Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет                    Комп’ютерної інженерії та управління

(повна назва)

Кафедра                           Безпеки інформаційних технологій

(повна назва)

**КВАЛІФІКАЦІЙНА РОБОТА**

**Пояснювальна записка**

рівень вищої освіти                                     перший (бакалавр)

    Безпечне проєктування смартконтрактів і механізми захищення облікових

                                            даних у системі Ethereum

Виконав:

студент   4   курсу, групи    КБІКСу-20-1

   Юрченко Арсеній Євгенович

                          (прізвище, ініціали)

Спеціальність    125 «Кібербезпека»

                          (код і повна назва спеціальності)

Освітня програма «Безпека інформаційних

і комунікаційних систем»

                              (повна назва освітньої програми)

Керівник    асист. Шафоростов М. О.

                              (посада, прізвище, ініціали)

Допускаю до захисту.

Завідувач кафедри        \_\_\_\_\_\_\_\_\_\_\_\_    Халімов Г. З.

                                         (підпис)     (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет    Комп’ютерної інженерії та управління

Кафедра    Безпеки інформаційних технологій

Рівень вищої освіти    перший (бакалавр)

Спеціальність    125 «Кібербезпека»

(код і повна назва)

Освітня програма    «Безпека інформаційних і комунікаційних систем»

(повна назва)

ЗАТВЕРДЖУЮ.

Зав. кафедри \_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

«       » 20 р.

**ЗАВДАННЯ**

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові    Юрченку Арсенію Євгеновичу

(прізвище, ім’я, по батькові)

1. Тема роботи:    «Безпечне проєктування смартконтрактів і механізми захищення

облікових даних у системі Ethereum».

Затверджена наказом по університету від  26    05   2023  р. №   554Ст .

2. Термін подання студентом роботи до екзаменаційної комісії:   14     06    2023 р.

3. Вихідні дані до роботи:

Buterin V. Ethereum White Paper: A Next-Generation Smart Contract and Decentralized Application Platform. 2014;

Antonopoulos A., Wood G. Mastering Ethereum: Building Smart Contracts and Dapps. O’Reilly Media, 2018;

Finck M. Blockchain Regulation and Governance in Europe. Cambridge University Press, 2018.

4. Перелік питань, які потрібно опрацювати в роботі:

види атак на смартконтракти та методи запобігання їм;

механізми захищення даних у межах смартконтрактів у системі Ethereum;

правовий статус смартконтрактів у світі.

5. Матеріали із зазначенням креслеників, схем, плакатів, комп’ютерних ілюстрацій (слайдів): презентаційний матеріал у формі слайдів.

6. Календарний план:

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів виконання дипломного проєкту (роботи) | Строк виконання етапів роботи | Примітка |
| 1 | Вибрання теми кваліфікаційної роботи | 03.04.2023 |  |
| 2 | Затвердження плану та завдання кваліфікаційної роботи | 05.05.2023 |  |
| 3 | Оформлення пояснювальної записки до кваліфікаційної роботи | 08.06.2023 |  |
| 4 | Здання роботи керівнику | 08.06.2023 |  |
| 5 | Підписання кваліфікаційної роботи в керівника | 12.06.2023 |  |
| 6 | Перевірення та підписання кваліфікаційної роботи у нормоконтролера | 12.06.2023 |  |
| 7 | Проходження перевірки на оригінальність кваліфікаційної роботи | 13.06.2023 |  |
| 8 | Отримання допуску до захисту від завідувача кафедри | 13.06.2023 |  |
| 9 | Захищення кваліфікаційної роботи | 14.06.2023 |  |

7. Дата видачі завдання   05     05   2023 р.

Студент   Юрченко А. Є.

                                    (підпис)

Керівник роботи   Шафоростов М. О.

                                    (підпис)             (посада, прізвище, ініціали)

**РЕФЕРАТ**

Кваліфікаційна робота містить 55 сторінок, 20 рисунків, 10 додатків, 10 джерел посилання.

БEЗПEКA СМАРТКОНТРАКТІВ, БЛОКЧЕЙН, ДЕЦЕНТРАЛІЗОВАНІ ЗАСТОСУНКИ, СМАРТКОНТРАКТ, ETHEREUM, SOLIDITY.

Об’єкт дослідження — процеси створення та аналізу властивостей смартконтрактів на платформі Ethereum, які використовуються в системах з відкритим джерельним кодом для реалізації децентралізованих фінансових операцій.

Предмет дослідження — алгоритми створення смартконтрактів і методи захищення облікових даних у межах смартконтрактів від різних типів атак.

Мета роботи — розглянути основи створення смартконтрактів у Ethereum, проаналізувати потенційні проблеми безпеки та виробити методику для їх запобігання та виявлення, а також обґрунтувати вибір інструментів і практик для розробляння безпечних смартконтрактів.

Методи досліджень базуються на використанні принципів криптографії, програмування, блокчейн-технологій, а також методів аналізу і тестування програмного забезпечення.

Проведено аналіз архітектури смартконтрактів, механізму роботи віртуальної машини Ethereum, а також різних видів атак на смартконтракти. Наведено аналіз значущих інцидентів, пов’язаних із безпекою смартконтрактів. Підготовлено рекомендації щодо розробляння та використання безпечних смартконтрактів. Практичною частиною роботи є аналіз і оцінка безпеки конкретного набору смартконтрактів, які використовуються в проєкті StudyChain.

**ABSTRACT**

The qualifying work contains 55 pages, 20 figures, 10 appendices, 10 reference sources.

BLOCKCHAIN, DECENTRALIZED APPLICATIONS, ETHEREUM, SMART CONTRACT, SMART CONTRACT SECURITY, SOLIDITY.

The object of study is the processes of creating and analyzing the properties of smart contracts on the Ethereum platform, which are used in open source systems to implement decentralized financial operations.

The subject of the study is algorithms for creating smart contracts and methods for protecting accounting data within smart contracts from various types of attacks.

The purpose of the work is to examine the basics of creating smart contracts in Ethereum, to analyze potential security issues and to develop a methodology for their prevention and detection, as well as to justify the choice of tools and practices for developing secure smart contracts.

Research methods are based on the use of principles of cryptography, programming, blockchain technology, as well as methods of analysis and testing of software.

An analysis of smart contracts, their architecture, the mechanism of the Ethereum virtual machine, as well as various types of attacks on smart contracts was carried out. An analysis of significant incidents related to the security of smart contracts was provided. Recommendations for the development and use of secure smart contracts have been prepared. The practical part of the work is an analysis and assessment of the security of a specific set of smart contracts used in the StudyChain project.

**ЗМІСТ**

[ВСТУП 8](#_Toc137578837)

[1 КОНЦЕПЦІЯ ТА ПРИНЦИПИ ВИКОНАННЯ СМАРТКОНТРАКТІВ У СИСТЕМІ ETHEREUM 10](#_Toc137578838)

[1.1 Визначення й історія смартконтрактів 10](#_Toc137578839)

[1.2 Основні характеристики смартконтрактів 12](#_Toc137578840)

[1.3 Принципи виконання смартконтрактів на платформі Ethereum 14](#_Toc137578841)

[2 ПРАВОВИЙ СТАТУС СМАРТКОНТРАКТІВ У РІЗНИХ КРАЇНАХ СВІТУ 17](#_Toc137578842)

[2.1 Правове регулювання смартконтрактів у США 17](#_Toc137578843)

[2.2 Правове регулювання смартконтрактів у ЄС 18](#_Toc137578844)

[2.3 Правове регулювання смартконтрактів у країнах Азії 19](#_Toc137578845)

[2.4 Порівняння правового регулювання смартконтрактів у різних країнах 20](#_Toc137578846)

[3 ОСОБЛИВОСТІ ПРОГРАМУВАННЯ СМАРТКОНТРАКТІВ МОВОЮ SOLIDITY 23](#_Toc137578847)

[3.1 Основи мови програмування Solidity 23](#_Toc137578848)

[3.2 Розгортання та виконання смартконтрактів на Solidity 27](#_Toc137578849)

[3.3 Взаємодіяння смартконтрактів із зовнішніми застосунками 29](#_Toc137578850)

[3.4 Інструменти для розробляння та виконування смартконтрактів 32](#_Toc137578851)

[4 РОЗРОБЛЕННЯ ТА ВПРОВАДЖЕННЯ ДЕЦЕНТРАЛІЗОВАНОГО ВЕБЗАСТОСУНКУ НА ОСНОВІ СМАРТКОНТРАКТУ 37](#_Toc137578852)

[4.1 Планування проєкту 37](#_Toc137578853)

[4.2 Розроблення смартконтракту 38](#_Toc137578854)

[4.3 Тестування смартконтракту 39](#_Toc137578855)

[4.4 Налаштування конфігурації та структури проєкту 41](#_Toc137578856)

[4.5 Розроблення клієнтської частини децентралізованого вебзастосунку 43](#_Toc137578857)

[4.6 Розгорнення смартконтракту та вебзастосунку 47](#_Toc137578858)

[5 ПРОБЛЕМИ БЕЗПЕКИ, ЯКІ ВИНИКАЮТЬ У РОЗРОБКАХ СМАРТКОНТРАКТІВ 50](#_Toc137578859)

[5.1 Загальні проблеми безпеки смартконтрактів 50](#_Toc137578860)

[5.2 Типові атаки на смартконтракти та способи запобігання їм 51](#_Toc137578861)

[5.3 Приклади значущих інцидентів із безпекою смартконтрактів у системі Ethereum 52](#_Toc137578862)

[ВИСНОВКИ 54](#_Toc137578863)

[ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 55](#_Toc137578864)

[ДОДАТОК А 56](#_Toc137578865)

[ДОДАТОК Б 58](#_Toc137578866)

[ДОДАТОК В 61](#_Toc137578867)

[ДОДАТОК Г 63](#_Toc137578868)

[ДОДАТОК Д 67](#_Toc137578869)

[ДОДАТОК Е 69](#_Toc137578870)

[ДОДАТОК Ж 71](#_Toc137578871)

[ДОДАТОК И 73](#_Toc137578872)

[ДОДАТОК К 75](#_Toc137578873)

[ДОДАТОК Л 79](#_Toc137578874)

# ВСТУП

У сучасному світі технології швидко розвиваються, надаючи людям безліч можливостей та засобів для спілкування, розваги, навчання та роботи. Зокрема, блокчейн-технології стали одним з найбільш перспективних і впливових напрямків, що впливають на відносини між людьми, державами, і компаніями. Ці технології революціонізують традиційні підходи до ведення бізнесу, фінансових операцій та керування даними, адже вони надають надійність, прозорість та ефективність у вирішенні різноманітних завдань.

Одним із ключових компонентів успішної реалізації блокчейн-технологій є смартконтракти, які дозволяють створювати автоматизовані договори, що виконуються без посередників. Смартконтракти відкривають нові можливості в різних сферах, таких як фінанси, логістика, нерухомість, охорона авторських прав та навіть голосування. Система Ethereum, що базується на блокчейні, є одним з найвідоміших прикладів використання смартконтрактів. Її гнучкість та можливості розширення дозволяють розробникам створювати різноманітні застосунки на основі смартконтрактів.

Враховуючи величезний потенціал смартконтрактів та системи Ethereum, вивчення їх особливостей, проблем та можливостей стає вкрай актуальним для науковців, розробників, юристів та представників бізнесу. Дослідження цих питань дасть можливість краще зрозуміти тенденції розвитку технологій, виявити їх переваги та недоліки, а також знайти способи оптимізації та покращення існуючих рішень на базі смартконтрактів.

Однак поряд з численними перевагами смартконтракти та система Ethereum мають свої виклики та обмеження. Вони включають питання правового регулювання та правового статусу смартконтрактів, безпеки розробки та використання, а також проблеми масштабування та швидкодії. Вивчення цих аспектів є необхідним для розробки ефективних стратегій керування ризиками та формування рекомендацій щодо впровадження смартконтрактів у різних відносинах.

Також важливим аспектом є розробка смартконтрактів, що залежить від вибору мови програмування, налагодження взаємодії із зовнішніми застосунками та платформами, а також дотримання кращих практик розробки для забезпечення стабільності та безпеки коду.

У цій роботі буде розкрито зазначені вище питання та проаналізовано особливості застосування смартконтрактів в системі Ethereum. Буде розглянуто актуальні тенденції, дослідимо різні аспекти смартконтрактів, а також надамо рекомендації щодо оптимізації їх використання. Це допоможе створити загальне розуміння особливостей смартконтрактів в системі Ethereum та їх потенціалу в різних сферах життя людей, компаній та держав.

# 1 КОНЦЕПЦІЯ ТА ПРИНЦИПИ ВИКОНАННЯ СМАРТКОНТРАКТІВ У СИСТЕМІ ETHEREUM

## 1.1 Визначення й історія смартконтрактів

Смартконтракти є цифровими протоколами, які спрощують, автоматизують та забезпечують виконання умов контракту між сторонами. Вони дозволяють виконувати транзакції та угоди без потреби в посередниках, які зазвичай забезпечують довіру між сторонами. Смартконтракти базуються на блокчейн-технології, що забезпечує надійність, безпеку та прозорість їх виконання.

Ідею смартконтрактів вперше запропонував у 1994 році криптограф та комп’ютерний науковець Нік Сабо. Він описав смартконтракти як протоколи, які автоматизують виконання угод на основі прямих та об’єктивних правил, вбудованих у код контракту [1]. Проте на той час не існувало відповідної технологічної бази для реалізації смартконтрактів.

Вирішення цієї проблеми наблизила у 2009 році поява блокчейн-системи, яку запропонував Сатоші Накамото, — криптовалюти Біткоїн [2]. Однак Біткоїн мав обмежені можливості для створення смартконтрактів. Це спонукало дослідників та розробників до пошуку нових підходів та технологій для реалізації смартконтрактів.

У 2013 році Віталік Бутерін, співзасновник та головний розробник системи Ethereum, представив концепцію нової блокчейн-платформи з універсальним механізмом для реалізації смартконтрактів [3]. Система Ethereum була офіційно запущена у 2015 році та надала широкі можливості для реалізації смартконтрактів. Вона дозволила розробникам створювати різноманітні децентралізовані застосунки (dApps) та проєкти, які здійснюються на основі смартконтрактів.

Однією з ключових особливостей смартконтрактів є те, що вони працюють на децентралізованих мережах, таких як Ethereum. Це означає, що вони не залежать від одного центрального сервера чи організації, а виконуються розподіленими вузлами мережі, що забезпечують надійність та безпеку виконання контрактів.

Смартконтракти працюють на основі принципу «якщо–тоді», де умови та дії, які мають відбутися при їх виконанні, програмуються безпосередньо у коді контракту. Це дозволяє автоматично виконувати взаємопов’язані транзакції, коли виконуються певні умови, без потреби в людському втручанні.

З ростом популярності смартконтрактів стали з’являтися різні платформи та інструменти для їх реалізації. Наприклад, було розроблено мову програмування Solidity, що спеціально призначена для створення смартконтрактів на платформі Ethereum. Сучасні розробники та організації можуть використовувати ці інструменти для створення різноманітних dApps та сервісів, забезпечуючи зручність, ефективність та децентралізацію.

З розвитком смартконтрактів та їх широким застосуванням у різних галузях, таких як фінанси, страхування, логістика, нерухомість та інші, стали з’являтися проблеми, пов’язані з їх правовим статусом, регулюванням та стандартами безпеки. У зв’язку з цим ряд країн почав активно розробляти законодавчі норми та регуляторні рамки для смартконтрактів, що сприяють їхньому легітимному використанню та інтеграції в національні правові системи.

Одним з прикладів успішної адаптації смартконтрактів на законодавчому рівні є Сполучені Штати Америки (США), де деякі штати вже прийняли закони, що визнають смартконтракти як правомірні та обов’язкові до виконання. Це допомагає забезпечити юридичну опору та впевненість у використанні смартконтрактів як альтернативи традиційним контрактам.

Тим не менш, застосування смартконтрактів у системі Ethereum також має свої виклики та проблеми. Наприклад, можуть виникати питання щодо безпеки, масштабування та швидкодії системи, а також необхідності вдосконалення механізмів керування та голосування всередині децентралізованих організацій. Для вирішення цих проблем розробники Ethereum працюють над постійними оновленнями платформи та впровадженням нових технологій, таких як шардинг, протоколи стійкості до відмов та інші рішення.

