УДК 336.7+004.42
DOI: https://doi.org/10.32782/1814-1161/2023-4-17

Хрипко С.Л.
доктор технічних наук, професор,
завідувач кафедри інформаційних технологій та дизайну
Класичного приватного університету

Щербаков С.С.
аспірант
Класичного приватного університету

Khrypko Serhii
Doctor of Technical Sciences, Professor,
Head of the Department of Information Technologies and Design
Classic Private University

Shcherbakov Serhii
Postgraduate Student
Classic Private University

ДОСЛІДЖЕННЯ ТЕХНОЛОГІЇ БЛОКЧЕЙНУ ДЛЯ МІКРОКРЕДИТУВАННЯ І ФІНАНСУВАННЯ В ОСВІТНІЙ СФЕРІ

RESEARCH ON BLOCKCHAIN TECHNOLOGY FOR MICROCREDIT AND FINANCING IN THE EDUCATIONAL SECTOR

У даній статті досліджується, як блокчейн перетворює фінансування освіти через мікрокредитування. Автори пропонують модель базового смарт-контракту на платформі Ethereum для надання мікрокредитів. Описано його структуру, яка включає: модифікатори, події, функції дозволу на випуск токенів та їх відправку, запиту та погодження та погашення кредиту, перевірку балансу. Також надається пояснення особливостей програмного коду на мові Solidity, яка включає важливі складові, як перевірка параметрів логіки функцій, шляхом розгортання в тестовій мережі. Вона також надає засоби для роботи зі змінними стану всередині написаних процедур, розрахунок витрат і можливість взаємодії з іншими Web3 гаманцями та транзакціями. Розгортання даного програмного коду відбувається на віртуальній машині Ethereum в IDE Remix. Це надає можливість розробникам експериментувати з моделлю і функціоналом в тестовій мережі без ризику втрати реальних коштів чи завдання шкоди. Вона має подібну архітектуру до основної мережі, але використовує тестові ETH. Підтримує стандартні протоколи і інтерфейси. Застосування даного застосунку для надання мікрокредитів в навчальних закладах може полегшити фінансування студентів, забезпечуючи більше прозорості та надійності в освітньому процесі. Важливо зауважити, що ця модель не враховує де які деталі реального проекту. Впровадження моделі як реального проекту вимагає тестування та безпеку, а також розуміння специфіки використання. Звертаємо увагу, що данні функції викликаються з Ethereum адреси, яка належить власнику контракту. Виклик `getDebt`, `approveLoan` або `mint` з іншої адреси, призведе до помилки. Крім того, будь-яка адреса може викликати `requestLoan` та `repayLoan` для запиту та повернення кредиту.

Ключові слова: *мікрокредитування, смарт-контракт, ethereum, фінансування, сценарії виплати, технологія блокчейн, освітня сфера.*

The article discusses how the current microcredit system has its limitations and risks. Classically structured financial institutions set high interest rates and require a large number of documents to obtain a loan. In addition, there is the problem of controlling the use of funds provided as part of a microloan. Blockchain technology can provide a transparent and automated system that will allow students and young researchers to receive microloans on affordable terms. The aim of our work is to study blockchain technology for microcredit in the educational sphere by writing and implementing the program code of the program "Smart contract for microcredit on the Ethereum platform". Methods. The study used data from various sources, including scientific literature, statistical data, and information from websites specializing in blockchain technologies and microcredit. A comparative analysis of these data was conducted to determine the potential advantages and disadvantages of using blockchain in microcredit. Results. The research of our program code has shown that the use of blockchain can have significant advantages for

improving the microcredit process in education. In particular, blockchain can provide a more transparent and secure lending process, as well as reduce the risk of fraud. Discussion. Based on the results obtained, the advantages of blockchain technology for microcredit in the education sector are discussed. A number of limitations and issues that should be addressed when considering the application of this technology are discussed. The conclusions show that while there are some challenges associated with using blockchain for microcredit in education, its advantages as an innovative product to address these challenges are clear. Additional research in this subject area may help to unlock the full potential of this approach.

Keywords: *microcredit, smart contract, ethereum, financing, payment scenarios, blockchain technology, education.*

Постановка проблеми. В сучасному світі освіта стала не лише правом, але й ключовим чинником для розвитку індивіда та суспільства в цілому. Однак доступ до освіти може бути обмеженим через фінансові бар'єри, які включають:

– *Вартість навчання*: висока вартість навчання у вищих навчальних закладах може стати значним фінансовим обмеженням для багатьох осіб, особливо для тих, хто мешкає в регіонах з низьким рівнем доходів.

– *Витрати на матеріали та підручники*: придбання підручників та методичних матеріалів, навчальних посібників, програмного забезпечення та інших матеріалів може вимагати додаткових витрат, що ускладнює доступ до освіти.

– *Житлові витрати*: Для тих, хто намагається отримати вищу освіту в інших містах чи країнах, витрати на проживання, харчування та інші повсякденні потреби можуть бути великим фінансовим тягарем.

– *Втрата доходу:* для працездатних осіб навчання може вимагати відмови від постійної оплачуваної роботи або зменшення робочого графіку, що може спричинити втрату доходу та ускладнити фінансове забезпечення.

– *Кредити та борги:* Багато студентів беруть кредити для оплати навчання, що може призвести до накопичення великого боргу, який слід погашати після завершення освіти.

– *Брак фінансування та стипендій:* Не всі студенти мають доступ до достатнього фінансування або можливості отримати стипендії, особливо ті, хто має обмежений фінансовий стан.

– *Економічна нестабільність:* Негативні зміни у фінансовому стані сім'ї або в країні можуть суттєво ускладнити можливість отримання освіти.

Ці фінансові бар'єри створюють нерівність у доступі до освіти та можуть обмежувати розвиток навичок та потенціалу багатьох осіб, особливо тих, хто має обмежені фінансові можливості.

Розв'язання цих проблем може включати в себе політичні реформи, надання фінансової допомоги, зменшення вартості навчання та підвищення свідомості громадськості щодо важливості доступності освіти для всіх.

Аналіз останніх досліджень і публікацій. Grech Alex та Camilleri Anthony [1] представляють фундаментальні принципи блокчейну, зосереджуючись на його потенціалі для освітнього сектору. Вони пояснюють, як ця технологія може одночасно порушити інституційні норми та розширити можливості учнів. Він пропонує вісім сценаріїв застосування блокчейну в освітньому контексті на основі поточного стану розвитку та розгортання технологій.

