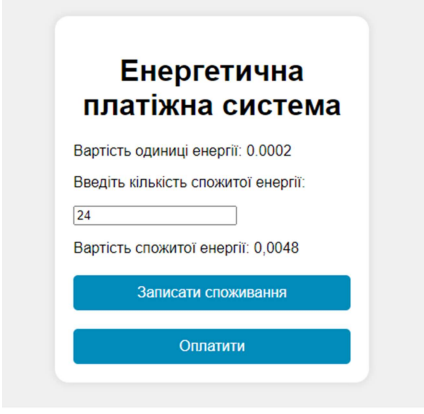


Для того, щоб реалізувати смарт-контракт, його треба розгорнути. В Remix IDE заходимо в Deploy & run transactions і натискаємо на кнопку Deploy. Також вказуємо вартість одиниці енергії.

10/15

Рисунок В.10 – Слайд 10

Реалізація смарт-контракту



Енергетична платіжна система

Вартість одиниці енергії: 0.0002

Введіть кількість спожитої енергії:

Вартість спожитої енергії: 0.0048

[Записати споживання](#)

[Оплатити](#)

Після розгортання контракту переходимо в систему оплати. Споживач одразу бачить вартість одиниці енергії, яку виставило підприємство. За потреби воно може змінити цю вартість.

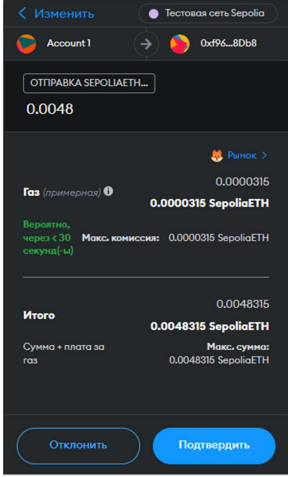
Споживачу треба ввести кількість енергії, яку він спожив. Після чого натиснути кнопку “Записати споживання”.

Далі споживач може побачити вартість спожитої енергії, після чого оплатити її, натиснувши кнопку “Оплатити”.

11/15

Рисунок В.11 – Слайд 11

Реалізація смарт-контракту



Измeнить Тестовая сеть Sepolia

Account 1 → 0x96...8Dy8

ОТПРАВКА SEPOLIAETH... 0.0048

Рынок >

Газ (примерная) 0.0000315

Всего 0.0000315 SepoliaETH

Вероятно, через ~30 секунд (ч) Макс. комиссия: 0.0000315 SepoliaETH

Итого 0.0048315

0.0048315 SepoliaETH

Сумма + плата за газ Макс. сумма: 0.0048315 SepoliaETH

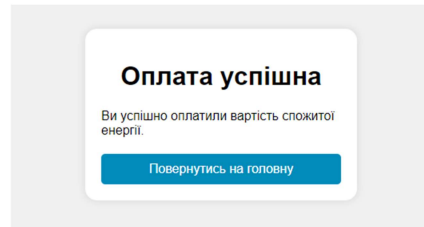
[Отклонить](#) [Подтвердить](#)

Оплата услуг проходит в крипто гаманці MetaMask. Споживач може побачити дані про транзакцію, а саме вартість спожитої енергії та комісію за дану транзакцію. Якщо дані правильні, то споживач підтверджує операцію та чекає, коли пройде обробка.

12/15

Рисунок В.12 – Слайд 12

Реалізація смарт-контракту



В результаті на сторінці системи оплати з'являється вікно про успішну оплату.

Завдяки даному смарт-контракту споживачі можуть побачити вартість за одиницю енергії, записати показання, а потім оплатити її. Власник контракту (підприємство) може змінювати вартість одиниці енергії і знімати накопичені кошти з контракту.

13/15

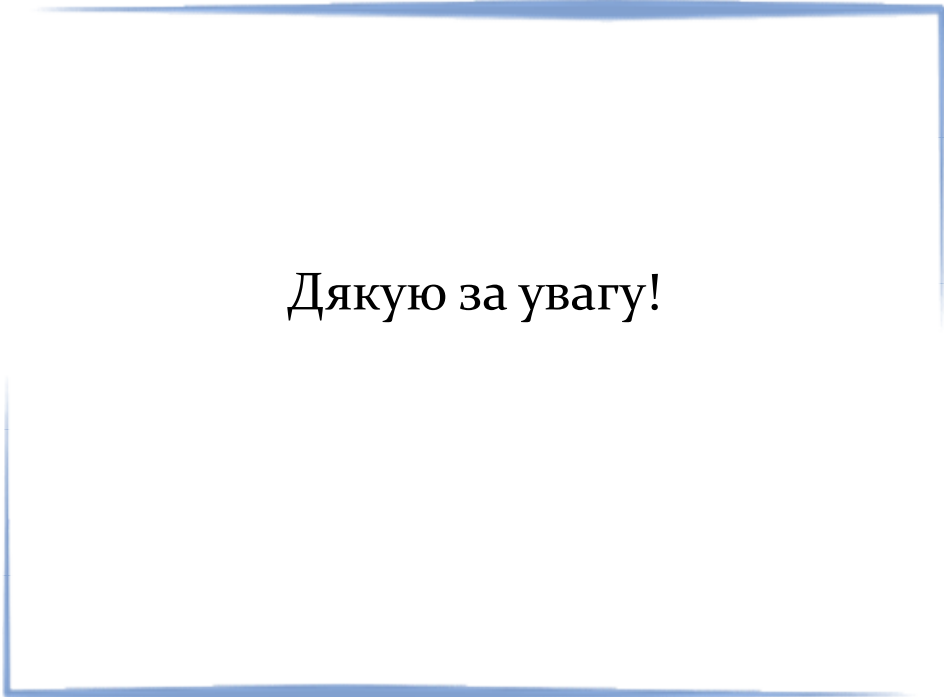
Рисунок В.13 – Слайд 13

Висновки

В ході роботи було розглянуто та досліджено поняття та ідею смарт-контрактів. Для написання контракту було обрано мову програмування Solidity та середовище розробки Remix IDE. Було розроблено систему оплати комунальних послуг на основі смарт-контракту. Також було розроблено інтерфейс, через який можна буде взаємодіяти із контрактом на основі мови HTML та CSS. Завдяки розумному контракту споживачі бачать вартість за одиницю енергії, записують свої показання, а потім сплачують. Підприємство за потреби змінює вартість одиниці енергії і знімає накопичені кошти з контракту.

14/15

Рисунок В.14 – Слайд 14



Дякую за увагу!

Рисунок В.15 – Фінальний слайд