Незважаючи на виклики, смартконтракти на платформі Ethereum продовжують набирати обертів, перетворюючи багато традиційних бізнес-процесів та галузей на більш автоматизовані, прозорі та ефективні системи. Вони відкривають нові можливості для розвитку децентралізованих технологій та сприяють створенню глобальної, відкритої та взаємозв’язаної економіки.

## 1.2 Основні характеристики смартконтрактів

Смартконтракти є інноваційною технологією, яка змінює спосіб укладання та виконання угод у різних галузях. Вони мають ряд ключових характеристик, що відрізняють їх від традиційних контрактів:

* автоматизація;

Смартконтракти автоматично виконуються та контролюються комп’ютерною програмою, що дозволяє зменшити витрати на адміністрування та виконання угод.

* незмінність;

Смартконтракти записуються на блокчейні, що гарантує їх незмінність та неможливість зміни умов контракту без згоди всіх сторін.

* децентралізованість;

Смартконтракти виконуються в децентралізованій мережі, як Ethereum, що позбавляє їх залежності від одного центрального органу чи сервера.

* прозорість;

Умови смартконтрактів доступні для перегляду всіма учасниками мережі, що забезпечує високий рівень прозорості та довіри між сторонами.

* ефективність.

Завдяки автоматизації, смартконтракти можуть знизити витрати на проведення та виконання угод, в той же час підвищуючи швидкість виконання.

Ці характеристики роблять смартконтракти привабливими для ряду застосувань та галузей. Вони можуть бути використані для створення децентралізованих фінансових систем, в яких позики, депозити та страхові поліси можуть бути автоматично укладені та виконані без посередництва традиційних банків або фінансових установ.

Смартконтракти також можуть бути використані в логістиці та постачанні, де вони можуть автоматизувати процеси, пов’язані з відстеженням вантажів, виконанням оплати та підтвердженням доставки товарів. У нерухомості смартконтракти можуть спростити процес передачі права власності, автоматично виконуючи угоди та перевіряючи документи.

Основні характеристики смартконтрактів роблять їх потужним інструментом для розвитку технологій та галузей в умовах децентралізації, автоматизації та прозорості. Завдяки їх широким можливостям, смартконтракти продовжують залучати увагу користувачів та розробників по всьому світу.

Смартконтракти надають можливості для реалізації dApps, які можуть функціонувати відкрито, безпечно та автономно на платформах, таких як Ethereum. dApps можуть мати різні форми, від простих голосувань та лотерей до складних платформ для децентралізованого фінансування та керування компаніями.

Одним з ключових аспектів смартконтрактів є їхня взаємодія з цифровими активами на платформі Ethereum. За допомогою смартконтрактів користувачі можуть створювати токени, які представляють цінності або активи, та передавати їх іншим учасникам мережі, створюючи екосистеми власних криптовалют та децентралізованих економічних систем.

Загалом, смартконтракти мають ряд основних характеристик, які роблять їх важливим інструментом для розвитку блокчейн-технологій та децентралізованих систем. Вони пропонують автоматизацію, непохідність, децентралізацію, прозорість та ефективність, що відкриває нові можливості для різних галузей та застосувань.

## 1.3 Принципи виконання смартконтрактів на платформі Ethereum

Смартконтракти в системі Ethereum використовують блокчейн для забезпечення децентралізованого, безпечного та прозорого виконання угод. Розглянемо основні принципи виконання смартконтрактів на платформі Ethereum.

Віртуальна машина Ethereum (Ethereum virtual machine, EVM) є середовищем, в якому виконуються смартконтракти в мережі Ethereum. EVM є механізмом, який дозволяє розробникам створювати та виконувати dApps у блокчейні Ethereum [4]. Коли користувач відправляє транзакцію до смартконтракту, EVM виконує цей код у контексті цієї транзакції, обробляючи вхідні дані, які були відправлені разом з транзакцією. Все це відбувається в ізольованому середовищі, яке відокремлене від основної мережі Ethereum, що дозволяє EVM безпечно виконувати код, незалежно від зовнішнього світу. Кожна операція, що виконується на EVM, вимагає певної кількості газу. Газ — це внутрішня вартість виконання операцій на Ethereum, і він вимірюється в ether (ETH). Операції, які вимагають більше обчислювальних ресурсів, вартують більше газу. Це вимога газу регулює використання ресурсів мережі і захищає її від зловживань. Схема роботи EVM зображена на рис. 1.

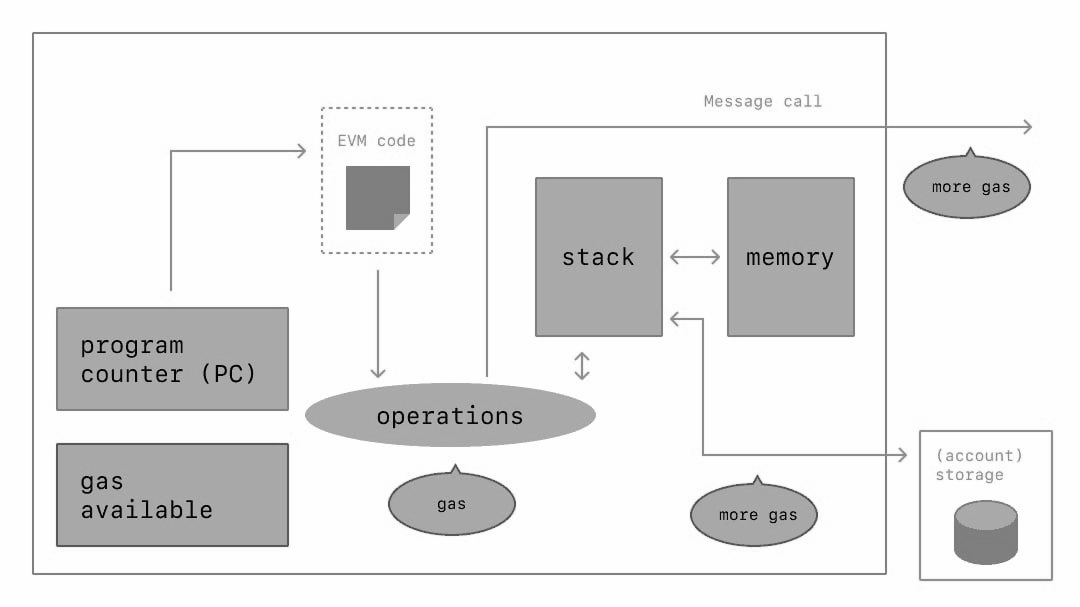


Рисунок 1 — Схема роботи EVM

Газ є внутрішнім механізмом виміру ресурсів, необхідних для виконання операцій на платформі Ethereum, включаючи створення, виконання та взаємодію з смартконтрактами. Газ вимірюється в одиницях та має певну вартість, яка виражається в криптовалюті Ether (ETH) .

Смартконтракти в Ethereum містять функції, які виконують певні дії або завдання. Функції можуть виконуватись автоматично або в результаті взаємодії з іншими контрактами або користувачами. Кожна функція має певні вхідні та вихідні параметри та може змінювати стан смартконтракту.

Тригери смартконтрактів — це події, які активують виконання певних функцій смартконтракту. Тригери можуть бути зовнішніми (наприклад, взаємодія з користувачем або іншим контрактом) або внутрішніми (наприклад, виконання певної умови або досягнення певного часу). Тригери дозволяють смартконтрактам реагувати на зміни умов та виконувати відповідні дії відповідно до вбудованих логічних правил.

Смартконтракти в Ethereum мають певний стан, який зберігається на блокчейні. Стан смартконтракту може включати різні дані, такі як баланси рахунків, інформацію про власників та інші змінні, пов’язані з логікою контракту. Зміни стану смартконтракту здійснюються в результаті виконання функцій та взаємодії з іншими контрактами або користувачами.

Для забезпечення децентралізації та безпеки в системі Ethereum використовується механізм консенсусу, який дозволяє учасникам мережі погоджуватись на правильність виконання смартконтрактів та стан блокчейну. До 15.09.2022 р. Ethereum використовував механізм консенсусу Proof of Work (PoW), але з оновленням «The Merge» здійснив перехід на Proof of Stake (PoS).

PoS забезпечує безпеку та стабільність мережі, залучаючи учасників до процесу валідації транзакцій та формування блоків на основі їхньої «ставки» у формі криптовалюти Ether (ETH). Цей підхід замінює попередній механізм Proof of Work (PoW), який вимагав значних обчислювальних потужностей та енергоспоживання.

Зміна консенсусного механізму на PoS також впливає на роботу смартконтрактів, забезпечуючи швидше та екологічно стале виконання угод та децентралізованих застосунків на платформі Ethereum.

Загалом, смартконтракти на платформі Ethereum виконуються в децентралізованому середовищі за допомогою віртуальної машини Ethereum (EVM). Основні принципи виконання смартконтрактів включають використання газу, функцій, тригерів, стану та механізмів консенсусу, які спільно дозволяють створювати автоматичні, безпечні та прозорі взаємодії між учасниками мережі.

# 2 ПРАВОВИЙ СТАТУС СМАРТКОНТРАКТІВ У РІЗНИХ КРАЇНАХ СВІТУ

## 2.1 Правове регулювання смартконтрактів у США

Правове регулювання смартконтрактів у США знаходиться на етапі розвитку, але вже встановлені певні правила та норми, які регулюють сферу криптовалют та технологій блокчейн, включаючи смартконтракти.

На федеральному рівні, смартконтракти регулюються кількома установами. Наприклад, Комісія з цінних паперів та бірж (SEC) має право регулювати смартконтракти, якщо вони відносяться до цінних паперів, зокрема у сфері ICO (первинних розміщень монет) та токенів [5].

Крім того, Фінансові злочини мережі правоохоронних органів (FinCEN) можуть відстежувати смартконтракти, якщо вони використовуються для передачі віртуальних валют, які підпадають під дію банківського секретаріату [6].

Деякі штати у США також розглядають смартконтракти в контексті своїх законодавчих актів. Наприклад, штат Аризона в 2017 році прийняв закон, який визнає смартконтракти як законні угоди. Закон забезпечує, що смартконтракти мають те ж саме правове значення, що й традиційні контракти, якщо вони відповідають всім необхідним вимогам.

Оскільки смартконтракти є новим і відносно маловивченим явищем, визначення їх правового статусу у США є складним завданням. Загалом, смартконтракти можуть вважатися дійсними та обов’язковими угодами, якщо вони відповідають основним вимогам до угод, встановленим у договірному праві. Це включає волевиявлення сторін, предмет угоди, умови виконання та відповідність законодавству. У контексті смартконтрактів можливі юридичні питання, такі як відповідальність сторін за помилки або зловживання, встановлення особи, що несе відповідальність за виконання смартконтракту, а також можливість вирішення суперечок у судовому порядку.

В майбутньому правове регулювання смартконтрактів у США, ймовірно, буде розвиватися разом із розвитком технології блокчейн та криптовалют. Це може включати розширення дії існуючих законодавчих актів або створення нових норм, що спеціально стосуються смартконтрактів. Зокрема, можуть з’явитися нові норми щодо захисту споживачів, прозорості та безпеки, а також стандарти для розвитку та використання смартконтрактів.

## 2.2 Правове регулювання смартконтрактів у ЄС

У Європейському Союзі (ЄС) правове регулювання смартконтрактів та технології блокчейн також знаходиться на етапі розвитку. ЄС прагне створити єдині правила та стандарти для смартконтрактів та криптовалют, що сприятиме інтеграції та розвитку технологій на всьому континенті.

Одним з основних принципів ЄС є гармонізація законодавства між державами-членами. Це означає, що ЄС прагне до встановлення єдиних норм та правил для смартконтрактів у всіх державах-членах. Наразі, правове регулювання смартконтрактів у ЄС в основному здійснюється на національному рівні, але ЄС може впроваджувати директиви та регуляторні акти, які стосуються технології блокчейн та криптовалют, щоб сприяти гармонізації [7].

ЄС визнає потенційні переваги технології блокчейн і смартконтрактів, але також відзначає потенційні ризики, такі як відмивання грошей та фінансування тероризму. Відповідно, ЄС прагне до збалансованого підходу до регулювання смартконтрактів, який дозволить розвивати технології та забезпечити безпеку користувачів.

У 2018 році ЄС створив Європейську блокчейн-ініціативу (European Blockchain Initiative) з метою розвитку та впровадження технології блокчейн у ряді відомчих та міжнародних проєктів. Ця ініціатива сприяє розвитку смартконтрактів у ЄС, а також співпраці між державами-членами та приватним сектором. Європейська блокчейн-ініціатива спрямована на підтримку наукових досліджень, навчання та розвитку підприємництва у сфері блокчейна та смартконтрактів [8].

Національні законодавства держав-членів ЄС можуть мати різні підходи до визначення правового статусу смартконтрактів. В деяких країнах смартконтракти вважаються дійсними та обов’язковими угодами, якщо вони відповідають загальним вимогам до угод, встановленим у договірному праві. В інших країнах правовий статус смартконтрактів може бути менш визначеним або обмеженим.

В майбутньому можна очікувати, що ЄС продовжить розробляти правове регулювання смартконтрактів та блокчейн технології, щоб стимулювати інновації та забезпечити захист споживачів. Це може включати розробку нових директив або регуляторних актів, а також співпрацю з державами-членами, приватним сектором та міжнародними організаціями для підтримки та розвитку смартконтрактів та пов’язаних технологій.

## 2.3 Правове регулювання смартконтрактів у країнах Азії

Правове регулювання смартконтрактів в Азії суттєво відрізняється від країни до країни через різницю в законодавчих системах, культурних та економічних особливостей. Деякі азійські країни активно розвивають правові рамки для смартконтрактів та блокчейн технологій, зокрема, Сінгапур, Гонконг, Південна Корея та Японія.

Сінгапур відомий своєю інноваційною політикою та розвинутим законодавством у сфері фінансових технологій та криптовалют. Місцеві органи регулювання, такі як Управління з регулювання діяльності фінансових служб (MAS), активно розробляють та впроваджують правила для регулювання блокчейн технологій та смартконтрактів. В основному, смартконтракти вважаються дійсними та обов’язковими угодами, якщо вони відповідають вимогам законодавства.

Гонконг також активно розвиває правові рамки для смартконтрактів та блокчейн технологій. Уряд Гонконгу визнає потенційні переваги технології блокчейн та створює сприятливе середовище для розвитку інновацій. Гонконгські органи регулювання, такі як Управління з регулювання діяльності фінансових служб (SFC), розробляють та впроваджують регуляторні акти, які стосуються смартконтрактів та блокчейн технологій.

Південна Корея є однією з країн, які лідирують у регулюванні блокчейн технологій та смартконтрактів в Азії. Уряд Південної Кореї активно співпрацює з приватним сектором, щоб стимулювати розвиток технології блокчейн та криптовалют. Правовий статус смартконтрактів в Південній Кореї визначений, і вони вважаються дійсними та обов’язковими угодами, якщо вони відповідають вимогам законодавства.

Японія є ще однією країною Азії, яка активно розвиває правові рамки для смартконтрактів та блокчейн технологій. Уряд Японії відноситься позитивно до технології блокчейн та створює сприятливе середовище для розвитку інновацій. Смартконтракти вважаються дійсними та обов’язковими угодами в Японії, якщо вони відповідають вимогам законодавства.

В майбутньому можна очікувати, що країни Азії продовжать розробляти правове регулювання смартконтрактів та блокчейн технологій, щоб забезпечити стабільність, безпеку та інноваційність у сфері фінансових технологій. Очікується, що уряди азійських країн будуть співпрацювати на міжнародному рівні з метою гармонізації правових рамок і підтримки глобального розвитку смартконтрактів та блокчейн-технологій.

## 2.4 Порівняння правового регулювання смартконтрактів у різних країнах

Проведення порівняльного аналізу правового регулювання смартконтрактів у різних країнах дозволяє виявити спільні тенденції та відмінності у регулюванні цієї технології. Основні відмінності зазвичай зумовлені різницею в законодавчих системах, культурних та економічних особливостях країн.

Такі спільні тенденції у правовому регулюванні смартконтрактів можна виділити на основі проведеного аналізу:

* визнання смартконтрактів як дійсних та обов’язкових угод;

Більшість країн визнають смартконтракти як дійсні та обов’язкові угоди, якщо вони відповідають вимогам законодавства.

* нормативне регулювання;

У багатьох країнах органи регулювання розробляють та впроваджують нормативні акти, які стосуються смартконтрактів, криптовалют та блокчейн технологій.

* співпраця на міжнародному рівні.