Timothy Arndt [2] надає огляд недавніх досліджень застосування технології блокчейну в освіті. Він розглядає, блокчейн, як відкриту та розподілену книгу, що може ефективно записувати транзакції між двома сторонами у незмінному форматі без потреби в третій стороні.

Li Min та Ge Bin [3] досліджує використання технології блокчейну в онлайн-навчанні в університетах та розглядає, як блокчейн може покращити якість навчання та підвищити довіру до онлайн-освіти.

T.M. Fernandez-Carames, P. Fraga-Lamas [4] розглядає застосування ключових технологій для розробки розумних кампусів та університетів. Вони зазначають основні характеристики розумного університету, деталізують останні технології зв’язку та аналізують найбільш актуальні питання розгортання розумного кампусу на базі блокчейн. Крім того, в статті досліджується використання блокчейна в різних процесах вищої освіти.

F. Altinay, O. Beyatli, G. Dagli, Z. Altinay [5] досліджують використання моделі Edmodo для професійного розвитку та застосування технології блокчейну в управлінні школами. Розглядають, як школи почали робити тенденції до цифровізації, особливо в навчанні. В дослідженні було опитано 21 вчитель, які працюють в закладах середньої освіти та виявлено їх ставлення до цього процесу.

A. Cheriguene, T. Tai Kabache, A. Adnane, C.A. Ker rache, F. Farhan Ahmad [6], досліджували технічні переваги блокчейна та пропонують безпечний та надійний онлайн-фреймворк для навчання на основі блокчейн. Вони пропонують використовувати технологію блокчейн для забезпечення очікуваних стандартів навчання та справедливості оцінювання дотримуючись графіка курсів та іспитів. За допомогою методів винагороди блокчейн, має можливість мотивувати студентів, та вчителів продовжувати свої зусилля, навіть працюючи з дому.

M. Khan, T. Naz, [7] досліджували використання технології блокчейн в системах управління навчанням (LMS). Розглядають, як блокчейн може допомогти подолати ризики безпеки та вразливості, пов’язані з приватною інформацією, що зберігається в LMS. Автори пропонують безпечну децентралізовану LMS на основі приватної мережі блокчейн, яку назвали системою управління навчанням на основі блокчейн (BLMS).

В дослідженнях A.Y. Al-Zoubi, M. Dmour, R. Aldmour [8] показано використання технології

блокчейн в системах управління навчанням. Розглянуто шляхи покращення ефективності та безпеки управління навчальними лабораторіями за допомогою технології блокчейн.

Рядом дослідників, зокрема – S. Purnama, Q. Aini, U. Rahardja, N.P.L. Santoso, S. Millah [9] пропонують використовувати технологію блокчейн в процесах управління навчанням в різних хмарних середовищах.

W. Räther, S. Kolvenbach, R. Ruland, J. Schutte, C. Torres, F. Wendland [10], досліджували використання технології блокчейну в освіті. Вони розглядають, як може допомогти технологія блокчейн довгостроково зберігати освітні записи в доступних та захищених від змін реєстрах. Автори представляють платформу "Blockchain for Education" як практичне рішення для видачі, моніторингу, перевірки та обміну сертифікатів.

Однак, у вище згаданих статтях не наведено конкретних прикладів застосування блокчейн в освітньому секторі, а також немає обговорення можливих проблем або викликів, пов'язаних з його впровадженням. Критичний аналіз цих аспектів міг би бути корисним для користувачів. Крім того, вони не аналізують специфічних для країни аспектів застосування блокчейн в освітньому секторі.

Мета статті. Щоб зрозуміти блокчейн-технологію необхідно вивчити основи блокчейн-технології, які включаючи роботу з розподіленим реєстром та консенсус-алгоритмами, мережею блокчейн та використанням концепції безпеки.

Вибір платформи, на якій ми будемо розробляти смарт-контракт включають Ethereum, Binance Smart Chain, Cardano та інші.

Вивчення мови програмування смарт-контрактів. Для Ethereum, наприклад, ми повинні вивчити Solidity – мову програмування для смарт-контрактів. Для інших платформ існують відповідні мови програмування, такі як Plutus для Cardano.

Проведення дослідження проектів. Вивчення проектів, які використовують смарт-контракти на обраній платформі. Розглянути як приклад успішних, так і неуспішних проектів.

Розробка власних смарт-контрактів. Розробити власні смарт-контракти для вивчення роботи з платформою та мовою програмування.

Тестування та безпека. Оцінити методи тестування смарт-контрактів та розглянути аспекти безпеки, такі як захист від атак і аудит смарт-контрактів.

Аналіз витрат та продуктивності. Дослідити витрати на виконання смарт-контрактів і методи оптимізації для підвищення продуктивності.

Виклад основних результатів дослідження. Мікрокредитування на основі блокчейну – це інноваційний підхід до надання малих кредитів, який використовує технологію блокчейну для полегшення та удосконалення процесу надання та управління мікрокредитами. Ось деякі основні переваги цього підходу:

– Прозорість і довіра. Усі транзакції та угоди зберігаються в розподіленому реєстрі блокчейну, що робить їх доступними для перевірки та аудиту всіма учасниками. Це збільшує довіру до процесу мікрокредитування.

– Ефективність та низькі витрати. Блокчейн дозволяє автоматизувати багато аспектів процесу мікрокредитування за допомогою смарт-контрактів. Це призводить до скорочення витрат на посередників та бюрократичні процедури.

– Глобальний доступ. Блокчейн не має географічних обмежень, що означає, що інвестори можуть надавати кредити людям з будь-якого куточка світу. Це розширює можливості для людей отримати доступ до фінансової підтримки.

– Швидкість та доступність. Блокчейн дозволяє проводити транзакції миттєво, що особливо важливо в сфері мікрокредитів, де швидкий доступ до фінансування може врятувати ситуацію.

– Контроль за даними. Кожен користувач має контроль над своїми особистими даними, і доступ до них може бути наданий лише з дозволу власника даних. Це підвищує конфіденційність та безпеку.

Мікрокредитування на основі блокчейну може значно полегшити доступ до фінансування для малих підприємців, студентів та інших потенційних користувачів, покращуючи ефективність та прозорість цього процесу. Для цього необхідно розробити смарт-контракт та розгорнути його.

Смарт-контракти і блокчейн – це дві ключові складові, які працюють разом для створення безпечних, децентралізованих та автономних систем. Ось як вони поєднуються.

