НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

«До захисту допущено»

Завідувач кафедри

_____ В.О. РОМАНКЕВИЧ
(підпис)

"___" червня 20___ р.

Дипломний проєкт

на здобуття ступеня бакалавра

зі спеціальності **123 «Комп'ютерна інженерія»**

на тему: Програмна система для керування власноруч випущеною криптовалютою з використанням технології блокчейн

Виконав: студент IV курсу, групи КВ-91

Селетков Владислав Русланович _____
(підпис)

Керівник доц.каф.СПСКС, к.т.н. Тарасенко-Клятченко О.В. _____
(підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М. _____
(підпис)

Рецензент ст. викладач каф.ПЗКС, к.т.н. Люшенко Л.А. _____
(підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2023 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ В.О. РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)
«___» червня 20___ р.

ЗАВДАННЯ
на дипломний проєкт студента
Селеткова Владислава Руслановича

1. Тема проєкту «Програмна система для керування власноруч випущеною криптовалютою з використанням технології блокчейн», керівник проєкту Тарасенко-Клятченко О.В., кандидат технічних наук, доцент кафедри системного програмування і спеціалізованих комп'ютерних систем, затверджені наказом по університету від «31» травня 2023 р. № 2107-С

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту: див. Технічне завдання

4. Зміст пояснювальної записки
- аналіз літератури та існуючих рішень;
- розробка програмної системи для керування власноруч випущеною криптовалютою;
- тестування системи.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)
- послідовність дій взаємодії з системою. Схема алгоритму;
- обробка прийнятих повідомлень від сервера. Схема алгоритму;

- вхід в обліковий запис користувача. Схема алгоритму;
- операція купівлі токенів у мережі Ethereum. Схема алгоритму;
- презентація за темою проєкту.

6. Консультанти розділів проєкту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я.М., к.т.н., доц. каф. СПСКС		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Видача завдання на дипломне проектування	28.10.2022	
2.	Вивчення літератури за тематикою проєкту	08.11.2022	
3.	Розроблення та узгодження технічного завдання	15.11.2022	
4.	Розроблення структури застосунку	05.12.2022	
5.	Розроблення дизайну та графічних елементів	22.12.2022	
6.	Програмна реалізація застосунку	12.01.2023	
7.	Тестування застосунку	02.03.2023	
8.	Підготовка матеріалів першого розділу дипломного проєкту	11.03.2023	
9.	Підготовка матеріалів другого розділу дипломного проєкту	07.04.2023	
10.	Підготовка матеріалів третього розділу дипломного проєкту	04.05.2023	
11.	Підготовка матеріалів графічної частини проєкту	19.05.2023	
12.	Оформлення технічної документації проєкту	26.05.2023	

Студент _____ В.Р. Селетков
(підпис) (ініціали, прізвище)

Керівник проєкту _____ О.В. Тарасенко-Клятченко
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено керівника дипломного проекту.

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (80 с., 77 рис., 4 додатки).

Об’єкт розробки – програмна система, що дозволяє керувати власною випущеною криптовалютою. Такий застосунок повинен взаємодіяти з технологією блокчейн та реалізовувати базовий функціонал, що дозволяє купувати та продавати власні токени, переглядати баланс користувача та переказувати криптовалюту з одного рахунку на інший.

Комп’ютерна система дозволяє:

- зберігати створені та випущені токени на базі стандарту ERC20 в системі для керування криптовалютою;
- купувати та продавати токени через ether, обмінювати та переглядати баланс токенів на рахунку;
- виконувати операції продажу та переказу токенів без оплати комісії за транзакції;
- використовувати легкий та інтуїтивно зрозумілий інтерфейс застосунку для виконання операцій;
- створити або імпортувати вже існуючий акаунт до застосунку;
- встановити пароль до акаунту, щоб можна було безпечно користуватись криптовалютою;
- мати постійний та швидкий доступ до своїх токенів.

В ході розробки:

- проаналізовано вже існуючі подібні криптовалютні системи на основі технології блокчейн та проведено розбір методів їх створення;
- створено зручний користувацький застосунок для управління криптовалютою;
- розроблено смарт-контракт токена на базі стандарту ERC20 та операції для взаємодії з ним, використовуючи блокчейн;

- створено веб-сервер для взаємодії з користувацьким застосунком та мережею Ethereum;
- реалізовано з’єднання десктопного застосунку з веб-сервером для виконання заданих криптовалютних операцій;
- смарт-контракт було розгорнуто в тестовій блокчейн-мережі Ethereum;
- Використовуючи метод шифрування даних, розроблено захист від несанкціонованого доступу до акаунту користувача.

Впровадження заданої програмної системи для керування криптовалютою дозволить використовувати власний створений токен, мати можливість бути конкурентоспроможним для подальшого інтегрування криптовалюти та її майбутньої реалізації.