Уряди різних країн співпрацюють на міжнародному рівні з метою гармонізації правових рамок і підтримки глобального розвитку смартконтрактів та блокчейн технологій.

Утім, у правовому регулюванні смартконтрактів також наявні такі відмінності:

* рівень регулювання;

У деяких країнах правові рамки для смартконтрактів розвиваються повільно, тоді як в інших країнах уряди активно сприяють інноваціям та впровадженню нових технологій.

* відмінності у правових системах;

Різні правові системи країн можуть мати відмінні підходи до регулювання смартконтрактів, в тому числі щодо їх визнання, виконання та правової відповідальності.

* застосування смартконтрактів у різних сферах;

Законодавство деяких країн може обмежувати або сприяти застосуванню смартконтрактів у певних сферах, таких як фінанси, страхування, нерухомість та інше.

* регулювання криптовалют;

Відмінності у регулюванні криптовалют, які часто використовуються в смартконтрактах, також впливають на правовий статус та регулювання смартконтрактів в різних країнах.

* вимоги до ідентифікації сторін.

У деяких країнах законодавство може вимагати ідентифікації сторін угоди, що може впливати на анонімність та конфіденційність смартконтрактів.

Узагальнюючи, можна сказати, що правове регулювання смартконтрактів у різних країнах світу продовжує розвиватися і адаптуватися до швидко змінюваних технологічних умов. Враховуючи спільні тенденції та відмінності, країни мають широкі можливості для навчання одна від одної та досягнення найкращої практики у регулюванні смартконтрактів та пов’язаних з ними технологій.

# 3 ОСОБЛИВОСТІ ПРОГРАМУВАННЯ СМАРТКОНТРАКТІВ МОВОЮ SOLIDITY

## 3.1 Основи мови програмування Solidity

Solidity є статично типізованою, об’єктно-орієнтованою мовою програмування, що використовується для написання смартконтрактів на блокчейн платформі Ethereum. Вона була створена в 2014 році командою Ethereum, яка прагнула створити мову, що була б зрозумілою для розробників, знайомих з JavaScript і іншими C-подібними (сі-подібними) мовами, але в той же час була б спеціально адаптована для написання смартконтрактів.

До основних сутностей Solidity належать такі [9]:

* типи даних;

Solidity має різні типи даних, включаючи примітивні типи (наприклад, uint для беззнакових цілих чисел, bool для булевих значень), складні типи (наприклад, масиви, структури та відображення) та користувацькі типи (наприклад, контракти та перелічення).

* функції;

Solidity використовує функції для виконання дій. Функції можуть бути внутрішніми (тобто вони можуть викликатися тільки всередині контракту) або зовнішніми (тобто вони можуть викликатися іншими контрактами або зовнішніми акаунтами).

* модифікатори;

Solidity використовує модифікатори для зміни поведінки функцій. Модифікатори дозволяють додавати додаткові умови до функцій, що можуть бути використані для перевірки умов перед виконанням функції.

* успадкування;

Solidity підтримує множинне успадкування. Це дозволяє розробникам створювати нові контракти, які успадковують властивості та функції наявних контрактів.

* змінні стану;

Solidity використовує змінні стану для зберігання значень, які повинні пережити виклики функцій. Змінні стану зберігаються безпосередньо в блокчейні Ethereum.

* безпека.

Solidity має ряд вбудованих функцій для роботи з безпекою, таких як винятки для оброблення помилок, модифікатор payable для функцій, які приймають ether, та використання модифікаторів для контролю доступу.

Solidity має такі типи даних [9]:

* bool;

Булевий тип даних, який може приймати значення true або false.

* uint;

Беззнакове ціле число. Розмір цього типу можна вказати як uint8, uint16, uint256 тощо, де число після uint вказує на кількість бітів, які використовуються для зберігання значення.

* int;

Знакове ціле число. Розмір цього типу можна вказати як int8, int16, int256 тощо.

* address;

Використовується для зберігання ether-адрес. Адреси можуть бути використані для відправки ether за допомогою вбудованої функції transfer.

* string;

Використовується для зберігання рядків.

* bytes1, bytes2, …, bytes32;

Використовуються для зберігання байтових рядків фіксованого розміру.

* bytes;

Використовується для зберігання байтових рядків довільного розміру.

* mapping;

Використовується для створення структур даних, що подібні до асоціативних масивів, де унікальний ключ відображається на значення.

Функції в Solidity використовуються для виконання дій. Вони можуть бути внутрішніми (тобто викликатися тільки всередині контракту) або зовнішніми (тобто вони можуть викликатися іншими контрактами або зовнішніми акаунтами). Ось приклад функції, яка приймає один аргумент і повертає результат (рис. 2). У цьому прикладі multiply є функцією, яка приймає одне ціле число *x* як аргумент і повертає результат множення *x* на 2.

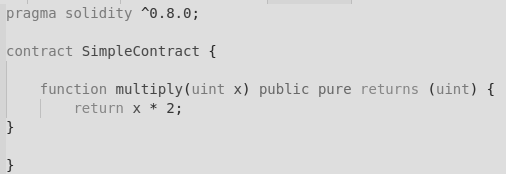


Рисунок 2 — Функція multiply

Solidity використовує модифікатори для зміни поведінки функцій. Модифікатори дозволяють додавати додаткові умови до функцій, що можуть бути використані для перевірки умов перед виконанням функції. Ось приклад модифікатора onlyOwner (рис. 3). У цьому прикладі onlyOwner є модифікатором, який перевіряє, чи є відправник повідомлення (msg.sender) власником контракту перед виконанням функції set. Якщо відправник не є власником, то викликається виняток із повідомленням “Only the contract owner can call this function” (“Тільки власник контракту може викликати цю функцію”).

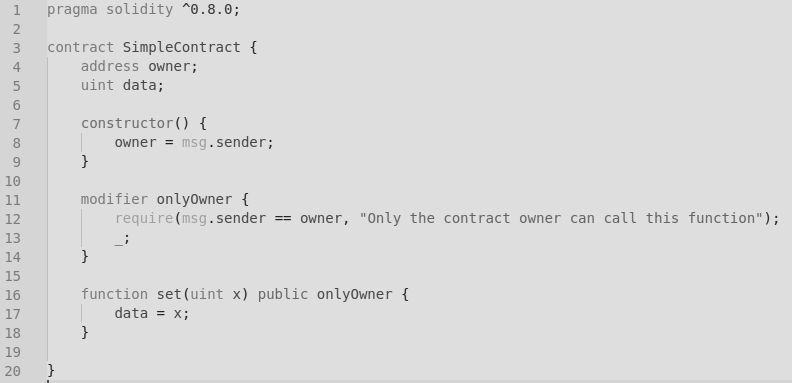


Рисунок 3 — Модифікатор onlyOwner

Специфічні для Solidity структури даних дозволяють організувати та маніпулювати комплексними типами даних. Приклад зображено на рис. 4. У цьому прикладі Player є структурою, яка складається з рядка name і цілого числа score. Масив players зберігає всіх гравців. Функція addPlayer дозволяє додавати нових гравців до масиву.

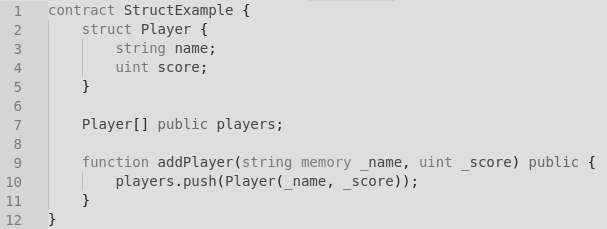


Рисунок 4 — Структура Player

Змінні стану в Solidity зберігаються безпосередньо на блокчейні Ethereum. Це означає, що їх значення залишається незмінним між окремими викликами функцій. Приклад змінної стану зображено на рис. 5. У цьому прикладі count є змінною стану. Її початкове значення встановлено як 0, але це значення може змінюватися за допомогою функції incrementCount.

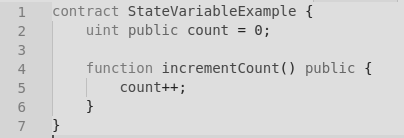


Рисунок 5 — Змінна стану count

Бібліотеки в Solidity дозволяють розбити код на менші компоненти (компоненти, які повторно використовуються), що можуть спростити розробку і підтримку смартконтрактів. Наприклад, бібліотека SafeMath від OpenZeppelin надає безпечні математичні операції, які допомагають запобігти переповненню і недостатньому заповненню. Приклад зображено на рис. 6. У цьому прикладі використовується бібліотека SafeMath для безпечного додавання двох чисел. Якщо сума виходить за межі дозволених значень для типу uint, транзакція зазнає збою, що допомагає запобігти переповненню.

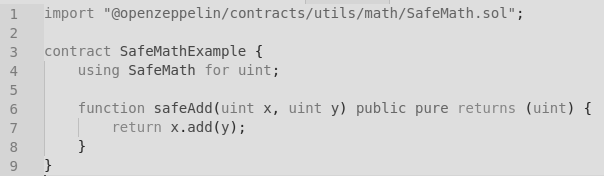


Рисунок 6 — Використання бібліотеки SafeMath

## 3.2 Розгортання та виконання смартконтрактів на Solidity

Виконання смартконтрактів на Solidity є важливою частиною екосистеми Ethereum. Цей процес включає в себе не лише само виконання коду контракту, але й його розгортання, взаємодію з контрактом та врахування різних особливостей Ethereum, таких як вартість газу, взаємодію з іншими контрактами та безпеку.

Розгортання смартконтракту на Ethereum — це процес публікації коду контракту на блокчейн. Коли контракт розгорнуто, він отримує власну адресу в блокчейні, за якою його можна знайти та з якою можна взаємодіяти.

Процес розгортання контракту вимагає компіляції коду Solidity в байткод Ethereum та відправки цього байткоду в мережу Ethereum як частину спеціальної транзакції, відомої як транзакція створення контракту. Ця транзакція також включає в себе вартість газу, яка покриває витрати на виконання транзакції [10].

Є різні інструменти та сервіси, які можуть допомогти у розгортанні смартконтрактів, включаючи Remix, Truffle, Hardhat, та інші. Вибір конкретного інструменту залежить від потреб та вподобань розробника.

Після розгортання смартконтракт може бути виконаний. Виконання смартконтракту полягає в тому, щоб викликати одну з його функцій за допомогою ether-транзакції. Ця транзакція включає в себе інформацію про адресу контракту, функцію, яку потрібно викликати, а також будь-які вхідні параметри для цієї функції. Подібно до транзакції створення контракту, транзакція виклику функції також вимагає платежу за газ для покриття витрат на виконання транзакції [10].

Коли транзакція виклику функції потрапляє в блокчейн, вона виконується всіма вузлами мережі. Це означає, що всі вузли виконують код функції контракту і оновлюють стан контракту відповідно до результатів виконання.

Після розгортання і виконання, смартконтракт може взаємодіяти з іншими контрактами та акаунтами Ethereum. Контракти можуть викликати функції інших контрактів, відправляти їм ether, або навіть створювати нові контракти.

Для взаємодії з контрактом потрібно знати його адресу та двійковий інтерфейс застосунку (application binary interface, ABI), який описує, як викликати його функції. ABI можна отримати після компіляції контракту.

## 3.3 Взаємодіяння смартконтрактів із зовнішніми застосунками

Смартконтракти у системі Ethereum мають вбудовані можливості для взаємодії із зовнішніми застосунками, що значно розширює їх потенціал. Зокрема, вони можуть спілкуватися з вебзастосунками, мобільними застосунками, іншими смартконтрактами та зовнішніми сервісами. Це стає можливим завдяки використанню таких технологій, як Web3.js, а також різних інтерфейсів програмування застосунків (application programming interface, API).

Web3.js — це бібліотека JavaScript, яка дозволяє вебзастосункам взаємодіяти з Ethereum блокчейном. Вона надає API для взаємодії з Ethereum на рівні мережі, що дозволяє виконувати такі дії, як відправка та отримання транзакцій, виклик методів смартконтрактів, отримання інформації про блоки та транзакції, а також робота з Ethereum адресами і ключами.

Використання Web3.js дозволяє розробникам створювати вебзастосунки, що взаємодіють з Ethereum блокчейном, зазвичай ці застосунки називаються dApps. dApps можуть включати все, від простих ігор до складних фінансових застосунків, які використовують смартконтракти для керування цифровими активами і виконання автоматизованих дій на основі заданих умов.

Наприклад, розглянемо створення простого dApp, який взаємодіє з смартконтрактом. По-перше, розробник повинен написати смартконтракт на Solidity і розгорнути його на Ethereum блокчейні. Після цього, вони можуть використовувати Web3.js для створення вебінтерфейсу, який викликає методи цього смартконтракту.

Для цього розробнику потрібно створити об’єкт Web3, що підключений до Ethereum ноди (наприклад, через Infura), а потім використовувати цей об’єкт для виклику методів смартконтракту.

Наприклад, вони можуть використовувати метод web3.eth.Contract для створення нового об’єкту контракту, який представляє смартконтракт на блокчейні, та методи цього об’єкту для виклику функцій контракту.

Отже, якщо маємо смартконтракт, який дозволяє користувачам ставити ставки на результат гри, вебзастосунок може використовувати Web3.js для відправки транзакцій до цього контракту з вказаною ставкою користувача. Також можливо зчитувати дані з контракту, отримувати події, які викликаються контрактом, і т. ін.

Це відкриває безмежні можливості для розробки вебзастосунків, що використовують переваги блокчейна Ethereum і смартконтрактів. Ці dApps можуть надати користувачам новий рівень контролю над їхніми даними, прозорості в операціях і незалежності від централізованих органів.

Важливо зазначити, що використання Web3.js вимагає від розробників глибокого розуміння Ethereum, Solidity і смартконтрактів, а також деяких особливостей роботи з асинхронними JavaScript API. Однак, з правильними знаннями і інструментами, розробники можуть створювати потужні, децентралізовані вебзастосунки, які використовують переваги блокчейна Ethereum.

Розглянемо приклад взаємодії вебзастосунку зі смартконтрактом на Ethereum за допомогою Web3.js. Припустимо, що маємо простий смартконтракт, який зберігає повідомлення. Контракт може бути написаний на Solidity таким чином. Приклад зображено на рис. 7. У цьому контракті є одна змінна message і одна функція setMessage, яка дозволяє змінювати це повідомлення.



Рисунок 7 — Смартконтракт для взаємодії з вебзастосунком

Тепер ми можемо використовувати Web3.js у вебзастосунку для взаємодії з цим контрактом. Простий приклад, зображений на рис. 8, показує, як можна використовувати Web3.js для взаємодії вебзастосунку з контрактом Ethereum. У цьому прикладі ми спочатку створюємо об’єкт web3, вказавши провайдера Ethereum. Потім ми створюємо об’єкт контракту, вказавши його ABI і адресу. Потім ми викликаємо функції контракту message і setMessage для читання і запису повідомлення.

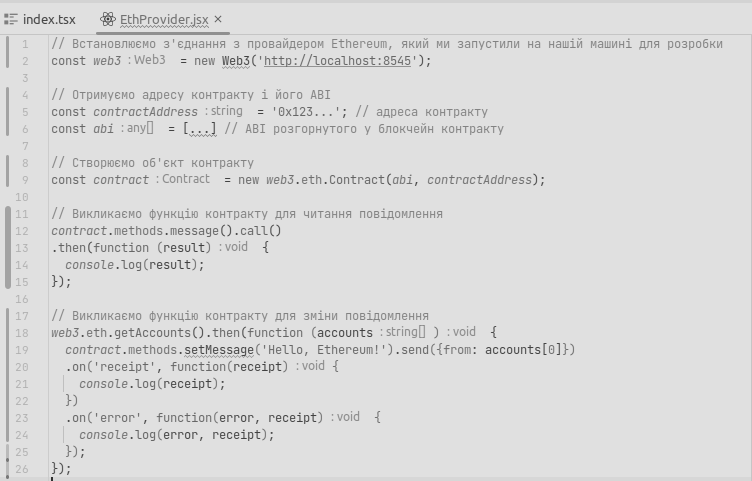


Рисунок 8 — Підключення смартконтракту до вебзастосунку

Утім, використання Web3.js у реальних застосунках може бути значно складнішим. Наприклад, розробникам може знадобитися розробити систему автентифікації для взаємодії з контрактами, обробити різні сценарії помилок, які можуть виникнути під час виконання контракту, або використовувати події Ethereum для слухання змін стану контракту.