Блокчейн – це розподілена база даних, яка зберігає та публікує дані в режимі реального часу на тисячах вузлів (комп'ютерів) у всьому світі. Ця децентралізованість робить блокчейн незалежним від централізованих організацій і одержується завдяки глибокому використанню смарт-контрактів.

Смарт-контракти містять умови та правила для виконання автоматизованих дій на основі певних подій або умов. Вони дозволяють автоматизувати процеси і забезпечують виконання угод без посередництва третьої сторони. Виконуються на всіх вузлах блокчейну, що робить їх вкрай надійними та стійкими до атак. Крім того, вони безпечні завдяки внутрішнім механізмам безпеки та вбудованій автентифікації.

Смарт-контракти дозволяють автоматизувати виконання угод і правил, зменшуючи ризик помилок та швидше виконуючи дії. Вони виконують дії лише відповідно до попередньо визначених умов, що додає надійності та прозорості в угоди.

Блокчейн – це база даних транзакцій, яка оновлюється та доступна з багатьох комп'ютерів у мережі. Щоразу, коли додається група транзакцій, з'являється "блок" – звідси і назва "блокчейн" (тобто "ланцюжок блоків"). Це публічна база даних, і інформація, яку зберігають смарт-контракти, доступна для перевірки всіма учасниками мережі. Це створює високий рівень прозорості та довіри.

Завдяки смарт-контрактам можливе безпосереднє укладення угод між сторонами, що виключає потребу в посередниках, таких як банки чи юридичні консультанти. Це може значно зменшити витрати та прискорити процеси.

Спільна робота смарт-контрактів та блокчейну відкриває безліч можливостей для створення нових додатків і систем, в яких автоматизовані умови і правила грають ключову роль в децентралізованому світі. Це може бути застосовано в багатьох галузях, включаючи фінанси, логістику, управління ресурсами, медицину, голосування та багато інших.

Ефективне рішення для проблеми мікрокредитування можна надати за технологією блокчейн. Розглянемо послідовність дій технології.

1. Спершу, користувачі реєструються на платформі мікрокредитування на базі блокчейну та проходять ідентифікацію. Їх особисті дані зберігаються в блокчейні, що забезпечує високий рівень безпеки.

2. Користувачі можуть подавати запити на мікрокредити через платформу. Ці запити записуються у блокчейні.

3. Спеціалізовані алгоритми аналізують дані, які зберігаються в блокчейні, для визначення кредитоспроможності користувачів. Це може включати інформацію про фінансову історію, роботу та інші фактори.

Смарт-контракти на видачу кредиту: Якщо користувач має прийнятну кредитоспроможність, то через смарт-контракти на блокчейні може бути автоматично видача кредиту. Умови кредиту, такі як відсоткова ставка та строк, також фіксуються в смарт-контракті.

4. Користувачі можуть вносити платежі за кредит через блокчейн, і ці транзакції також будуть зафіксовані у розподіленому реєстрі.

5. Прозорість блокчейну дозволяє стежити за всіма операціями та забезпечує можливість проведення аудиту.

Створення повноцінного застосунку для надання малих кредитів на блокчейні – це досить складна задача і вимагає розробки великого проекту.

На даний момент існує безліч платформ для розробки смарт-контрактів. Ось деякі з них:

– Binance Smart Chain – це блокчейн, який працює паралельно з Binance Chain і підтримує смарт-контракти. Він дозволяє розробникам створювати децентралізовані додатки та токени [12].

– Polkadot – це мультиблокчейн-платформа, яка дозволяє інтероперабельність між різними блокчейнами. Розробники можуть створювати смарт-контракти на різних підблокчейнах Polkadot [13].

– Cardano – це блокчейн, який використовує мову програмування Haskell і має вбудовану підтримку смарт-контрактів. Він розвивається з фокусом на безпеку і скалькомпосабельність [14].

– Ethereum – це мережа комп'ютерів по всьому світу, які дотримуються набору правил, які називаються протоколом Ethereum. Мережа Ethereum є основою для спільнот, програм, організацій та цифрових активів, які може створити та використовувати кожен [15].

– Solana – це високошвидкісний блокчейн, який підтримує смарт-контракти. Він відомий своєю високою продуктивністю і низькими комісіями [16].

– Tezos – це самозмінюваний блокчейн, який використовує мову програмування Michelson для смарт-контрактів. Він має механізм управління і оновленням протоколу через голосування спільноти [17].

– Avalanche – це блокчейн-платформа, яка підтримує різні віртуальні машини, включаючи EVM (Ethereum Virtual Machine), що дозволяє розробникам використовувати смарт-контракти, схожі на Ethereum[16].

– EOS – це платформа для створення децентралізованих додатків, яка підтримує смарт-контракти. Вона відома своєю високою швидкістю та масштабованістю [15].

– Tron – це блокчейн, який також підтримує смарт-контракти і фокусується на розвитку додатків та контенту [14].

Кожна з цих платформ має свої особливості та переваги, і вибір платформи для розробки смарт-контрактів залежить від конкретних вимог вашого проекту та вашого досвіду.

Природним вибором для розробки смарт-контрактів на платформі Ethereum є мова програмування Solidity [18]. У загальнодоступні блокчейн, такі як Ethereum, будь-хто може додавати дані, але їх не можна видаляти. Якщо хтось хоче змінити інформацію або обдурити систему, це доведеться зробити на більшості комп'ютерів у мережі. Такий підхід робить децентралізовані блокчейну в Ethereum, дуже надійними.

Для створення моделі нами було використано мову Solidity. Для компіляції та розгортання смарт контракту середовище Remix IDE. Для оплати транзакцій – гаманець MetaMask з тестовою чи головною мережею обраної криптовалюти.

Remix IDE – це потужне та популярне інтегроване середовище розробки (IDE) для розробки розумних контрактів Solidity1. Веб-засноване IDE, яке дозволяє розробникам писати, тестувати, налагоджувати та розгортати розумні контракти на блокчейні Ethereum [19]. Remix IDE надає зручний інтерфейс, який спрощує процес розробки Solidity, роблячи його доступним для розробників всіх рівнів майстерності.

Однією з ключових переваг Remix є те, що він пропонує готове до використання рішення для розробки децентралізованих додатків. Це означає, що розробникам не потрібно встановлювати жодні додаткові плагіни або модулі, щоб почати кодування. Всі необхідні інструменти та функції вже вбудовані в платформу, що полегшує роботу та зручність для розробників.