Ключові слова: КРИПТОВАЛЮТА, ТОКЕН, СИСТЕМА ДЛЯ КЕРУВАННЯ КРИПТОВАЛЮТОЮ, ПРОГРАМНА СИСТЕМА, БЛОКЧЕЙН, ERC20, ETHEREUM, ETHER, СМАРТ-КОНТРАКТ, ВЕБ-СЕРВЕР, ЗАСТОСУНОК.

ABSTRACT

Qualifying work includes an explanatory note (80 p., 77 fig., 4 applications).

The object of development is a software system that allows you to manage your own cryptocurrency. Such an application should interact with blockchain technology and implement the basic functionality that allows you to buy and sell your own tokens, view the balance of the user and transfer cryptocurrency from one account to another.

The computer system allows:

- store created and released tokens based on the ERC20 standard in a cryptocurrency management system;
- buy and sell tokens by using ether, exchange and view the tokens balance on account;
- perform the sale and transfer of tokens without paying the transaction commission;
- use a simple and intuitive application interface to perform operations;
- create or import an existing account to the application;
- set the password to the account for safe use of cryptocurrency;
- have permanent and quick access to your tokens.

In the course of development:

- analyzed already existing cryptocurrency systems based on blockchain technology and methods of creating them;
- created a convenient user application for managing cryptocurrency;
- a smart contact of the token based on ERC20 and operations for interaction with it has been developed by using blockchain;
- a web server was created to interact with the user application and with the Ethereum network;
- the desktop application connection with the web server has been implemented to perform the given cryptocurrency operations;

- smart contract was deployed in the Ethereum blockchain test network;
- using data encryption method unauthorized access to the user account has been developed.

The introduction of a given cryptocurrency software system will allow you to use your own created token, and be able to be competitive for further integration of cryptocurrency and its future implementation.

Keywords: CRYPTOCURRENCY, TOKEN, CRYPTO, CRYPTOCURRENCY MANAGEMENT SYSTEM, SOFTWARE SYSTEM, BLOCKCHAIN, ERC20, ETHEREUM, ETHER, SMART CONTRACT, WEB SERVER, APPLICATION.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.002 ТЗ	Програмна система для	4		
			керування власноруч			
			випущеною криптовалютою			
			з використанням технології			
			блокчейн			
			Технічне завдання			
	А4	ІАЛЦ.045490.003 ТП	Програмна система для	2		
			керування власноруч			
			випущеною криптовалютою			
			з використанням технології			
			блокчейн			
			Відомість технічного			
			проєкту			
	А4	ІАЛЦ.045490.004 ПЗ	Програмна система для	80		
			керування власноруч			
			випущеною криптовалютою			
			з використанням технології			
			блокчейн			
			Пояснювальна записка			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045490.001 ОА		
Розробив		Селетков В.Р.			*Програмна система для керування власноруч випущеною криптовалютою з використанням технології блокчейн*	Літ.	Аркуш 1 / Аркушів 2
Перевірила		Тарасенко-Клятченко О.В.				КПІ ім. Ігоря Сікорського, ФПМ КВ-91	
Н. контроль		Клятченко Я.М.			***Опис альбому***		
Зав. каф.		Романкевич В.О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045490.005 Д1	Послідовність дій взаємодії	1		
			з системою. Схема			
			алгоритму			
	А4	ІАЛЦ.045490.006 Д2	Обробка прийнятих	1		
			повідомлень від сервера.			
			Схема алгоритму			
	А4	ІАЛЦ.045490.007 Д3	Вхід в обліковий запис	1		
			користувача. Схема			
			алгоритму			
	А4	ІАЛЦ.045490.008 Д4	Операція купівлі токенів	1		
			у мережі Ethereum.			
			Схема алгоритму			
		Диск CD-ROM	Текст пояснювальної	1		
			записки.			
			Графічний матеріал			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045490.001 ОА	Арк. 2

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Програмна система для керування власноруч випущеною криптовалютою з використанням технології блокчейн».

Галузь застосування: розробка системи для керування власним токеном, який можна буде використовувати в торгівлі, інвестуванні, оплаті, зберіганні та в різноманітних фінансових послугах.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проєкту є розробка десктопного застосунку у вигляді програмної системи для керування криптовалютою, яка повинна забезпечувати зручне та безпечне надання доступу користувачу у будь-який час, зберігання, переказ, купівлю та продаж власного створеного токена.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації для розроблення є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях, електронні статті та відповідні ресурси у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до програмного продукту, що розробляється

- сумісність з будь-якою операційною системою;

- приємний інтерфейс застосунку для користувачів;
- можливість безпечно зберігати криптовалюту;
- впровадження обміну власного токена на ether та навпаки;
- наявність надійного захисту особистих даних;
- можливість переказу криптовалюти на інший акаунт;
- впровадження перегляду балансу криптовалюти;
- виконання операцій переказу та продажу без оплати комісії;
- можливість перегляду адреси та приватного ключа акаунту;
- функціонал створення, авторизації та імпорту акаунту до застосунку.