Також можуть виникнути технічні труднощі, наприклад, з’ясування, як правильно форматувати і передавати дані до і від контракту, особливо якщо контракт використовує складні типи даних.

Враховуючи це, розробники часто використовують додаткові бібліотеки та інструменти, такі як Truffle, Drizzle або ethers.js, які надають більш високорівневий, абстрактний інтерфейс для роботи з контрактами Ethereum і можуть допомогти вирішити деякі з цих проблем.

Щодо взаємодії смартконтрактів з іншими смартконтрактами, вони можуть викликати функції один одного та надсилати Ether між собою. Для цього смартконтракт, який викликає функцію іншого контракту, повинен мати відповідні права доступу, а також знати інтерфейс контракту-одержувача.

## 3.4 Інструменти для розробляння та виконування смартконтрактів

Сучасна екосистема Ethereum надає розробникам широкий спектр інструментів та платформ, які допомагають створювати, тестувати, розгортати та взаємодіяти зі смартконтрактами. Ці інструменти мають різні функції, починаючи від локального тестування блокчейну і закінчуючи розгортанням смартконтрактів на живу мережу Ethereum. Розглянемо декілька з них.

Ganache є частиною Truffle Suite і надає локальне тестове середовище блокчейну для розробки Ethereum. Воно дозволяє розробникам створювати власний локальний блокчейн для тестування транзакцій та виконання смартконтрактів без витрат на газ або необхідності з’єднання з живою мережею Ethereum. Ganache надає користувацький інтерфейс для перегляду деталей транзакцій, блоків та стану блокчейну. Це інструмент для етапу розробки та тестування.

Remix є вебінтерфейсним середовищем для розробляння смартконтрактів на Solidity. Він надає широкий спектр функцій, включаючи редактор коду, компілятор Solidity, вбудований віртуальний блокчейн для тестування, а також інтерфейс для взаємодії зі смартконтрактами. Через Remix можна безпосередньо розгортати смартконтракти на живу мережу Ethereum або тестову мережу.

Основні можливості Remix такі:

* редагування коду;

Remix містить повнофункціональний редактор коду з підсвічуванням синтаксису, що спеціально розроблений для Solidity. Редактор також включає засоби для автоматичного форматування коду та знаходження помилок.

* компіляція та відлагодження;

Remix може компілювати смартконтракти, використовуючи різні версії компілятора Solidity. Він також має потужні інструменти для відлагодження, які дозволяють вам перевіряти стан смартконтракту, крок за кроком, під час виконання.

* тестування та розгортання;

Із Remix можна безпосередньо взаємодіяти зі смартконтрактами на Ethereum мережі. Можливо створювати транзакції, викликати функції контракту та переглядати події. Remix також дозволяє розгортати смартконтракти на різні мережі Ethereum, включаючи основну мережу, тестові мережі та віртуальну машину JavaScript.

* підтримка плагінів.

Remix підтримує плагіни, які додають нові можливості до середовища. Існують плагіни для формальної перевірки, аналізу безпеки, використання зовнішніх бібліотек Solidity та багато іншого.

Truffle є набором інструментів для розробки смартконтрактів для Ethereum. Він надає структуру проєкту, компіляцію, розгортання та тестування смартконтрактів, а також підтримку скриптів міграції. Truffle також має вбудований Mocha та Chai для юніт-тестування смартконтрактів, що робить цей інструмент незамінним для серйозних проєктів на Ethereum.

Основні можливості Truffle такі:

* розробляння та впроваджування смартконтрактів;

Truffle містить скрипти міграції, які допомагають керувати життєвим циклом смартконтракту, включаючи його розгортання та оновлення.

* тестування смартконтрактів;

Truffle підтримує написання автоматичних тестів для смартконтрактів на Javascript та Solidity, що дозволяє вам гарантувати їх правильну роботу.

* налаштовування мережних середовищ;

Truffle дозволяє налаштовувати різні мережні середовища для розгортання смартконтрактів, включаючи локальні, приватні, тестові та публічні мережі Ethereum.

* взаємодіяння зі смартконтрактами;

Truffle містить Truffle Console, інтерактивну оболонку, яка дозволяє взаємодіяти з вашими смартконтрактами безпосередньо.

* інтеграція з іншими інструментами.

Truffle інтегрується з іншими інструментами, такими як Ganache і Drizzle, щоб надати повноцінний досвід розробки dApps.

EtherScan — це блокчейн-оглядач для Ethereum, який надає інформацію про транзакції, адреси, блоки та смартконтракти в мережі Ethereum. Це особливо корисно, коли розгортаєте смартконтракт на живу мережу, оскільки ви можете відстежувати стан вашого контракту в реальному часі, перевіряти транзакції та навіть переглядати джерельний код смартконтрактів (якщо його було належно верифіковано).

MetaMask — це криптовалютний гаманець, який використовується як браузерне розширення, що дозволяє користувачам взаємодіяти з dApps у блокчейні Ethereum. Він фактично діє як міст між браузерами користувачів та Ethereum blockchain, що дозволяє проводити транзакції і взаємодіяти зі смартконтрактами.

Основні можливості MetaMask такі:

* ether-гаманець;

MetaMask дозволяє користувачам створювати та керувати ether-адресами. Користувачі можуть зберігати ether і ERC-20 токени в MetaMask.

* взаємодіяння з dApps;

MetaMask може взаємодіяти з dApps на Ethereum. Це означає, що користувачі можуть використовувати MetaMask для входу в dApps, відправки транзакцій, участі в голосуваннях та інших взаємодій з контрактами.

* безпечність;

MetaMask шифрує ваші приватні ключі і зберігає їх локально на вашому пристрої, що допомагає забезпечити вашу безпеку. Крім того, MetaMask має вбудовані заходи безпеки, які допомагають захистити користувачів від шахрайства та атак.

* відправляння й отримування ether та токенів;

Користувачі можуть відправляти та отримувати ether та ERC-20 токени використовуючи MetaMask.

* вибирання мереж.

MetaMask дозволяє користувачам переключатися між різними Ethereum мережами, включаючи основну мережу, тестові мережі та користувацькі виклики віддалених процедур (remote procedure calls, RPC).

Тестові мережі Ethereum дозволяють розробникам тестувати свої застосунки та смартконтракти в безпечному та ізольованому середовищі перед їх запуском в основній мережі Ethereum, що є важливим кроком для запобігання помилок та забезпечення безпеки.

Основні тестові мережі Ethereum такі:

* Ropsten;

Ця тестова мережа має ту саму функціональність, що і основна мережа Ethereum, включаючи Proof of Work. Це дозволяє розробникам тестувати свої застосунки в умовах, які максимально наближені до реальних.

* Rinkeby і Kovan;

Ці мережі використовують консенсус Proof of Authority, що робить їх стабільнішими та передбачуванішими. Однак вони не підтримують такі речі, як EIP-1559 або Ethereum 2.0.

* Goerli.

Це тестова мережа, яка намагається поєднати краще з Rinkeby і Kovan, використовуючи консенсус Proof of Authority, але здатна працювати з різними клієнтами Ethereum, включаючи Geth та OpenEthereum.

Усі ці тестові мережі дозволяють розробникам безкоштовно отримувати ether для тестування їхніх застосунків. Це можна зробити через т. зв. крани (faucets), які видають малі кількості ether для тестових мереж.

Важливо пам’ятати, що тестові мережі не мають жодної матеріальної вартості і слугують тільки для тестування. Крім того, вони не є стільки ж безпечними, як основна мережа, тому важливо не використовувати реальні акаунти або приватні ключі в тестових мережах.

# 4 РОЗРОБЛЕННЯ ТА ВПРОВАДЖЕННЯ ДЕЦЕНТРАЛІЗОВАНОГО ВЕБЗАСТОСУНКУ НА ОСНОВІ СМАРТКОНТРАКТУ

## 4.1 Планування проєкту

У сучасному світі, де набули поширення блокчейн-технології та dApps, основою успішного проєкту часто стає вміле поєднання розробки контрактів та створення візуального інтерфейсу, який може ефективно взаємодіяти з ними. Цей розділ буде присвячений процесу створення краудфандингової платформи, що використовує смартконтракт Ethereum для збору та розподілу коштів, а також інтеграції з вебзастосунком на основі бібліотеки React.

React — це сучасна бібліотека JavaScript, яка забезпечує зручний інтерфейс для створення динамічних вебзастосунків. Завдяки своїм можливостям, React дозволяє розробникам створювати великі вебзастосунки, які здатні відображати змінні дані без необхідності перезавантажувати сторінку. Це дуже важливо для децентралізованих застосунків, де потребується постійна взаємодія з блокчейном.

У цьому розділі розглянемо процес створення краудфандингової платформи на основі React, яка буде взаємодіяти з смартконтрактом Ethereum. Розіб’ємо на етапи процес використання Web3.js для взаємодії зі смартконтрактом, а також покажемо, як використовувати React для відображення даних контракту та взаємодії з ним.

Краудфандингова платформа — це відмінний приклад використання блокчейн-технологій, оскільки вона вимагає прозорості, надійності та відмінної взаємодії між користувачами. Платформа, що розробляється, дозволить користувачам створювати кампанії для збору коштів, відстежувати прогрес збору та безпосередньо взаємодіяти з контрактами для внесення внесків або зняття коштів.

Розглянемо, як використовувати бібліотеку Web3.js для взаємодії з Ethereum блокчейном: від створення екземпляру web3 та підключення до провайдера Ethereum, до використання цього екземпляра для відправлення транзакцій та взаємодії з контрактами.

Також подивимося, як можна використовувати React для створення інтерфейсу користувача, який відображає дані з контракту і дозволяє користувачам взаємодіяти з ним. React дозволяє створювати реактивні вебзастосунки, які можуть динамічно оновлювати інтерфейс користувача відповідно до змін у стані застосунка, що робить його ідеальним інструментом для створення децентралізованих вебзастосунків.

Розпочнемо з написання контракту за допомогою Solidity, розгорнемо його на локальному Ethereum вузлі.

Наступним кроком буде налаштування проєкту, включаючи налаштування проєкту React, встановлення Web3.js, та інших необхідних бібліотек, а також налаштування середовища розробки для взаємодії з локальним Ethereum вузлом.

Після цього перейдемо до розробки інтерфейсу користувача за допомогою React. Розглянемо створення компонентів React, які відображають дані з розробленого контракту та дозволяють користувачам вносити внески та створювати нові кампанії для збору коштів.

Наприкінці розгорнемо контракт у тестовій мережі Ethereum та розгорнемо вебзастосунок на хостингу. Обговоримо деякі нюанси щодо впровадження та тестування застосунку, включаючи використання тестових мереж Ethereum, керування витратами газу та оптимізацію роботи застосунку для найкращого користувацького досвіду.

## 4.2 Розроблення смартконтракту

Почнемо з розробки контракту самої платформи та контракту кампанії.

Код, поданий у додатку А та додатку Б, є двома контрактами для реалізації платформи краудфандингу на Ethereum. Контракт CrowdfundingPlatform дозволяє користувачам створювати нові кампанії збору коштів, а контракт Campaign визначає логіку кожної окремої кампанії, включаючи донати, відшкодування донатів і виведення зібраних коштів.

## 4.3 Тестування смартконтракту

Компіляція контракту пройшла успішно, тож наступним кроком спробуємо розгорнути контракт у локальному блокчейні, який пропонує середовище Remix. На рис. 9 можна побачити таке:

1. Підтвердження успішного розгорнення контракту в мережі.
2. Вартість транзакції з розгорнення контракту.
3. Вікно для взаємодії з функціями та змінними контракту.

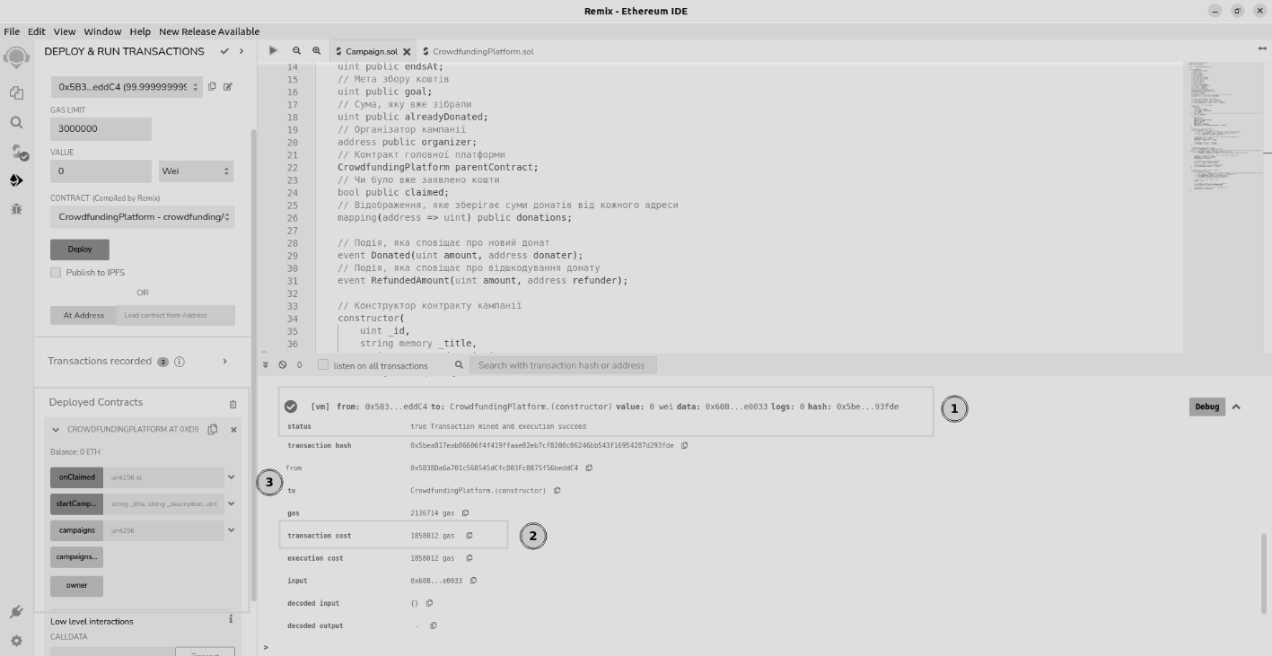


Рисунок 9 — Розгорнення контракту в середовищі Remix

Далі створимо кампанію для збору коштів, вкажемо суму у wei, 1000000000000000000 wei дорівнює 1 ETH, а тривалість кампанії вкажемо 30 днів у секундах від моменту транзакції. На рис. 10 можно побачити таке:

1. Вхідні дані для створення кампанії.
2. Адресу контракту кампанії, розгорнутої у блокчейні.



Рисунок 10 — Створення кампанії

Маючи адресу кампанії, можемо викликати вікно для взаємодії зі змінними та функціями кампанії, та побачити її баланс. Це зображено на рис. 11.

Оскільки вся функціональність працездатна, можна приступати до розробки безпосередньо конфігурації, а потім клієнтської частини застосунку, у якій користувач у зручній формі може взаємодіяти з платформою, яку буде розгорнуто.

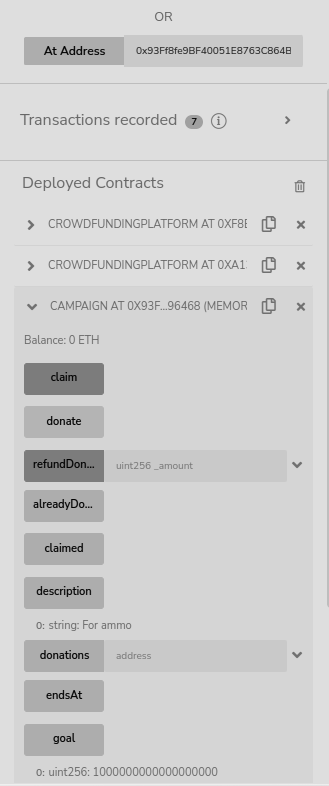


Рисунок 11 — Вікно взаємодії з кампанією

## 4.4 Налаштування конфігурації та структури проєкту

Truffle React Box — це попередньо налаштований проєкт, який включає в себе необхідні конфігурації для швидкого розгортання і розробки децентралізованого застосунку на основі Ethereum. Він включає React для створення інтерфейсу користувача, а також web3.js для взаємодії з Ethereum.

Базова структура проєкту (рис. 12), що надається Truffle React Box, включає:

1. Contracts. Цей каталог містить всі смартконтракти, які використовуються у проєкті.
2. Migrations. Міграції допомагають керувати розгортанням контрактів. Вони зберігаються в каталозі Migrations.
3. Client. Це каталог для клієнтського коду, включаючи React та web3.js.