Відкривши браузер і отримавши доступ до Remix IDE, розробники можуть негайно почати кодування та тестування своїх розумних контрактів. Це економить час на процедурах пов’язаних з встановленням та налаштуванням, а також дозволяє розробникам зосередитися на цілях проекту та написані програмного коду.

Комбінація Solidity та Remix IDE – використовується для індустрії блокчейна. За допомогою Solidity розробники можуть створювати безпечні та децентралізовані застосунки, які працюють на

блокчейн Ethereum. A Remix IDE розробники можуть швидко писати, тестувати, налагоджувати та розгортати свої розумні контракти, прискорюючи процес розробки та зменшуючи ризик помилок.

Solidity – це об'єктно-орієнтована, високо рівнєва мова програмування, розроблена для створення розумних контрактів, які працюють на Ethereum [20].

Solidity є статично типізованою мовою, яка підтримує наслідування, бібліотеки та складні типи, визначені користувачем та має інші функції. За допомогою Solidity можливо створювати старт-контракти для таких застосувань, як голосування, краудфандинг, сліпі аукціони та багатопідписні гаманці.

При розгортанні контрактів слід використовувати останню випущену версію Solidity. Крім виняткових випадків, лише остання версія отримує виправлення безпеки. Більше того, зміни, що порушують сумісність, а також нові функції регулярно вводяться. Ми використовували версію 0.8.18 для позначення цього швидкого темпу змін.

Для встановлення та налаштування MetaMask для роботи з тестовою мережею, необхідно виконати наступні кроки:

1. Встановити розширення MetaMask для вашого браузера. Для цього завантажити на офіційному сайті MetaMask [21] або в магазині розширень вашого браузера. В нашому випадку браузер Google Chrome.

2. Після встановлення MetaMask необхідно відкрити у вашому браузері.

3. Створити новий гаманець або імпортувати існуючий.

4. Вибравши "Settings" перейдіть до налаштувань MetaMask.

5. У меню налаштувань виберіть "Networks" (Мережі), а потім натисніть "Add Network" (Додати мережу). Ми використовували WhiteBIT Network.

6. У форму, що з'явилася, введіть необхідну інформацію про тестову та основну мережу. Ця інформація зазвичай включає назву мережі, URL RPC, ID ланцюга, символ та URL провідника блоку.

Основна мережа WhiteBIT Network:
Ім'я мережі: WB Network
Новий URL RPC: https://rpc.whitebit.network
Ідентифікатор ланцюжка: 1875
Символ валюти: WBT
Адреса провідника блоку: https://explorer.whitebit.network

Тестова мережа WhiteBIT Network Testnet:
Ім'я мережі: WB Network Testnet
Новий URL RPC: https://rpc-testnet.whitebit.network
Ідентифікатор ланцюжка: 2625
Символ валюти: WBT
Адреса провідника блоку: https://explorer.whitebit.network/testnet

7. Натисніть "Save" (Зберегти), щоб додати мережу.

8. Тепер є можливість переключитися між основною мережею Ethereum та новою тестовою мережею, використовуючи спадаюче меню мереж у верхньому середньому положенні головного інтерфейсу MetaMask.

9. Поповнити рахунок тестовими чи справжніми токенами.

Було створено Single смарт-контракт на Solidity для створення ERC20 токенів. Загальна кількість токенів 1000, назва MicroloanContract, будь який крипто гаманець має мати можливість змінити токени, але не більше xCount=100. Після того як всі токени будуть змінені, необхідно закрити можливість створення нових токенів. Не будемо використовувати бібліотеку OpenZeppling. Зробимо функцію SendToken на вхід якої подається масив із адрес і кількості токенів і ця функція має відправити токени по цим адресам. Ліцензія – Mit.

Версія ПЗ – pragma solidity ^0.8.18;

Для використання цього контракту було виконано наступні кроки:

1. Включення мінтингу: Власник контракту може дозволити іншим адресам мінтити токени, викликавши функцію `enableMinting(address wallet)`. Ця функція приймає адресу гаманця, якому дозволено мінтити токени. Наприклад:

```solidity
enableMinting(0x123456789abcdef);
```

Це означає, що власник дозволяє адресі `0x123456789abcdef` мінтити токени.

2. Мінтинг токенів: Адреса, якій було дозволено мінтити токени, може викликати функцію `mint(address recipient, uint256 amount)` для створення нових токенів і надсилання їх одержувачу. Наприклад:

```solidity
mint(0xabcdef123456789, 100);
```

Це означає, що ви створюєте 100 нових токенів і надсилаєте їх адресі `0xabcdef123456789`.

3. Вимкнення мінтингу: Власник контракту може заборонити іншим адресам мінтити токени, викликавши функцію `disableMinting(address wallet)`. Ця функція приймає адресу гаманця, якому заборонено мінтити токени. Наприклад:

```solidity
disableMinting(0x123456789abcdef);
```

Це означає, що власник забороняє адресі `0x123456789abcdef` змінити токени.

4. Запит на кредит: Ви можете подати запит на кредит, викликавши функцію `requestLoan(uint256 _loanAmount, uint256 _interestRate, uint256 _loanTerm)`. Ця функція приймає суму кредиту, процентну ставку і строк кредиту. Наприклад:

```solidity
requestLoan(100, 5, 30);
```

Це означає, що ви подаєте запит на кредит у розмірі 100 токенів з процентною ставкою 5% і строком кредиту 30 днів.

5. Погодження кредиту: Після того, як ви подали запит на кредит, власник контракту може

погодити ваш запит. Власник викликає функцію `approveLoan(address _borrower)`, де `_borrower` – це ваша адреса. Наприклад:

```solidity
approveLoan(0x123456789abcdef);
```

Це означає, що власник погоджує ваш запит на кредит.

6. Повернення кредиту: Після того, як ваш запит на кредит було погоджено, ви можете повернути кредит. Ви можете викликати функцію `repayLoan(uint256 _repaidAmount)`, де `_repaidAmount` – це сума, яку ви повертаєте. Наприклад:

```solidity
repayLoan(105);
```

Це означає, що ви повертаєте 105 токенів (100 токенів основного боргу плюс 5% відсотків).

7. Перевірка боргу: Ви можете перевірити залишок вашого боргу за допомогою функції `getDebt(address _borrower)`. Ця функція приймає адресу позичальника і повертає суму заборгованості з урахуванням процентної ставки і суми, яка вже сплачена. Наприклад:

```solidity
getDebt(0x123456789abcdef);
```

Це означає, що ви перевіряєте залишок вашого боргу.