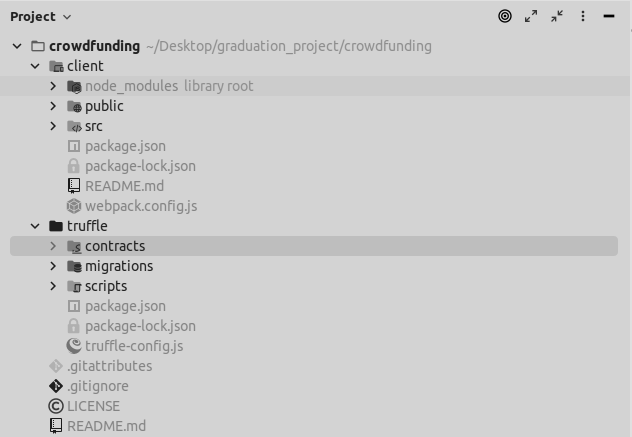


Рисунок 12 — Структура проєкту

Truffle є потужним інструментом для розробки смартконтрактів. Він надає середовище для розробки, тестування і розгортання контрактів, що полегшує цілісний процес розробки. До того ж, використання Truffle React Box дозволяє нам швидко і легко інтегрувати розроблені контракти з клієнтською частиною, що є критично важливим для створення повноцінного децентралізованого застосунку.

Наступними діями треба налаштувати блокчейн для розробки, який буде мати роль бази даних у застосунку. Для початку на етапі розробки будемо використовувати локальний блокчейн, який не підключений до мережі ethereum та у якому будемо тестувати застосунок, для цих цілей було обрано Ganache, інтерфейс можна побачити на рис. 13.

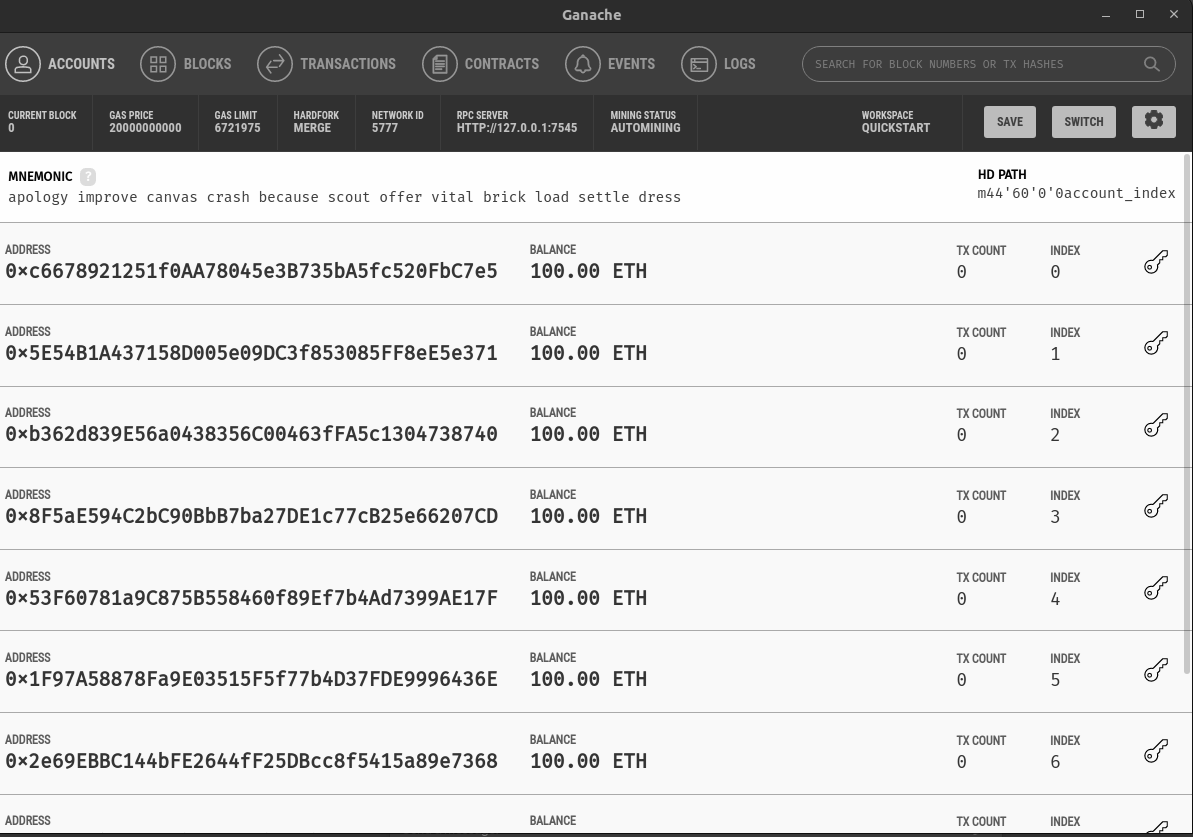


Рисунок 13 — Інтерфейс Ganache

Ganache пропонує зручний інтерфейс та ряд гаманців з балансом за допомогою яких можливо буде робити транзакції та сплачувати за них газ.

Далі у файлі конфігурації truffle вкажемо локальну мережу, запропоновану Ganache, у яку truffle буде розгортати смартконтракти.

Після перенесення коду контрактів до проєкту та встановлення усіх залежностей, налаштування truffle завершено та можна переходити до розробки кліентської частину застосунку.

## 4.5 Розроблення клієнтської частини децентралізованого вебзастосунку

Клієнтська частина є важливою, оскільки вона надає користувачам інтерфейс для взаємодії зі смартконтрактом. Truffle React Box, надає базову структуру для проєкту, у яку вже включено бібліотеку React.

Почнемо з підключення клієнту до Ethereum-вузла в Ganache, але спочатку визначимо декілька допоміжних функцій та винесемо їх в окремий файл “helpers.js” (його наведено в додатку В).

Далі створимо Ethereum-провайдер, який використовується для ініціалізації підключення до Ethereum та керування цим підключенням. Ethereum-провайдер буде основним компонентом, який надає всі необхідні ресурси Ethereum. Це функціональний компонент, яким буде огорнуто увесь застосунок, та всі дочірні компоненти, які матиме розроблений застосунок, зможуть мати доступ до всіх змінних та функцій контракту, код можна побачити у додатку Г.

Для керування станом в усьому застосунку будемо використовувати вбудований у React useReducer. Він альтернативний useState і корисний, коли у вас є складний стан або стан, який включає в себе кілька значень, що мають взаємозв’язок.

useReducer використовує підхід, подібний до того, як працює Redux. Він приймає функцію-редьюсер та початковий стан як вхідні параметри і повертає поточний стан та функцію dispatch. Функція редьюсер приймає поточний стан та дію, а потім повертає новий стан.

У програмному проєкті, що розробляється, useReducer використовується для керування станом Ethereum-провайдера. Цей стан включає в себе різні значення, такі як екземпляр web3, ABI контракту, інформацію про користувацький акаунт, екземпляр контракту CrowdfundingPlatform та список кампаній. Всі ці значення є взаємопов’язаними, тому useReducer виглядає як відмінний вибір для керування цим станом. Він буде зберігатися у файлі state.js, який зображено у додатку Д.

Наступною дією буде обгорнення застосунку Ethereum-провайдером, та додання основних сторінок та логіки переходу на них в залежності від того, яка адреса ресурсу в браузері, це зображено у додатку Е.

Далі необхідно створити візуальну частину застосунку та відобразити на сторінках логіку взаємодії з контрактом, що буде надавати користувачам приємний досвід використання.

Після створення основних компонентів, на які буде поділено застосунок, та їх стилизування можемо приступити до створення компоненту кампанії. Це буде невелика картка на якій буде відображена основна інформація про збір та меню для взаємодії, код цього компоненту зображено у додатку Ж.

Такі компоненти як ProgressBar або DonateButton, було створено щоб відокремити складну логіку. Розбивання сторінок на компоненти в сучасних вебзастосунках є важливою практикою, це допомагає частіше повторно використовувати елементи у застосунках та запобігати дублювання. Також це важливо для зручності читання коду.

Далі створимо компонент, який циклом буде проходити по масиву кампаній, на кожний елемент масиву створювати окремий компонент CampaignElement та передавати в нього дані кампанії. Код цього компоненту наведено у додатку И.

Тепер так само напишемо компонент для створення кампанії. Використаємо бібліотеку Formik для зручної взаємодії з формою, яка нам знадобиться. Код компонента наведено у додатку К.

Перенесемо створені компоненти до сторінок та запустимо програмний проєкт. Після запуску локального серверу React побачимо інтерфейс застосунку, та перейдемо до сторінки створення кампанії як це зображено на рис. 14.

Створимо кампанію та сплатимо газ за допомогою гаманця Metamask підключеного до Ganache, як це наведено на рис. 15.

Перейдемо на головну сторінку та побачимо новостворену кампанію (рис. 16).

Переведемо кошти у кампанію тим же чином, вказавши суму ETH та підтвердивши транзакцію, як це зображено на рис. 17.

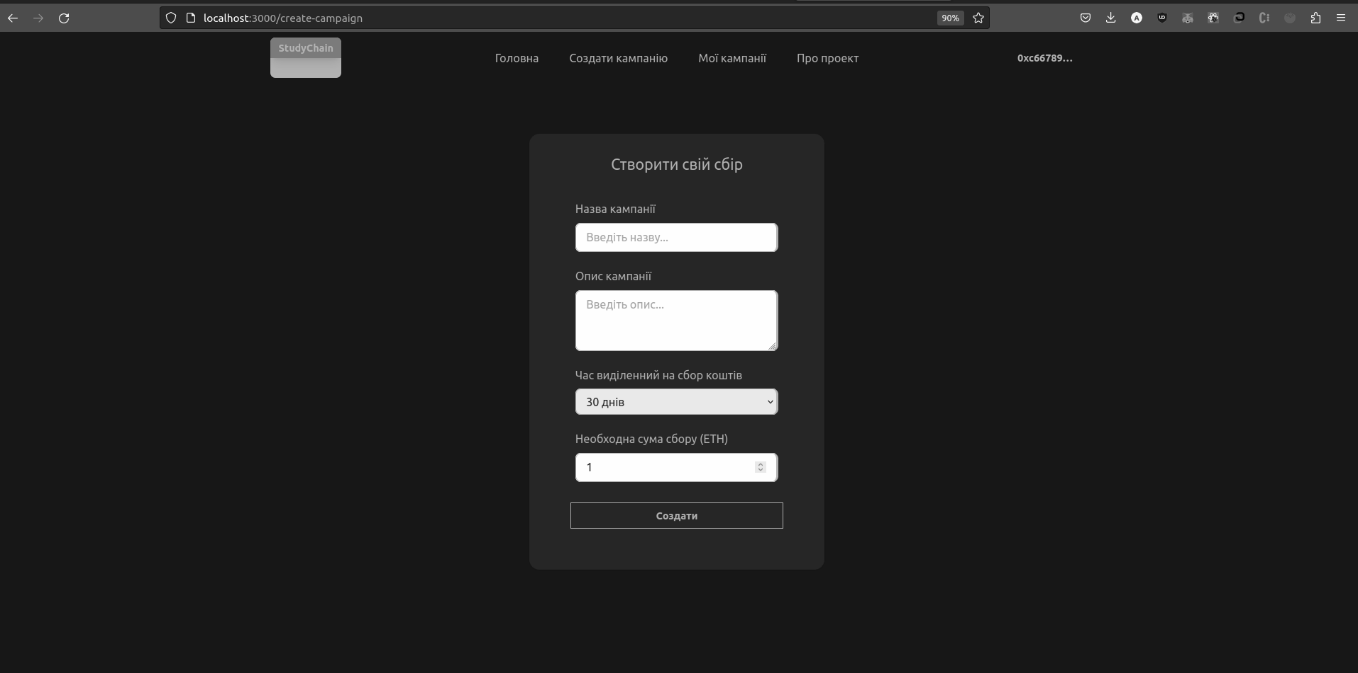


Рисунок 14 — Сторінка створення кампанії

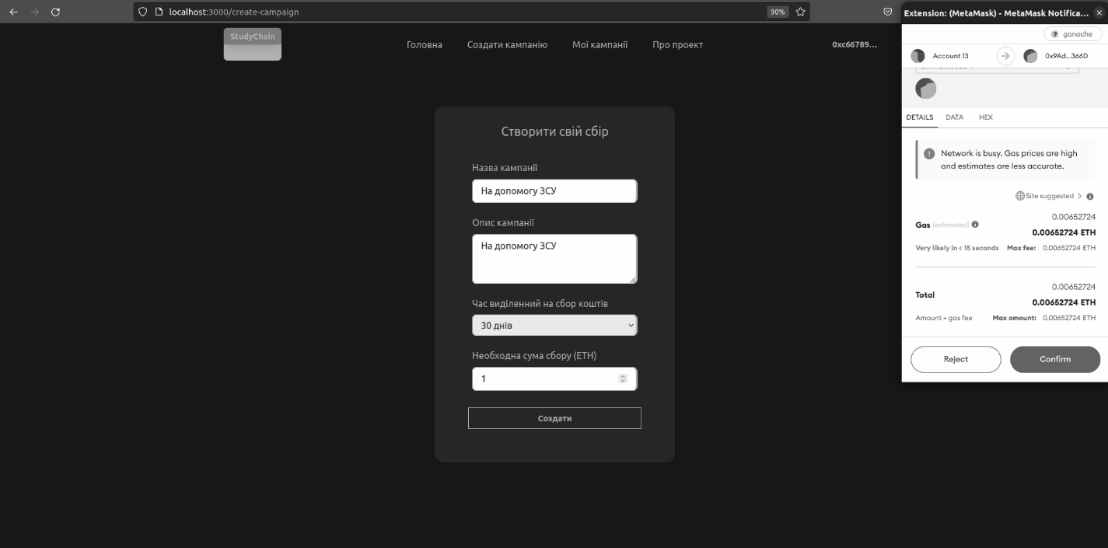


Рисунок 15 — Підтвердження транзакції в Metamask

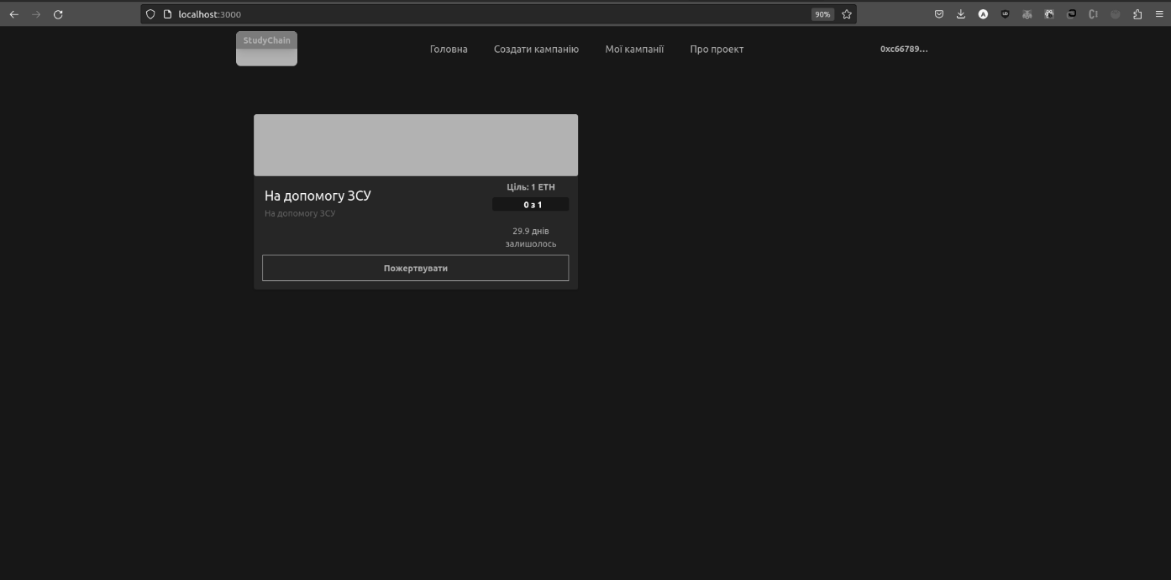


Рисунок 16 — Головна сторінка з новоствореною кампанією

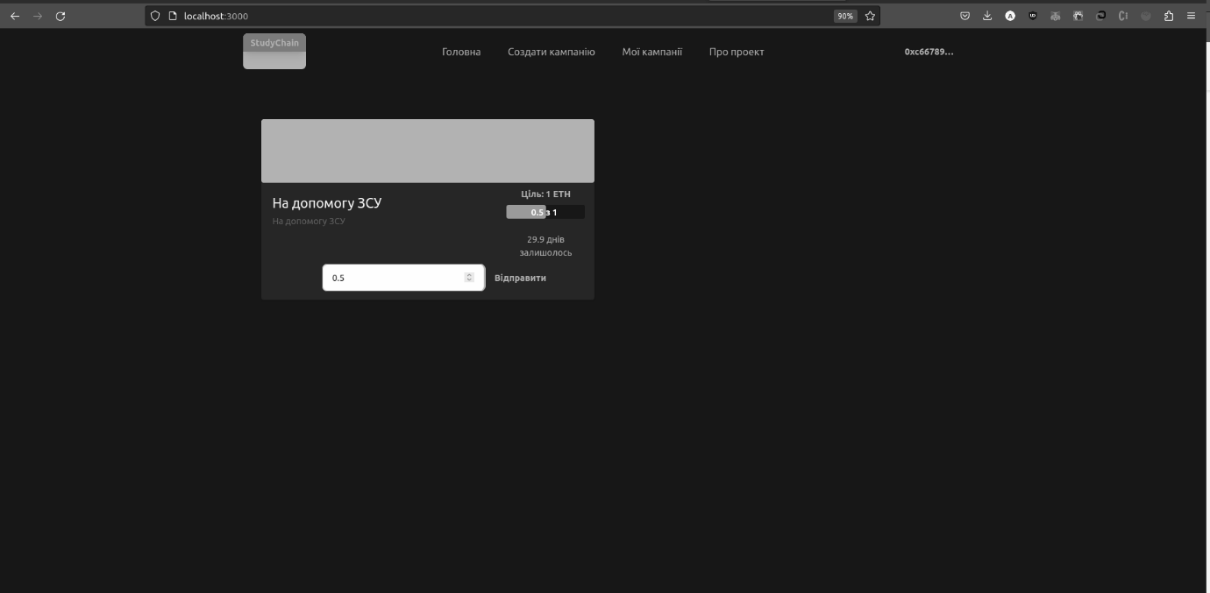


Рисунок 17 — Кампанія з балансом

## 4.6 Розгорнення смартконтракту та вебзастосунку

Розгортання смартконтракту на Ethereum включає кілька етапів. Спочатку потрібно створити новий проєкт на Infura. Infura — це API, що дає можливість взаємодіяти з Ethereum-мережею без необхідності самостійно підтримувати повноцінний Ethereum-вузол.

Після створення проєкту на Infura, потрібно встановити в своєму проєкті бібліотеку “@truffle/hdwallet-provider”. Ця бібліотека дозволяє використовувати приватні ключі (або mnemonic — набір з 12 слів, які генеруються гаманцем) для створення ether-адреси, з якої буде проведено транзакцію.

Знаючи mnemonic і адресу проєкту в Infura, можна налаштувати Truffle для відправки транзакцій в мережу Ethereum. Всі ці параметри вказуються в файлі налаштувань Truffle (“truffle-config.js”), приклад зображено на рис. 18.

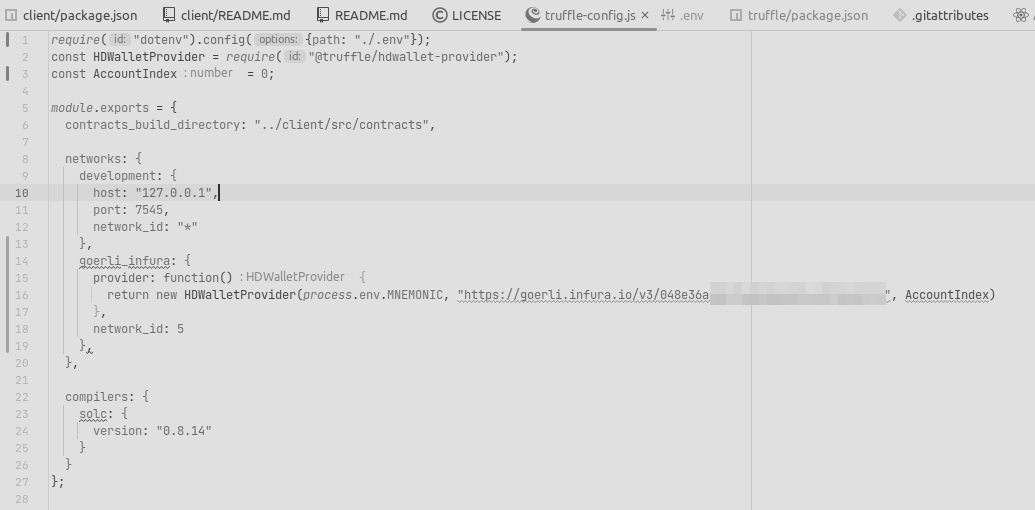


Рисунок 18 — Уміст файлу “truffle-config.js”

Тепер можна розгорнути свій контракт в мережу goerli, виконавши команду “truffle migrate --network goerli”, результат цієї команди зображено на рис. 19.

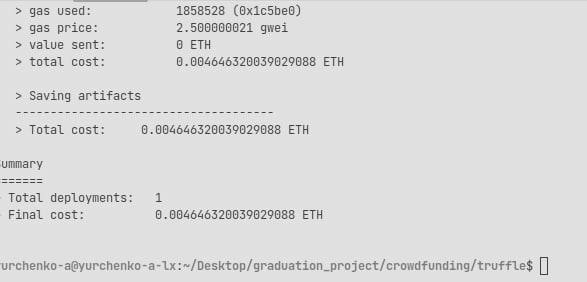


Рисунок 19 — Успішне розгорнення контракту

Тестування і розгортання смартконтрактів в реальному блокчейні відрізняється від локального середовища: транзакції не є миттєвими і потребують певного часу для підтвердження, і вони вимагають витрати газу (витрати ETH).

Стосовно розгортання вебзастосунку, Vercel — це хмарна платформа, що дозволяє легко розгортати вебзастосунки прямо з Git. Це надзвичайно корисно для веброзробників, оскільки вони можуть просто пушити свої зміни в репозиторій і Vercel автоматично створити нову версію вебзастосунку, успішне розгорнення застосунку з платформи Github зображено на рис. 20.

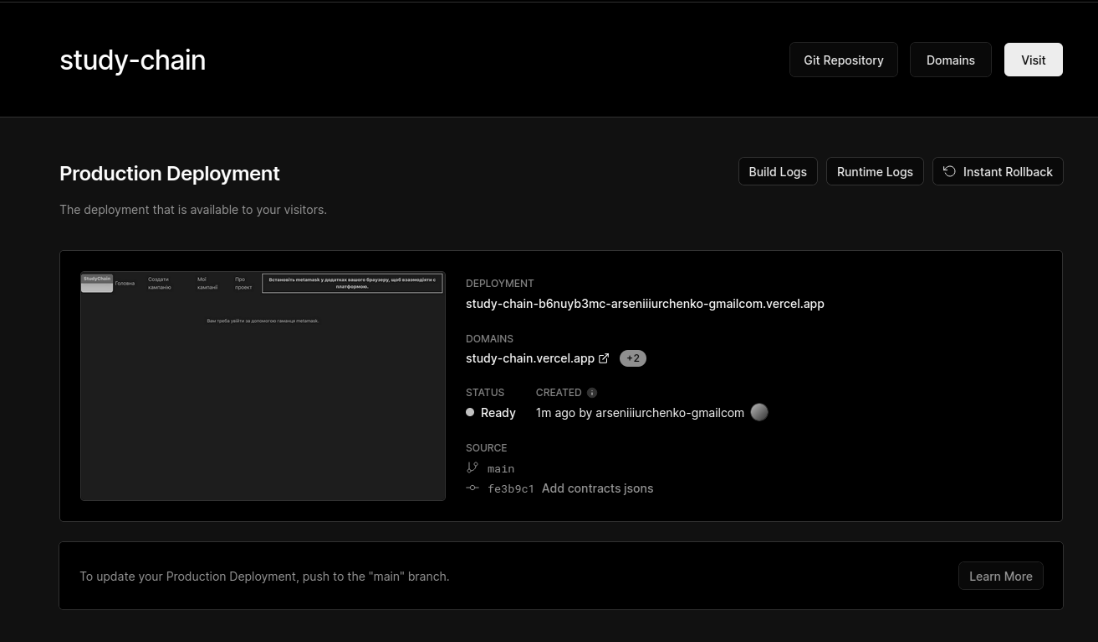


Рисунок 20 — Успішне розгорнення React-застосунку

Тепер застосунок успішно розгорнуто у мережі Інтернет та будь-який користувач, який має гаманець Metamask, може ознайомитись з можливостями застосунку у тестовому режимі за посиланням “https://study-chain.vercel.app/”.

# 5 ПРОБЛЕМИ БЕЗПЕКИ, ЯКІ ВИНИКАЮТЬ У РОЗРОБКАХ СМАРТКОНТРАКТІВ

## 5.1 Загальні проблеми безпеки смартконтрактів

Смартконтракти в Ethereum засновані на технології blockchain, яка відома своєю прозорістю, відкритістю, незмінністю та децентралізацією. Однак ці самі властивості, які роблять blockchain та смартконтракти привабливими для розробників і користувачів, також створюють унікальні проблеми безпеки.

Відкритість коду. Код смартконтрактів є повністю відкритим і доступним для огляду. Це означає, що потенційні зловмисники можуть аналізувати код в пошуках вразливостей, які можуть бути використані для зловмисних дій.

Незмінність коду. Після того, як смартконтракт було розгорнуто в Ethereum, його код стає незмінним. Це означає, що якщо в коді контракту виявлено помилку або вразливість, її не можна виправити шляхом оновлення коду. Це робить незмінність коду смартконтрактів одним з найбільших викликів у сфері безпеки.

Складність коду. Смартконтракти в Ethereum пишуться на мові Solidity, яка має свої власні особливості та складності. Деякі з цих особливостей можуть привести до непередбачуваного поведінки контрактів, якщо розробники не розуміють повністю, як вони працюють. Наприклад, переповнення чисел або необережне використання викликів делегатів може призвести до серйозних проблем безпеки.

Відсутність централізованої організації. Оскільки Ethereum є децентралізованою платформою, немає одного органу, що контролює всі транзакції або смартконтракти. Це означає, що якщо стається щось підозріле або неправильне, немає централізованого органу, який може негайно втрутитись.

Відсутність типової моделі користувача. Ethereum не має типової моделі користувача, і користувачі відповідають за керування своїми власними ключами доступу. Якщо користувач втратить свій приватний ключ, він може втратити доступ до своїх коштів, і немає способу відновити ці кошти.

## 5.2 Типові атаки на смартконтракти та способи запобігання їм

Атаки переповнення/недоповнення (overflow/underflow attacks) виникають, коли числова змінна в Solidity досягає своєї максимальної або мінімальної границі і перевертається. Це може призвести до непередбачуваних результатів, які можуть бути використані зловмисниками. Розроблені в цьому проєкті контракти безпечні від цього типу атак, оскільки вони використовують версію Solidity 0.8.x, яка автоматично обробляє ці ситуації з переповненням та недоповненням, викидаючи помилку при їх виникненні.

Reentrancy attacks відбуваються, коли зовнішній контракт має можливість викликати смартконтракт ще раз до завершення початкового виклику. Це може дозволити зловмиснику вивести більше коштів, ніж він має. Розроблені в цьому проєкті смартконтракти захищені від цього типу атак за допомогою використання модифікаторів стану, як наприклад require в методах donate, refundDonation та claim, які переконуються, що певні умови виконуються, перш ніж дозволяти виконання дій.

Атаки на відкладені виклики (race condition attacks) відбуваються, коли зловмисник може подивитися в стан блоку Ethereum та змінити свої транзакції відповідно до цього. Розроблені в цьому проєкті смартконтракти захищають від цього типу атак, використовуючи умови require для перевірки стану перед виконанням важливих дій. Наприклад, метод donate перевіряє, чи не закінчився термін дії кампанії, та чи внесена сума не є нульовою.

Атаки типу phishing/scamming не є прямою вразливістю смартконтракту, але все ж є загрозою для користувачів. Зловмисники створюють фальшиві контракти або сайти, які виглядають як оригінальні, і спонукають користувачів відправити їм Ether або токени. Щоб боротися з цим, важливо забезпечити прозорість та довіру до проєкту. Смартконтракти можуть бути перевірені та публічно доступні для огляду, щоб користувачі могли бути впевнені в їх автентичності.

Під час front-running attacks зловмисник відстежує незахищені транзакції в мережі, які ще не були включені в блок, і подає свою власну транзакцію з більшою кількістю газу, щоб її було виконано раніше за інші. Розроблені в цьому проєкти контракти захищені від цього типу атаки, тому що вони використовують функцію require для перевірки умов перед виконанням важливих дій. Це означає, що, незалежно від порядку транзакцій, якщо умови не виконуються, транзакції будуть відхилені.

Вище було наведено лише кілька прикладів атак, які можуть бути спрямовані на смартконтракти. Важливо зазначити, що захист від цих атак вимагає ретельної уваги до деталей при розробці смартконтракту та проведення аудиту безпеки перед його розгортанням.

## 5.3 Приклади значущих інцидентів із безпекою смартконтрактів у системі Ethereum

DAO Hack (2016 рік). DAO (Decentralized Autonomous Organization) була одним з перших значних проєктів на платформі Ethereum. В результаті його ICO було зібрано еквівалент 150 мільйонів доларів США в ETH. Однак через помилку в його смартконтракті (рекурсивний виклик вразливості), хакер вкрав близько 50 мільйонів доларів. Ця подія спричинила т. зв. hardfork мережі Ethereum і створення Ethereum Classic (ETC).

Parity Multisig Wallet Hack (2017 рік). Це одна з найбільших крадіжок в історії Ethereum. Атакуючий використав вразливість в мультипідписному гаманці (multisignature wallet) Parity та вкрав близько 30 мільйонів доларів в ETH. Вразливість полягала в помилці в бібліотеці смартконтрактів, яка дозволила атакуючому взяти під свій контроль випадкові гаманці.

Parity Multisig Wallet Freeze (2017 рік). Той же мультипідписний гаманець Parity, який був зламаний в липні 2017 року, зазнав ще одного значного інциденту в листопаді того ж року. На цей раз користувач випадково викликав функцію самознищення (self-destruct) для бібліотеки смартконтрактів, що контролювала всі мультипідписні гаманці Parity. Це призвело до заморожування близько 280 мільйонів доларів в ETH, які залишаються недоступними й до сьогодні.

Інцидент DeFi Pickle Finance (2020 рік). Pickle Finance, популярний проєкт в сфері децентралізованих фінансів (DeFi), втратив близько 20 мільйонів доларів через помилку в одному з їхніх смартконтрактів. Хакер використав помилку в системі, що контролює jars (аналоги вкладів), щоб перерозподілити кошти на свій контрольований адрес.

Ці інциденти підтверджують важливість ретельного аудиту та тестування смартконтрактів перед їх запуском в основну мережу. Вони також показують необхідність правильного використання вже випробуваних бібліотек та паттернів, що зменшують ризик виникнення вразливостей.

# ВИСНОВКИ

Під час виконання цього проєкту було успішно розроблено, протестовано, впроваджено та проаналізовано з погляду безпеки децентралізований застосунок, побудований на основі блокчейна Ethereum та смартконтрактів.

Проєкт було розпочато з розробки двох ключових смартконтрактів — Campaign та CrowdfundingPlatform, використовуючи мову програмування Solidity. Ці контракти були спроєктовані таким чином, щоб відповідати основним потребам системи краудфандингу: створення кампаній та керування ними.

Наступним кроком було створення вебзастосунку на базі React.js, що надає користувачам графічний інтерфейс для взаємодії з контрактами. Для забезпечення цього взаємодії було використано бібліотеку web3.js.

Крім розроблення цих компонентів, було розгорнуто смартконтракти в тестовій мережі Ethereum — Goerli, а вебзастосунок — на сервісі Vercel. Це важливий етап, що демонструє готовність системи до реального використання.

Крім того, важливою частиною роботи було дослідження можливих проблем безпеки та ризиків, пов’язаних з розробленням та використанням смартконтрактів. Було розглянуто загальні проблеми безпеки смартконтрактів, описали типові атаки та способи їх запобігання. Також було проаналізовано розроблені контракти щодо можливості таких атак і було зазначено способи запобігання потенційним проблемам.