Функція sendTokens не використовується. Ця функція була включена в контракт для інших можливих сценаріїв використання, де може знадобитися відправка токенів до декількох отримувачів одночасно без кредитної позики.

8. Критерії та обмеження оцінки якості результатів:

– Забезпечення доступності мікрокредитів для позичальників з обмеженою кредитною історією та без бюрократичних перешкод.

– Зниження відсоткових ставок на позики, щоб полегшити їх погашення для позичальників.

– Забезпечення прозорості умов кредитування та обробки погашень.

– Мінімізація комісій та посередників у процесі надання та погашення позик.

9. Даний код представляє собою контракт Ethereum, написаний на мові програмування Solidity. Ось опис кожного рядка:

1. `contract MicroloanContract {`: Це початок оголошення контракту з назвою "MicroloanContract".

2. `string public name = "MicroloanContract";`: Це встановлює публічну змінну `name` контракту як "MicroloanContract".

3. `string public symbol = "MLC";`: Це встановлює публічний символ контракту як "MLC".

4. `uint8 public decimals = 18;`: Це встановлює кількість десяткових знаків для токенів у контракті як 18.

5. `uint256 public totalSupply = 1000 *10** int256(decimals);`: Це встановлює загальну кількість токенів, що постачаються, як 1000, помножене на 10 в степені кількості десяткових знаків.

6. `uint256 public xCount = 100;`: Це встановлює публічну змінну `xCount` як 100.

7. `uint256 public totalMinted = 0;` : Це встановлює загальну кількість виданих токенів як 0.

8. `address public owner;`: Це оголошує публічну змінну `owner`, яка буде містити адресу власника контракту.

9. `mapping(address => uint256) public balances;`: Це створює публічне відображення `balances`, яке зберігає баланси токенів для кожної адреси.

10. `mapping(address => bool) public canMint;`: Це створює публічне відображення `canMint`, яке визначає, чи може даний адресат видавати токени.

11. `struct Loan {...}`: Це оголошення структури "Loan", яка має поля для суми позики, процентної ставки, строку позики, статусу позики, статусу погашення та сплаченої суми. Структура Loan визначає об'єкт позики з наступними полями:

uint256 loanAmount;: Це поле вказує суму позики. uint256 – це беззнакове ціле число з 256 бітами, що може представляти дуже великі числа.

uint256 interestRate;: Це поле вказує процентну ставку позики. Знову, uint256 використовується для представлення цього числа.

uint256 loanTerm;: Це поле вказує строк позики. Це кількість часу, протягом якого позика повинна бути повернута.

uint8 loanStatus;: Це поле вказує статус позики. uint8 – це без знакове ціле число з 8 бітами, що може представляти числа від 0 до 255. Різні числа можуть представляти різні статуси позики.

uint8 repaymentStatus;: Це поле вказує статус погашення позики. Так само, як і loanStatus, різні числа можуть представляти різні статуси погашення.

uint256 repaidAmount;: Це поле вказує суму, яка була вже сплачена.

12. `mapping(address => Loan) public loans;`: Це створює публічне відображення `loans`, яке зберігає інформацію про позику для кожної адреси.

13 event Transfer(address indexed from, address indexed to, uint256 value);: Це оголошення події Transfer, яка використовується для відстеження передачі токенів від одного адресата до іншого. Вона приймає адресу відправника, адресу отримувача та значення передачі як параметри.

14 event MintEnabled(address indexed wallet);: Це оголошення події MintEnabled, яка використовується для відстеження, коли дозволено видавати токени для певного гаманця. Вона приймає адресу гаманця як параметр.

15 event MintDisabled(address indexed wallet);: Це оголошення події MintDisabled, яка використовується для відстеження, коли заборонено видавати токени для певного гаманця. Вона також приймає адресу гаманця як параметр.

16 event LoanRequested(address indexed borrower, uint256 loanAmount, uint256 interestRate, uint256 loanTerm);: Це оголошення події LoanRequested, яка використовується для відстеження, коли користувач запитує позику. Вона приймає адресу позичальника, суму позики, процентну ставку та строк позики як параметри.

17 event LoanApproved(address indexed borrower, uint256 loanAmount);: Це оголошення події LoanApproved, яка використовується для відстеження, коли позика схвалена. Вона приймає адресу позичальника та суму позики як параметри.

18 event LoanRepaid(address indexed borrower, uint256 repaidAmount, uint8 repaymentStatus);: Це оголошення події LoanRepaid, яка використовується для відстеження, коли позика повернута. Вона приймає адресу позичальника, сплачену суму та статус погашення як параметри.

```
19  constructor() {
        owner = msg.sender;
        balances[msg.sender] = totalSupply;
        canMint[msg.sender] = true;
    }
```

Це конструктор контракту, який викликається один раз при розгортанні контракту. Він ініціалізує приватні змінні.

owner = msg.sender;: Цей рядок встановлює власника контракту. msg.sender – це адреса того, хто створив контракт.

balances[msg.sender] = totalSupply;: Цей рядок встановлює баланс власника контракту рівним загальному постачанню токенів. Це означає, що всі токени належать власнику при створенні контракту.

canMint[msg.sender] = true;: Цей рядок дозволяє власнику контракту видавати нові токени. Значення true означає, що власник може видавати токени.

20 Далі реалізуємо два модифікатори. Вони використовуються для зміни поведінки функцій за допомогою додавання попередніх умов до виконання функції.

```
modifier onlyOwner() {
require(msg.sender == owner, "Only owner can call this function");
        _;
    }
```

modifier onlyOwner() {: Це початок оголошення модифікатора onlyOwner. Цей модифікатор забезпечує, що тільки власник контракту може викликати певну функцію.

require(msg.sender == owner, "Only owner can call this function");: Цей рядок перевіряє, чи є викликаючий (той, хто надсилає повідомлення) власником контракту. Якщо це не так, виклик функції припиняється, і повертається повідомлення про помилку “Only owner can call this function”.

_;: Цей символ означає, що після перевірки умови модифікатора буде виконано код функції.