На завершення проєкту можна ствердити, що план було успішно виконано, дослідивши та застосувавши ключові технології та підходи, які використовуються в сучасних блокчейн-проєктах, та дослідивши важливі аспекти безпеки.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. URL: https://bitcoin.org/bitcoin.pdf (дата доступу — 08.06.2023).
2. Buterin V. Ethereum White Paper: A Next-Generation Smart Contract and Decentralized Application Platform. 2014. URL: https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum\_Whitepaper\_-\_Buterin\_2014.pdf (дата доступу — 08.06.2023).
3. Mougayar W. The Business Blockchain: Promise, Practice, and Application of the Next Internet Technology. Wiley, 2016. 208 с.
4. Wood G. Ethereum: A Secure Decentralised Generalised Transaction Ledger. Berlin version. 2022. URL: https://ethereum.github.io/yellowpaper/paper.pdf (дата доступу — 08.06.2023).
5. Wright A., De Filippi P. Blockchain and the Law: The Rule of Code. Harvard University Press, 2018. 312 с.
6. FinCEN. Application of FinCEN's Regulations to Persons Administering, Exchanging, or Using Virtual Currencies. 2013. URL: https://www.fincen.gov/resources/statutes-regulations/guidance/application-fincens-regulations-persons-administering (дата доступу — 08.06.2023).
7. European Commission. European Blockchain Partnership. 2018. URL: https://digital-strategy.ec.europa.eu/en/policies/blockchain-partnership (дата доступу — 08.06.2023).
8. Finck M. Blockchain Regulation and Governance in Europe. Cambridge University Press, 2018. 214 с.
9. The Solidity Authors. Solidity 0.8.20 documentation. 2023. URL: https://docs.soliditylang.org/en/v0.8.20/ (дата доступу — 08.06.2023).
10. Antonopoulos A., Wood G. Mastering Ethereum: Building Smart Contracts and Dapps. O’Reilly Media, 2018. 424 с.

# ДОДАТОК А

Лістинг коду контракту платформи.

//SPDX-License-Identifier: MIT  
pragma solidity 0.8.14;  
  
import "./Campaign.sol";  
  
contract CrowdfundingPlatform {  
 // Структура кампанії збору коштів, яка зберігає контракт кампанії та інформацію про те, чи були вже заявлені кошти  
 struct CrowdfundingCampaign {  
 Campaign targetContract;  
 bool claimed;  
 }  
  
 // Відображення, яке зберігає всі кампанії  
 mapping(uint => CrowdfundingCampaign) public campaigns;  
 // Кількість кампаній  
 uint public campaignsCount;  
 // Власник платформи  
 address public owner;  
  
 // Подія, яка сповіщає про старт нової кампанії  
 event CampaignStarted(uint id, Campaign newCampaignAddress, uint endsAt, uint goal, address organizer);

// Функція для створення нової кампанії  
 function startCampaign(  
 string memory \_title,  
 string memory \_description,  
 uint \_goal,  
 uint \_endsAt  
 ) external {  
 // Перевіряємо, що дата завершення в майбутньому і мета кампанії більша за 0  
 require(\_endsAt > block.timestamp, "Campaign duration can't be earlier than now.");  
 require(\_goal > 0, "You must define goal of your campaign.");  
  
 // Збільшуємо кількість кампаній  
 campaignsCount = campaignsCount + 1;  
  
 // Створюємо нову кампанію  
 Campaign newCampaign = new Campaign(campaignsCount, \_title, \_description, \_endsAt, \_goal, msg.sender);  
  
 // Додаємо кампанію до нашого відображення  
 campaigns[campaignsCount] = CrowdfundingCampaign({  
 targetContract: newCampaign,  
 claimed: false  
 });  
  
 // Викликаємо подію про старт кампанії  
 emit CampaignStarted(campaignsCount, newCampaign, \_endsAt, \_goal, msg.sender);  
 }  
  
 // Функція для відзначення, що кошти були заявлені  
 function onClaimed(uint id) external {  
 CrowdfundingCampaign storage targetCampaign = campaigns[id];  
  
 // Перевіряємо, що ця функція була викликана контрактом цільової кампанії  
 require(msg.sender == address(targetCampaign.targetContract), "It is not required campaign.");  
  
 // Встановлюємо статус claimed у true  
 targetCampaign.claimed = true;  
 }  
}

# 

# ДОДАТОК Б

Лістинг коду контракту кампанії.

//SPDX-License-Identifier: MIT  
pragma solidity 0.8.14;  
  
import "./CrowdfundingPlatform.sol";  
  
contract Campaign {  
 // Унікальний ID кампанії  
 uint public id;  
 // Назва кампанії  
 string public title;  
 // Опис кампанії  
 string public description;  
 // Час закінчення кампанії  
 uint public endsAt;  
 // Мета збору коштів  
 uint public goal;  
 // Сума, яку вже зібрали  
 uint public alreadyDonated;  
 // Організатор кампанії  
 address public organizer;  
 // Контракт головної платформи  
 CrowdfundingPlatform parentContract;  
 // Чи було вже заявлено кошти  
 bool public claimed;  
 // Відображення, яке зберігає суми донатів від кожного адреси  
 mapping(address => uint) public donations;  
  
 // Подія, яка сповіщає про новий донат  
 event Donated(uint amount, address donater);  
 // Подія, яка сповіщає про відшкодування донату  
 event RefundedAmount(uint amount, address refunder);  
  
 // Конструктор контракту кампанії  
 constructor(  
 uint \_id,  
 string memory \_title,  
 string memory \_description,  
 uint \_endsAt,  
 uint \_goal,  
 address \_organizer  
 ) {  
 id = \_id;  
 title = \_title;  
 description = \_description;  
 endsAt = \_endsAt;  
 goal = \_goal;  
 organizer = \_organizer;  
 parentContract = CrowdfundingPlatform(msg.sender);  
 }  
  
 // Функція для здійснення донату  
 function donate() external payable {  
 // Перевіряємо, що кампанія ще не закінчилась і сума донату більша за 0  
 require(block.timestamp <= endsAt, "Campaign already has over.");  
 require(msg.value > 0, "You can't donate 0.");  
  
 // Збільшуємо суму вже зібраних коштів і суму донату від цієї адреси  
 alreadyDonated += msg.value;  
 donations[msg.sender] += msg.value;  
  
 // Викликаємо подію про новий донат  
 emit Donated(msg.value, msg.sender);  
 }  
  
 // Функція для відшкодування донату  
 function refundDonation(uint \_amount) external {  
// Перевіряємо, що кампанія ще не закінчилась і сума для відшкодування не більша, ніж сума донату від цієї адреси  
 require(block.timestamp <= endsAt, "Campaign already has over.");  
 require(\_amount <= donations[msg.sender], "you cant refund more than you gave.");  
  
 // Зменшуємо суму вже зібраних коштів та суму донату від цієї адреси  
 donations[msg.sender] -= \_amount;  
 alreadyDonated -= \_amount;  
  
 // Переказуємо кошти назад на адресу донора  
 payable(msg.sender).transfer(\_amount);  
  
 // Викликаємо подію про відшкодування донату  
 emit RefundedAmount(\_amount, msg.sender);  
 }  
  
 // Функція для заявлення коштів організатором  
 function claim() external {  
 // Перевіряємо, що ця функція була викликана організатором, сума зібраних коштів достатня і кошти ще не були заявлені  
 require(msg.sender == organizer, "You are not the organizer.");  
 require(alreadyDonated >= goal, "Not enough funds.");  
 require(!claimed, "Campaign already has claimed.");  
  
 // Встановлюємо статус claimed у true  
 claimed = true;  
  
 // Переказуємо зібрані кошти на адресу організатора  
 payable(organizer).transfer(alreadyDonated);  
  
 // Сповіщаємо головний контракт про заявлення коштів  
 parentContract.onClaimed(id);  
 }  
}

# 

# ДОДАТОК В

Лістинг коду допоміжних функцій (“helpers.js”).

// Функція для отримання адрес кампаній  
export const getCampaignsAddresses = async (contractInstance, campaignsCount) => {  
 const campaignsAddressesPending = [];  
  
 // Проходимо через всі кампанії в контракті  
 for (let i = 1; i <= campaignsCount; i++) {  
 // Отримуємо об'єкт запросу для кожної кампанії, що поверне її адресу  
 campaignsAddressesPending.push(contractInstance  
 .methods.campaigns(i).call());  
 }  
  
 // Виконуємо всі об'єкти запросу паралельно  
 const campaignsAddresses = await *Promise*.all(campaignsAddressesPending);  
  
 // Повертаємо масив адрес кампаній  
 return campaignsAddresses;  
};  
  
// Функція для створення екземпляру кампанії з вказаної адреси  
export const filterCampaignInstance = async (web3, abi, contractAddress, userAccount) => {  
 // Створюємо новий контракт на основі ABI та адреси  
 const { methods: campaignMethods } = new web3  
 .eth.Contract(abi, contractAddress);  
  
 // Зчитуємо дані кампанії з блокчейну та повертаємо їх  
 return {  
 id: +await campaignMethods.id().call(),  
 title: await campaignMethods.title().call(),  
 description: await campaignMethods.description().call(),  
 goal: +web3.utils.fromWei(await campaignMethods.goal().call(), 'ether'),  
 alreadyDonated: +web3.utils.fromWei(await campaignMethods.alreadyDonated().call(), 'ether'),  
 endsAt: +await campaignMethods.endsAt().call(),  
 organizer: await campaignMethods.organizer().call(),  
 claimed: await campaignMethods.claimed().call(),  
 currentUserDonations: +web3.utils.fromWei(await campaignMethods.donations(userAccount).call(), 'ether'),  
 donate: await campaignMethods.donate,  
 refundDonation: await campaignMethods.refundDonation,  
 claim: await campaignMethods.claim,  
 };  
};

# 

# ДОДАТОК Г

Лістинг коду Ethereum-провайдера.

import React, {  
 useReducer, useCallback, useEffect,  
} from 'react';  
import Web3 from 'web3';  
  
import *EthContext* from './EthContext'; // Контекст, який ми будемо використовувати для передачі даних про Ethereum  
import { reducer, *actions*, *initialState* } from './state'; // Наш ред'юсер та відповідні дії  
import {  
 filterCampaignInstance,  
 getCampaignsAddresses,  
} from './helpers/helpers'; // Допоміжні функції  
  
// Наши контракти для взаємодії з Ethereum  
const CrowdfundingPlatform = *require*('../../contracts/CrowdfundingPlatform.json');  
const CampaignContract = *require*('../../contracts/Campaign.json');  
  
// Наш Ethereum провайдер  
function EthProvider({ children }) {  
 const [state, dispatch] = useReducer(reducer, *initialState*); // Використовуємо useReducer для управління станом  
  
 const init = useCallback(async () => { // Ініціалізація Ethereum  
 if (CrowdfundingPlatform && CampaignContract) {  
 // Створюємо новий екземпляр Web3  
 const web3 = new Web3(Web3.*givenProvider* || 'http://localhost:7545');  
  
 const accounts = await web3.eth.requestAccounts(); // Запитуємо доступ до облікових записів  
 const networkID = await web3.eth.net.getId(); // Отримуємо ID мережі  
  
 // Витягуємо abi наших контрактів  
 const { abi } = CrowdfundingPlatform;  
 const { abi: campaignAbi } = CampaignContract;  
  
 // Отримуємо адресу нашої CrowdfundingPlatform в мережі  
 const { address } = CrowdfundingPlatform.networks[networkID];  
  
 // Створюємо екземпляр контракту CrowdfundingPlatform  
 const crowdfundingPlatformInstance = new web3.eth.Contract(abi, address);  
  
 // Зчитуємо кількість кампаній за допомогою змінних у контракті  
 const campaignsCount = await crowdfundingPlatformInstance  
 .methods.campaignsCount().call();  
  
 // Завантажуємо всі кампанії  
 const campaigns = await (async function () {  
 if (!campaignsCount) {  
 return [];  
 }  
  
 // Отримуємо адреси всіх кампаній  
 const campaignsAddresses = await getCampaignsAddresses(  
 crowdfundingPlatformInstance,  
 campaignsCount,  
 );  
  
 // Фільтруємо кампанії  
 const campaignsPending = campaignsAddresses  
 .map((campaign) => filterCampaignInstance(  
 web3,  
 campaignAbi,  
 campaign.targetContract,  
 accounts[0],  
 ));  
  
 // Очікуємо завершення всіх асінхронних запитів  
 const campaignsPromisses = await *Promise*.all(campaignsPending);  
  
 return campaignsPromisses;  
 }());  
  
 // Відправляємо дію для ініціалізації  
 dispatch({  
 type: *actions*.init,  
 data: {  
 web3,  
 campaignAbi,  
 isLoading: false,  
 userAccount: accounts[0],  
 crowdfundingPlatformInstance,  
 campaigns,  
 },  
 });  
 }  
 }, []);  
  
 // Перевірка наявності Metamask при монтуванні компонента  
 useEffect(() => {  
 dispatch({  
 type: *actions*.init,  
 data: { isMetamaskInstalled: !!*window*.ethereum },  
 });  
 }, []);  
  
 // Спостерігаємо за змінами в блокчейні або аккаунтах  
 useEffect(() => {  
 if (*window*.ethereum) {  
 isConnected();  
 const events = ['chainChanged', 'accountsChanged'];  
 const handleChange = () => {  
 init();  
 };  
  
 // Додаємо слухачів подій  
 events.forEach((e) => *window*.ethereum.on(e, handleChange));  
  
 // При розмонтовуванні компонента видаляємо слухачів  
 return () => {  
 events.forEach((e) => *window*.ethereum.removeListener(e, handleChange));  
 };  
 }  
 }, [init, CrowdfundingPlatform]);  
  
 // Перевірка на підключеність до Ethereum  
 const isConnected = async () => {  
 const accounts = await *window*.ethereum.request({ method: 'eth\_accounts' });  
 if (accounts.length) {  
 init();  
 }  
 };  
  
 // Функція для підключення гаманця  
 const connectWallet = () => {  
 init();  
 };  
  
 // Повертаємо провайдер EthContext  
 return (  
 <EthContext.Provider value={{  
 state,  
 dispatch,  
 connectWallet,  
 }}  
 >  
 {children}  
 </EthContext.Provider>  
 );  
}  
  
export default EthProvider;

# 

# ДОДАТОК Д

Лістинг коду логіки керування станом Ethereum-провайдера.

const *actions* = {  
 init: 'INIT',  
 addCampaign: 'ADD\_CAMPAIGN',  
 donateFunds: 'DONATE\_FUNDS',  
 refundFunds: 'REFUND\_FUNDS',  
 claimFunds: 'CLAIM\_FUNDS',  
};  
  
const *initialState* = {  
 web3: null,  
 campaignAbi: null,  
 isLoading: true,  
 isMetamaskInstalled: false,  
 userAccount: null,  
 crowdfundingPlatformInstance: null,  
 campaigns: [],  
};  
  
const reducer = (state, action) => {  
 const { type, data } = action;  
 switch (type) {  
 case *actions*.init:  
 return { ...state, ...data };  
 case *actions*.donateFunds:  
 return {  
 ...state,  
 campaigns: state.campaigns.map((campaign) => (campaign.id === data.id  
 ? {  
 ...campaign,  
 alreadyDonated: campaign.alreadyDonated + data.donatedAmount,  
 currentUserDonations: campaign.currentUserDonations + data.donatedAmount,  
 }  
 : campaign)),  
 };  
 case *actions*.refundFunds:  
 return {  
 ...state,  
 campaigns: state.campaigns.map((campaign) => (campaign.id === data.id  
 ? {  
 ...campaign,  
 alreadyDonated: campaign.alreadyDonated - data.currentUserDonations,  
 currentUserDonations: campaign.currentUserDonations - data.currentUserDonations,  
 }  
 : campaign)),  
 };  
 case *actions*.claimFunds:  
 return {  
 ...state,  
 campaigns: state.campaigns.map((campaign) => (campaign.id === data.id  
 ? {  
 ...campaign,  
 claimed: true,  
 }  
 : campaign)),  
 };  
 case *actions*.addCampaign:  
 return { ...state, campaigns: [...state.campaigns, data] };  
 default:  
 throw new *Error*('Undefined reducer action type');  
 }  
};  
  
export {  
 *actions*,  
 *initialState*,  
 reducer,  
};

# 

# ДОДАТОК Е

Лістинг коду кореневого компоненту “App.tsx”.

import { FC } from 'react';  
import {  
 BrowserRouter,  
 Routes,  
 Route,  
} from 'react-router-dom'; // імпортуємо компоненти для маршрутизації з react-router-dom  
import { HomePage } from '../../pages/HomePage/HomePage'; // імпортуємо компонент HomePage  
import { EthProvider } from '../../contexts/EthContext'; // імпортуємо провайдер контексту Ethereum  
import { CreateCampaignPage } from '../../pages/CreateCampaignPage/CreateCampaignPage'; // імпортуємо компонент CreateCampaignPage  
import { AboutPage } from '../../pages/AboutPage/AboutPage'; // імпортуємо компонент AboutPage  
import { CampaignPage } from '../../pages/CampaignPage/CampaignPage'; // імпортуємо компонент CampaignPage  
import { MyCampaignsPage } from '../../pages/MyCampaignsPage/MyCampaignsPage'; // імпортуємо компонент MyCampaignsPage  
  
// Основний компонент додатку  
export const App: FC = () => (  
 // Обгортка з контекстом Ethereum  
 <EthProvider>  
 {/\* Використовуємо BrowserRouter для маршрутизації \*/}  
 <BrowserRouter>  
 {/\* Визначаємо маршрути для нашого додатку \*/}  
 <Routes>  
 {/\* Домашня сторінка \*/}  
 <Route path="/" element={<HomePage />} />  
 {/\* Сторінка окремої кампанії, де :id - це параметр, що містить ідентифікатор кампанії \*/}  
 <Route path="/campaign/:id" element={<CampaignPage />} />  
 {/\* Сторінка для створення кампанії \*/}  
 <Route path="/create-campaign" element={<CreateCampaignPage />} />  
 {/\* Сторінка зі списком кампаній користувача \*/}  
 <Route path="/my-campaigns" element={<MyCampaignsPage />} />  
 {/\* Сторінка з інформацією про додаток \*/}  
 <Route path="/about" element={<AboutPage />} />  
 </Routes>  
 </BrowserRouter>  
 </EthProvider>  
);

# 

# ДОДАТОК Ж

Лістинг коду компоненту (картки) кампанії.

import React, { FC } from 'react';  
import { useNavigate } from 'react-router-dom'; // Hook для навігації між сторінками  
  
import { ProgressBar } from '../../ProgressBar/ProgressBar'; // Компонент для відображення прогресу збору коштів  
import { convertSecondsToDays } from '../../../utils/utils'; // Функція для конвертації секунд у дні  
import { DonateButton } from '../../DonateButton/DonateButton'; // Кнопка для внесення внеску  
import './campaignElement.scss'; // Стилі для компонента  
  
// Опис пропсів для компонента  
interface CampaignElementProps {  
 id: number;  
 title: string;  
 description: string;  
 goal: number;  
 alreadyDonated: number;  
 claimed: boolean;  
 endsAt: number;  
 donate: () => any;  
}  
  
// Головний компонент  
export const CampaignElement: FC<CampaignElementProps> = ({  
 id,  
 title,  
 description,  
 goal,  
 claimed,  
 alreadyDonated,  
 endsAt,  
 donate,  
}) => {  
 const navigate = useNavigate(); // Використовуємо Hook для навігації  
  
 // Функція для переходу на сторінку кампанії  
 const navigateToCampaignPage = () => navigate(`/campaign/${id}`);  
  
 // Рендеримо компонент  
 return (  
 <div  
 onClick={navigateToCampaignPage} // При кліку на компонент переходимо на сторінку кампанії  
 className="campaigns-element-item"  
 >  
 <div className="campaigns-element-item-image" />  
 <div className="campaigns-element-item-content">  
 <div className="campaigns-element-item-content-text">  
 <h2 className="campaigns-element-item-content-text-title">{title}</h2>  
 <p className="campaigns-element-item-content-text-description">{description}</p>  
 </div>  
 <div className="campaigns-element-item-content-details">  
 <div className="campaigns-element-item-content-details-goal">  
 <h3 className="font-bold">{`Ціль: ${goal} ETH`}</h3>  
 <ProgressBar  
 goal={goal}  
 alreadyDonated={+alreadyDonated.toFixed(3)}  
 />  
 </div>  
 <p className="campaigns-element-item-content-details-endsAt">{`${convertSecondsToDays(+endsAt)} днів залишолось`}</p>  
 </div>  
 </div>  
 {claimed  
 ? <div>Кампанію вже завершено. Кошти переведені на гаманець організатора.</div>  
 : <DonateButton id={id} donate={donate} />}  
 </div>  
 );  
};

# 

# ДОДАТОК И

Лістинг коду компоненту, який відображає список кампаній, узятий із Ethereum-провайдера.

import { FC } from 'react';  
// Використовуємо наш Ethereum провайдер  
import useEth from '../../contexts/EthContext/useEth';  
// Інтерфейс для опису структури кампанії  
import { Campaign } from '../../interfaces/interface';  
// Компонент, що відображає окрему кампанію  
import { CampaignElement } from './CampaignElement/CampaignElement';  
// Стилі для даного компонента  
import './campaigns.scss';  
  
// Головний компонент  
export const Campaigns: FC = () => {  
 // Отримуємо стан Ethereum провайдеру  
 const { state: { campaigns, isLoading, userAccount } } = useEth();  
  
 // Рендеримо компонент  
 return (  
 <> {/\* Якщо користувач залогінений \*/}  
 {userAccount  
 ? !isLoading  
 ? !campaigns.length  
 ? (<span>Кампаній на данний час не існує.</span>) // Якщо кампаній немає  
 : (  
 <div className="campaigns-container">  
 {/\* Рендеримо список кампаній \*/}  
 {campaigns.map(({  
 id, title, description, goal, alreadyDonated, claimed, endsAt, donate,  
 }: Campaign) => (  
 <CampaignElement  
 id={id}  
 key={id}  
 title={title}  
 description={description}  
 goal={goal}  
 alreadyDonated={alreadyDonated}  
 claimed={claimed}  
 endsAt={endsAt}  
 donate={donate}  
 />  
 ))}  
 </div>  
 )  
 : (<span>Загрузка...</span>) // Якщо дані ще завантажуються  
 : (<span>Вам треба увійти за допомогою гаманця metamask.</span>)}  
 {' '}  
 {/\* Якщо користувач не залогінений \*/}  
 </>  
 );  
};

# 

# ДОДАТОК К

Лістинг коду компоненту з формою для створення кампанії.

import { FC } from 'react';  
// useFormik для створення форми з реактивними властивостями  
import { useFormik } from 'formik';  
// useNavigate для перенаправлення користувача  
import { useNavigate } from 'react-router-dom';  
  
// Ethereum контекст  
import useEth from '../../contexts/EthContext/useEth';  
// Допоміжна функція для отримання екземпляра кампанії  
import { filterCampaignInstance } from '../../contexts/EthContext/helpers/helpers';  
// Actions для взаємодії з Ethereum контекстом  
import { *actions* } from '../../contexts/EthContext';  
// Стилі для форми створення кампанії  
import './CreateCampaignForm.scss';  
  
// Головний компонент форми створення кампанії  
export const CreateCampaignForm: FC = () => {  
 // Ініціалізація navigate для перенаправлення користувача  
 const navigate = useNavigate();  
  
 // Отримання поточного стану Ethereum контексту та функції dispatch для його оновлення  
 const {  
 state: {  
 web3,  
 campaignAbi,  
 crowdfundingPlatformInstance,  
 userAccount,  
 }, dispatch,  
 } = useEth();  
  
 // Ініціалізація formik для керування станом форми  
 const formik = useFormik({  
 initialValues: {  
 title: '',  
 description: '',  
 campaignDuration: '2592000',  
 requiredAmount: '1',  
 },  
 // Обробник події onSubmit  
 onSubmit: async ({  
 title,  
 description,  
 campaignDuration,  
 requiredAmount,  
 }) => {  
 // Обчислення часу закінчення кампанії  
 const endAt = *Math*.floor(Date.now() / 1000) + +campaignDuration;  
 // Конвертуємо ETH в Wei для зберігання в блокчейні  
 const ethToWei = web3.utils.toWei(`${requiredAmount}`, 'ether');  
  
 // Створюємо нову кампанію на блокчейні  
 await crowdfundingPlatformInstance.methods  
 .startCampaign(  
 title,  
 description,  
 ethToWei,  
 endAt,  
 ).send({ from: userAccount })  
 .then(async ({ events: { CampaignStarted: { returnValues } } }: any) => {  
 // Отримуємо екземпляр новоствореної кампанії  
 const campaign = await filterCampaignInstance(  
 web3,  
 campaignAbi,  
 returnValues[1],  
 userAccount,  
 );  
  
 // Додаємо нову кампанію в стан Ethereum контексту  
 dispatch({  
 type: *actions*.addCampaign,  
 data: campaign,  
 });  
  
 return campaign;  
 }).then(({ id }: {id: number }) => {  
 // Перенаправляємо користувача на сторінку новоствореної кампанії  
 navigate(`/campaign/${id}`);  
 });  
 },  
 });  
  
 // Рендеримо форму  
 return (  
 // Компонент форми створення кампанії  
 <div className="create-campaign">  
 <h1>Создайте свою кампанію!</h1>  
 <form onSubmit={formik.*handleSubmit*}>  
 <label htmlFor="title">  
 Назва кампанії  
 <input  
 id="title"  
 placeholder="Введіть назву..."  
 name="title"  
 type="text"  
 onChange={formik.*handleChange*}  
 value={formik.*values*.title}  
 />  
 </label>  
  
 <label htmlFor="description">  
 Опис кампанії  
 <textarea  
 id="description"  
 placeholder="Введіть опис..."  
 name="description"  
 onChange={formik.*handleChange*}  
 value={formik.*values*.description}  
 />  
 </label>  
  
 <label htmlFor="campaignDuration">  
 Час виділенний на сбор коштів  
 <select  
 name="campaignDuration"  
 value={formik.*values*.campaignDuration}  
 onChange={formik.*handleChange*}  
 >  
 <option value="2592000">30 днів</option>  
 <option value="3888000">45 днів</option>  
 <option value="5184000">60 днів</option>  
 </select>  
 </label>  
  
 <label htmlFor="requiredAmount">  
 Необхідна сума сбору (ETH)  
 <input  
 id="requiredAmount"  
 placeholder="Введіть суму яку плануєте зібрати..."  
 name="requiredAmount"  
 type="number"  
 onChange={formik.*handleChange*}  
 value={formik.*values*.requiredAmount}  
 />  
 </label>  
 <button className="form-button" type="submit">Создати</button>  
 </form>  
 </div>  
 );  
};

# ДОДАТОК Л

Відомості про опубліковані тези доповідей:

1. Юрченко А. Є., Шафоростов М. О. Проблеми безпеки та масштабованості в системі Ethereum, пов’язані із застосуванням смартконтрактів // Тези доповідей 13-ї міжнар. наук.-техн. конф. «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління». 2023. Т. 1. С. 82. Сторінки збірника показано на рис. Л.1–Л.3.

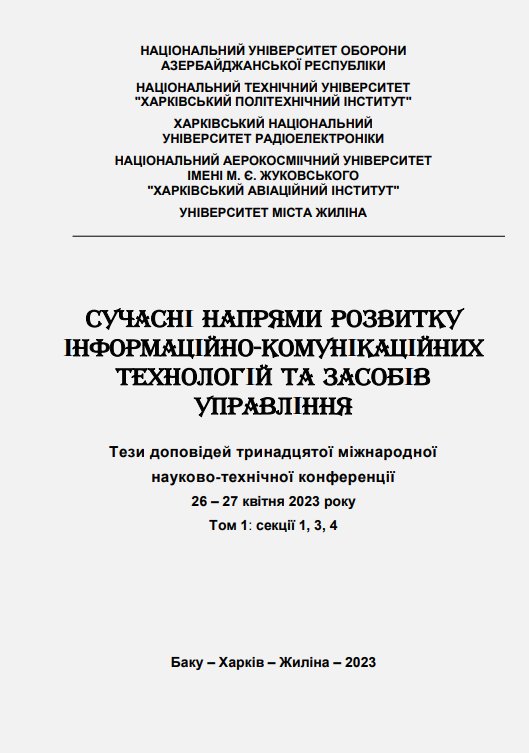


Рисунок Л.1 — Збірник із тезами доповіді, с. 1

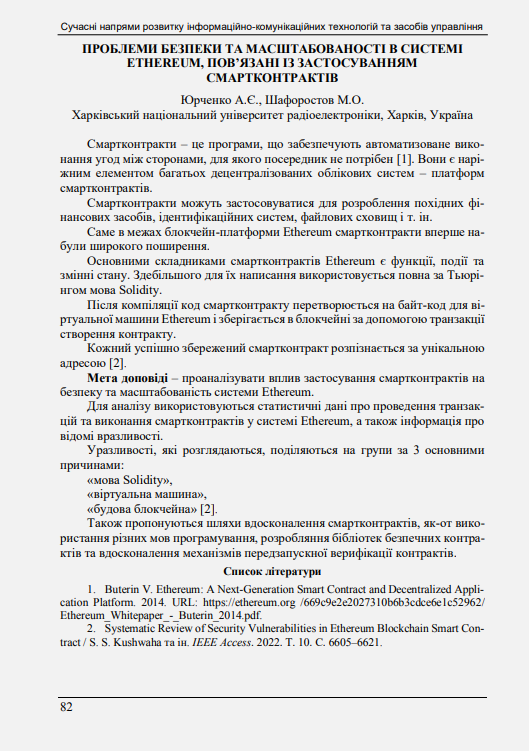


Рисунок Л.2 — Збірник із тезами доповіді, с. 82

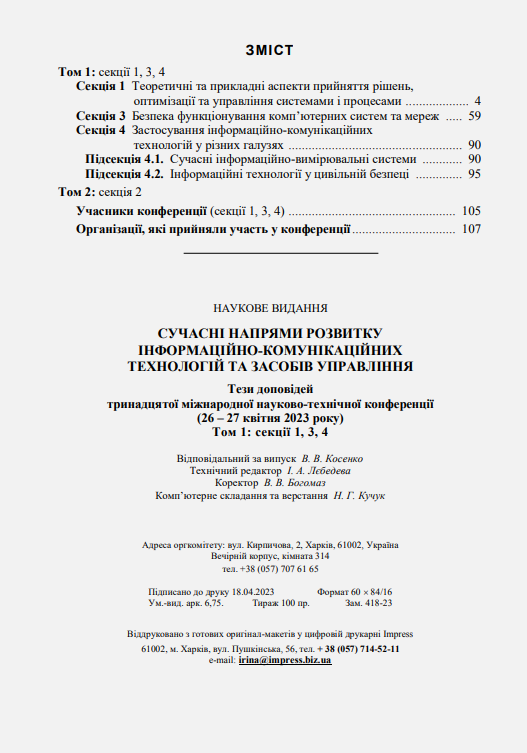


Рисунок Л.3 — Збірник із тезами доповіді, с. 108

ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | Позначення | | | | Найменування | | | | | Дод.  відомості | | |
|  |  | | | | Текстові документи | | | | |  | | |
|  | ГЮІК.ХХХХХХ.554Ст.07ПЗ | | | | Пояснювальна записка | | | | | 55 с. | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | | Графічні документи | | | | |  | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | | Презентаційний матеріал | | | | | 13 слайдів | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | | Інші документи | | | | |  | | |
|  |  | | | | Рецензія | | | | | 1 с. | | |
|  |  | | | | Відгук керівника | | | | | 1 с. | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | |  | | | | |  | | |
|  |  | | | |  | | | | |  | | |
|  |  |  |  |  | | ГЮІК.ХХХ554Ст.07ВД | | | | | | |
|  |  |  |  |  | |
| Змін. | Арк. | Номер докум. | Підп | Дата | |
| Розроб. | | Юрченко А. Є. |  |  | | Безпечне проєктування смартконтрактів і механізми захищення облікових даних у системі Ethereum.  Відомість кваліфікаційної роботи | Літ | | | | Аркуш | Аркушів |
| Перевір. | | Шафоростов М. О. |  |  | | У |  |  | | 76 | 76 |
|  | |  |  |  | |
| Н.контр. | | Шевченко Л. П. |  |  | | ХНУРЕ  Кафедра БІТ | | | | | |
| Затверд. | | Халімов Г. З. |  |  | |