}: Кінець оголошення модифікатора.

```
21  modifier canMintTokens() {
        require(totalMinted < totalSupply, "All tokens have been minted");
        require(canMint[msg.sender], "You are not allowed to mint tokens");
        _;
    }
```

modifier canMintTokens() {: Це початок оголошення модифікатора canMintTokens. Цей модифікатор перевіряє, чи можна видавати токени.

require(totalMinted < totalSupply, "All tokens have been minted");: Цей рядок перевіряє, чи всі токени були видані. Якщо це так, виклик функції припиняється, і повертається повідомлення про помилку “All tokens have been minted”.

require(canMint[msg.sender], "You are not allowed to mint tokens");: Цей рядок перевіряє, чи дозволено викликаючому видавати токени. Якщо це не так, виклик функції припиняється, і повертається повідомлення про помилку

“You are not allowed to mint tokens”.

_;: Так само як і в модифікаторі onlyOwner, цей символ означає, що після перевірки умови модифікатора буде виконано код функції.

}: Кінець оголошення модифікатора.

```
22  function enableMinting(address wallet) public onlyOwner {
        canMint[wallet] = true;
        emit MintEnabled(wallet);
    }
```

Цей блок коду є функцією enableMinting, яка дозволяє власнику контракту дозволити видавання токенів для певного гаманця. Ось що робить кожен рядок:

function enableMinting(address wallet) public onlyOwner {: Це початок оголошення функції enableMinting. Ця функція приймає адресу гаманця як параметр. Вона є публічною, тому може бути викликана будь-ким, але модифікатор onlyOwner обмежує її виклик тільки власником контракту.

canMint[wallet] = true;: Цей рядок встановлює значення true для адреси гаманця в мапі canMint. Це означає, що гаманець тепер може видавати токени.

emit MintEnabled(wallet);: Цей рядок викликає подію MintEnabled і передає адресу гаманця як параметр. Це повідомляє про те, що гаманець тепер може видавати токени.

}: Кінець оголошення функції.

```
23  function disableMinting(address wallet) public onlyOwner {
        canMint[wallet] = false;
        emit MintDisabled(wallet);
    }
```

Цей блок коду є функцією disableMinting, яка дозволяє власнику контракту заборонити видавання токенів для певного гаманця. Ось що робить кожен рядок:

function disableMinting(address wallet) public onlyOwner {: Це початок оголошення функції disableMinting. Ця функція приймає адресу гаманця як параметр. Вона є публічною, тому може бути викликана будь-ким, але модифікатор onlyOwner обмежує її виклик тільки власником контракту.

canMint[wallet] = false;: Цей рядок встановлює значення false для адреси гаманця в мапі canMint. Це означає, що гаманець більше не може видавати токени.

emit MintDisabled(wallet);: Цей рядок викликає подію MintDisabled і передає адресу гаманця як параметр. Це повідомляє про те, що гаманець більше не може видавати токени.

}: Кінець оголошення функції.

24 function mint(address recipient, uint256 amount) public canMintTokens onlyOwner {
require(recipient != address(0), "Invalid address");
require(totalMinted + amount <= totalSupply, "Minting would exceed the total supply");

balances[recipient] += amount;
totalMinted += amount;
emit Transfer(address(0), recipient, amount);

if (totalMinted >= totalSupply) {
totalMinted = totalSupply;
canMint[msg.sender] = false;
}
}

Цей блок коду є функцією mint, яка дозволяє власнику контракту видавати токени для певного отримувача. Ось що робить кожен рядок:

function mint(address recipient, uint256 amount) public canMintTokens onlyOwner {: Це початок оголошення функції mint. Ця функція приймає адресу отримувача та суму як параметри. Вона є публічною, тому може бути викликана будь-ким, але модифікатори canMintTokens та onlyOwner обмежують її виклик тільки власником контракту, який може видавати токени.

require(recipient != address(0), "Invalid address");: Цей рядок перевіряє, чи є адреса отримувача дійсною. Якщо це не так, виклик функції припиняється, і повертається повідомлення про помилку “Invalid address”.

require(totalMinted + amount <= totalSupply, "Minting would exceed the total supply");: Цей рядок перевіряє, чи не перевищить видавання токенів загальне постачання. Якщо це так, виклик функції припиняється, і повертається повідомлення про помилку “Minting would exceed the total supply”.

balances[recipient] += amount;: Цей рядок додає суму до балансу отримувача.

totalMinted += amount;: Цей рядок додає суму до загальної кількості виданих токенів.

emit Transfer(address(0), recipient, amount);: Цей рядок викликає подію Transfer і передає адресу 0 (що означає, що токени були видані, а не передані від іншого адресата), адресу отримувача та суму як параметри.

if (totalMinted >= totalSupply) {...}: Цей рядок перевіряє, чи було видано всі токени.

totalMinted = totalSupply;: Якщо всі токени були видані, цей рядок встановлює totalMinted рівним totalSupply.

canMint[msg.sender] = false;: Якщо всі токени були видані, цей рядок забороняє власнику контракту видавати більше токенів.

}: Кінець умовного блоку.

}: Кінець оголошення функції.

25 function sendTokens(address[] memory recipients, uint256[] memory amounts) public canMintTokens {
require(recipients.length == amounts.length, "Invalid input");
for (uint256 i = 0; i < recipients.length; i++)
{
address to = recipients[i];
uint256 amount = amounts[i];

require(to != address(0), "Invalid address");
require(balances[msg.sender] >= amount, "Insufficient balance");
require(balances[to] + amount >= balances[to], "Overflow");

balances[msg.sender] -= amount;
balances[to] += amount;
emit Transfer(msg.sender, to, amount);

totalMinted += amount;
if (totalMinted >= totalSupply)
{
totalMinted = totalSupply;
canMint[msg.sender] = false;
}
}
}

Цей блок коду є функцією sendTokens, яка дозволяє власнику контракту відправити токени до декількох отримувачів одночасно. Ось що робить кожен рядок:

function sendTokens(address[] memory recipients, uint256[] memory amounts) public canMintTokens {: Це початок оголошення функції sendTokens. Ця функція приймає масив адрес отримувачів та масив сум як параметри. Вона є публічною, тому може бути викликана будь-ким, але модифікатор canMintTokens обмежує її виклик тільки власником контракту, який може видавати токени.

require(recipients.length == amounts.length, "Invalid input");: Цей рядок перевіряє, чи мають масиви recipients та amounts однакову довжину. Якщо це не так, виклик функції припиняється, і повертається повідомлення про помилку “Invalid input”.

for (uint256 i = 0; i < recipients.length; i++) {...}: Цей рядок починає цикл for, який проходиться по кожному елементу масивів recipients та amounts.

address to = recipients[i];: Цей рядок зберігає адресу поточного отримувача в змінну to.

uint256 amount = amounts[i];: Цей рядок зберігає суму поточного переказу в змінну amount.

require(to != address(0), "Invalid address");: Цей рядок перевіряє, чи є адреса отримувача дійсною. Якщо це не так, виклик функції припиняється, і повертається повідомлення про помилку “Invalid address”.

require(balances[msg.sender] >= amount, "Insufficient balance");: Цей рядок перевіряє, чи достатньо балансу у власника контракту для переказу. Якщо це не так, виклик функції припиняється, і повертається повідомлення про помилку “Insufficient balance”.

require(balances[to] + amount >= balances[to], "Overflow");: Цей рядок перевіряє, чи не буде переповнення при додаванні суми до балансу отри-

мувача. Якщо це так, виклик функції припиняється, і повертається повідомлення про помилку “Overflow”.

balances[msg.sender] -= amount;: Цей рядок вираховує суму з балансу власника контракту.

balances[to] += amount;: Цей рядок додає суму до балансу отримувача.

emit Transfer(msg.sender, to, amount);: Цей рядок викликає подію Transfer і передає адресу власника контракту, адресу отримувача та суму як параметри.

totalMinted += amount;: Цей рядок додає суму до загальної кількості виданих токенів.

if (totalMinted >= totalSupply) {...}: Цей рядок перевіряє, чи було видано всі токени.

totalMinted = totalSupply;: Якщо всі токени були видані, цей рядок встановлює totalMinted рівним totalSupply.

canMint[msg.sender] = false;: Якщо всі токени були видані, цей рядок забороняє власнику контракту видавати більше токенів.

}: Кінець умовного блоку.

}: Кінець циклу for.

}: Кінець оголошення функції.

26 // Запит на кредит

```
function requestLoan(uint256 _loanAmount, uint256 _interestRate, uint256 _loanTerm) public {
        Loan memory newLoan = Loan({
            loanAmount: _loanAmount,
            interestRate: _interestRate,
            loanTerm: _loanTerm,
            loanStatus: 0,
            repaymentStatus: 0,
            repaidAmount: 0
        });

        loans[msg.sender] = newLoan;
        emit LoanRequested(msg.sender, _loanAmount, _interestRate, _loanTerm);
    }
```

Цей блок коду є функцією requestLoan, яка дозволяє користувачам запитувати позику. Ось що робить кожен рядок:

function requestLoan(uint256 _loanAmount, uint256 _interestRate, uint256 _loanTerm) public {: Це початок оголошення функції requestLoan. Ця функція приймає суму позики, процентну ставку та строк позики як параметри. Вона є публічною, тому може бути викликана будь-ким.

Loan memory newLoan = Loan({...});: Цей рядок створює новий об’єкт Loan з введеними користувачем параметрами. Статус позики та статус погашення встановлюються як 0, а сплачена сума – як 0.

loans[msg.sender] = newLoan;: Цей рядок додає нову позику до мапи loans за адресою викликаючого.

emit LoanRequested(msg.sender, _loanAmount, _interestRate, _loanTerm);: Цей рядок викликає подію LoanRequested і передає адресу викликаючого, суму позики, процентну ставку та строк позики як параметри.

}: Кінець оголошення функції.

27 // Погодження кредиту

```
function approveLoan(address _borrower) public onlyOwner {
    oans[_borrower].loanStatus = 1;
    emit LoanApproved(_borrower, loans[_borrower].loanAmount);
        }
```

Цей блок коду є функцією approveLoan, яка дозволяє власнику контракту погодити запит на позику. Ось що робить кожен рядок:

function approveLoan(address _borrower) public onlyOwner {: Це початок оголошення функції approveLoan. Ця функція приймає адресу позичальника як параметр. Вона є публічною, тому може бути викликана будь-ким, але модифікатор onlyOwner обмежує її виклик тільки власником контракту.

loans[_borrower].loanStatus = 1;: Цей рядок змінює статус позики позичальника на 1, що означає, що позика погоджена.

emit LoanApproved(_borrower, loans[_borrower].loanAmount);: Цей рядок викликає подію LoanApproved і передає адресу позичальника та суму позики як параметри.

}: Кінець оголошення функції.

28 // Повернення кредиту

```
function repayLoan(uint256 _repaidAmount) public {
    require(loans[msg.sender].loanStatus == 1, "Your loan is not approved yet");
    loans[msg.sender].repaidAmount += _repaidAmount;
    if (loans[msg.sender].repaidAmount >= loans[msg.sender].loanAmount + loans[msg.sender].loanAmount * loans[msg.sender].interestRate / 100) {
        loans[msg.sender].repaymentStatus = 1;
        }
    emit LoanRepaid(msg.sender, _repaidAmount, loans[msg.sender].repaymentStatus);
        }
```

Цей блок коду є функцією repayLoan, яка дозволяє позичальникам повертати позику. Ось що робить кожен рядок:

function repayLoan(uint256 _repaidAmount) public {: Це початок оголошення функції repayLoan. Ця функція приймає суму погашення як параметр. Вона є публічною, тому може бути викликана будь-ким.

require(loans[msg.sender].loanStatus == 1, "Your loan is not approved yet");: Цей рядок перевіряє, чи погоджена позика позичальника. Якщо це не так, виклик функції припиняється, і повертається повідомлення про помилку “Your loan is not approved yet”.

loans[msg.sender].repaidAmount += _repaidAmount;: Цей рядок додає суму погашення до загальної суми погашення позичальника.

if (loans[msg.sender].repaidAmount >= loans[msg.sender].loanAmount + loans[msg.sender].loanAmount * loans[msg.sender].interestRate / 100) {...}: Цей рядок перевіряє, чи була повністю сплачена сума позики разом з відсотками.

loans[msg.sender].repaymentStatus = 1;: Якщо позика повністю сплачена, цей рядок змінює статус погашення на 1, що означає, що позика повністю сплачена.

emit LoanRepaid(msg.sender, _repaidAmount, loans[msg.sender].repaymentStatus);: Цей рядок викликає подію LoanRepaid і передає адресу позичальника, суму погашення та статус погашення як параметри.

}: Кінець умовного блоку.

}: Кінець оголошення функції.

29 // Отримання боргу

```
function getDebt(address _borrower) public view returns (uint256) {
return loans[_borrower].loanAmount + loans[_borrower].loanAmount * loans[_borrower].interestRate / 100 - loans[_borrower].repaidAmount;
}
}
```

Цей блок коду є функцією getDebt, яка дозволяє отримати загальну суму боргу позичальника. Ось що робить кожен рядок:

function getDebt(address _borrower) public view returns (uint256) {: Це початок оголошення функції getDebt. Ця функція приймає адресу позичальника як параметр. Вона є публічною та має тип view, що означає, що вона не змінює стан контракту. Функція повертає беззнакове ціле число (uint256).

return loans[_borrower].loanAmount + loans[_borrower].loanAmount * loans[_borrower].interestRate / 100 - loans[_borrower].repaidAmount;

Цей рядок обчислює та повертає загальну суму боргу позичальника. Вона складається з суми позики, доданої до суми позики, помноженої на процентну ставку (відсотки діляться на 100, щоб перетворити їх з процентів у десятковий формат), і від цього віднімається сплачена сума.

}: Кінець оголошення функції.

}: Кінець оголошення контракту.

Наукова новизна: удосконалено підхід до регулювання відносин між користувачами в децентралізованій платформі, шляхом створення моделі базового смарт-контракту на платформі Ethereum для надання мікрокредитів з використанням ефективного методу блокчейн.

Висновки. Фінансовий бар'єр є основним чинником, що обмежує доступ до якісної освіти для багатьох людей. Вища освіта може бути дорогою, і студенти часто мають обмежену можливість отримати кредит в банку або визначити джерело фінансування.

Нами було досліджено технологію блокчейну для мікрокредитування і фінансування в освітній сфері та створено модель, яка демонструє базовий смарт-контракт для надання мікрокредитів на Ethereum. Він включає створення застосунку для створення кредиту, фінансування, погашення та сценарії виплати і дефолту.

Запропоновано на основі розробленого Single смарт-контракту в Solidity удосконалений підхід до регулювання відносин між користувачами для мікрокредитування в децентралізованій платформі Ethereum, шляхом використання ефективного методу смарт-контрактів на блокчейн.

References:

1. Grech, A., Camilleri, A. (2017) Blockchain in Education, EUR 28778 EN, Publications Office of the European Union, Luxembourg, DOI: https://doi.org/10.2760/60649
2. Arndt, T. (September 17–19, 2019) An Overview of Blockchain for Higher Education. In Proceedings of the IC3K 2019–11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, Vienna, Austria, Pp. 231–235. DOI: https://doi.org/10.5220/0008343902310235
3. Min L., Bin G. (2022) Online teaching research in universities based on blockchain. *Education and Information Technologies*, pp. 6459–6482. DOI: https://doi.org/10.1007/s10639-022-10889-w
4. Fernandez-Carames T.M., Fraga-Lamas P. (2019) Towards Next Generation Teaching, Learning, and Context-Aware Applications for Higher Education: A Review on Blockchain, IoT, Fog and Edge Computing Enabled Smart Campuses and Universities. *Applied Sciences*, no. 9(21). DOI: https://doi.org/10.3390/app9214479
5. Altinay F., Beyatli O., Dagli G., Altinay Z. (2020) The role of Edmodo model for professional development: The uses of blockchain in school management. *International Journal of Emerging Technologies in Learning (iJET),* no. 15(12), pp. 256–270. DOI: https://doi.org/10.3991/ijet.v15i12.13571
6. Cheriguene A., TaiKabache T., Adnane A., Kerrache C.A., Farhan Ahmad F. (2022) On the use of Blockchain Technology for Education during Pandemics. DOI: https://doi.org/10.1109/MITP.2021.3066252
7. Khan M., Naz T. (2021) Smart Contracts Based on Blockchain for Decentralized Learning Management System. *Sn comput. sci.* vol. 2. DOI: https://doi.org/10.1007/s42979-021-00661-1
8. Al-Zoubi A.Y., Dmour M., Aldmour R. (2022) Blockchain as a Learning Management System for Laboratories 4.0. *International Journal of Online and Biomedical Engineering (iJOE),* no. 18(12), pp. 16–34. DOI: https://doi.org/10.3991/ijoe.v18i12.33515
9. Purnama S., Aini Q., Rahardja U., Santoso N. P. L., Millah S. (2021) Design of Educational Learning Management Cloud Process with Blockchain 4.0 based E-Portfolio. *Journal of Education Technology*, no. 5(4), pp. 628–635. DOI: https://doi.org/10.23887/jet.v5i4.40557
10. Räther W., Kolvenbach S., Ruland R., Schutte J., Torres C., Wendland F. (2018) Blockchain for education: lifelong learning passport. In Proceedings of 1st ERCIM Blockchain workshop 2018. European Society for Socially Embedded Technologies (EUSSET). DOI: https://doi.org/10.18420/blockchain2018_07
11. Alharby M. and Van Moorsel A. (2017) Blockchain-based smart contracts: A systematic mapping study. arXiv preprint.
12. List of Top Smart Contracts Platforms 2023. Available at: https://www.trustradius.com/smart-contracts
13. 13 What are the 5 Best Smart Contract Platforms for 2023? Available at: https://www.devteam.space/blog/what-are-the-5-best-smart-contract-platforms-for-2022/
14. List of Top 5 Smart Contract Development Platforms. Available at: https://cryptosoftwares.com/top-smart-contract-development-platforms/
15. 10 Best Tools for Smart Contract Development. Available at: https://101blockchains.com/best-smart-contract-development-tools/
16. How to Create a Secure Smart Contract: 7 Best Practices. Available at: https://blaize.tech/article-type/web3-security/

how-to-create-and-deploy-a-smart-contract-in-a-secure-way/
17. A deep dive into the 5 popular smart contract development platforms. Available at: https://cointelegraph.com/learn/smart-contract-development-platforms
18. Security considerations-solidity 0.4.18 documentation. Available at: http://solidity.readthedocs.io/en/develop/security-considerations.html; https://znayshov.com/FR/11597/%D0%9E%D1%81%D0%B2%D1%96%D1%82%D0%B0_4_65_2021-28-30.pdf
19. Solidity Tutorial: How to Use Remix IDE for Solidity Smart. Available at: https://medium.com/coinmonks/solidity-tutorial-how-to-use-remix-ide-for-solidity-smart-contract-development-d0d2ce6da051
20. Learn Solidity – A Handbook for Smart Contract Development. Available at: https://www.freecodecamp.org/news/learn-solidity-handbook/
21. MetaMask Testnet Wallet Setup for Blockchain Development. Available at: https://devtonight.com/articles/metamask-testnet-wallet-setup-for-blockchain-develop