5.2 Вимоги до апаратного забезпечення

- процесор: Intel Core i3, AMD Ryzen 3;
- оперативна пам’ять: 2 Гб;
- простір на диску: 1 Гб;
- наявність доступу до мережі Wi-Fi (IEEE 802.11 b/g/n).

5.3 Вимоги до програмного та апаратного забезпечення користувача

- операційна система Windows, Linux або Mac OS;
- наявність доступу до мережі Wi-Fi (IEEE 802.11 b/g/n).

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів
1.	Видача завдання на дипломне проектування	28.10.2022
2.	Вивчення літератури за тематикою проєкту	08.11.2022
3.	Розроблення та узгодження технічного завдання	15.11.2022
4.	Розроблення структури застосунку	05.12.2022
5.	Розроблення дизайну та графічних елементів	22.12.2022
6.	Програмна реалізація застосунку	12.01.2023
7.	Тестування застосунку	02.03.2023
8.	Підготовка матеріалів першого розділу дипломного проєкту	11.03.2023
9.	Підготовка матеріалів другого розділу дипломного проєкту	07.04.2023
10.	Підготовка матеріалів третього розділу дипломного проєкту	04.05.2023
11.	Підготовка матеріалів графічної частини проєкту	19.05.2023
12.	Оформлення технічної документації проєкту	26.05.2023

ВІДОМІСТЬ ТЕХНІЧНОГО ПРОЄКТУ

№ з/п	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
1	А4	ІАЛЦ.045490.000 ТЗ	Завдання на дипломний	2		
			проєкт			
2	А4	ІАЛЦ.045490.001 ОА	Опис альбому	2		
3	А4	ІАЛЦ.045490.002 ТЗ	Технічне завдання	4		
4	А4	ІАЛЦ.045490.003 ТП	Відомість технічного	1		
			проєкту			
5	А4	ІАЛЦ.045490.004 ПЗ	Пояснювальна записка	80		
6	А4	ІАЛЦ.045490.005 Д1	Послідовність дій взаємодії	1		
			з системою. Схема			
			алгоритму			
7	А4	ІАЛЦ.045490.006 Д2	Обробка прийнятих	1		
			повідомлень від сервера.			
			Схема алгоритму			
8	А4	ІАЛЦ.045490.007 Д3	Вхід в обліковий запис	1		
			користувача. Схема			
			алгоритму			
9	А4	ІАЛЦ.045490.008 Д4	Операція купівлі токенів	1		
			у мережі Ethereum.			
			Схема алгоритму			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045490.003 ТП		
Розробив		Селетков В.Р.			*Програмна система для керування власноруч випущеною криптовалютою з використанням технології блокчейн*	Літ.	Аркуш 1 / Аркушів 1
Перевірила		Тарасенко-Клятченко О.В.					
Н. контроль		Клятченко Я.М.			***Відомість технічного проєкту***	КПІ ім. Ігоря Сікорського, ФПМ КВ-91	
Зав. каф.		Романкевич В.О.					

Пояснювальна записка

до дипломного проєкту

на тему: Програмна система для керування власноруч випущеною криптовалютою з використанням технології блокчейн

Київ – 2023 року

ЗМІСТ

ДОДАТКИ

Додаток 1. Копії графічних матеріалів

- ІАЛЦ.045490.005 Д1. Послідовність дій взаємодії з системою. Схема алгоритму
- ІАЛЦ.045490.006 Д2. Обробка прийнятих повідомлень від сервера. Схема алгоритму
- ІАЛЦ.045490.007 Д3. Вхід в обліковий запис користувача. Схема алгоритму
- ІАЛЦ.045490.008 Д4. Операція купівлі токенів у мережі Ethereum. Схема алгоритму

Додаток 2. Фрагменти програмного коду

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ETH – Ether (cryptocurrency).

BDP – Bakalavra Diploma Project (власний випущений токен системи).

ERC20 – Ethereum Request-for-Comments #20 (token standard).

GUI – Graphical User Interface.

ПЗ – програмне забезпечення.

EVM – Ethereum Virtual Machine.

PK – Private Key (приватний ключ).

ABI – Application Binary Interface.

JSON – JavaScript Object Notation.

RPC – Remote Procedure Call.

URL – Uniform Resource Locator.

AES – Advanced Encryption Standard.

ВСТУП

Значення криптовалюти доволі швидко розширюється протягом останнього десятиліття, і з цим також зростає попит на безпечні, функціональні, прибуткові та зручні для користувачів токени.

В даному дипломному проєкті фокус зосереджено на створенні програмної системи для керування власноруч випущеною криптовалютою з використанням технології блокчейн.

Під такою системою розглядається спеціальний криптовалютний гаманець, який включає в себе новий створений та випущений токен.

Мета полягає в розробці системи, яка проста у використанні, має приємний інтерфейс та широкий спектр функцій, включаючи можливість купувати, продавати, переглядати баланс та переказувати токени іншим користувачам. Оскільки криптовалюти зарекомендували себе як ефективний та безпечний засіб зберігання коштів, то створення власної криптовалюти та програми для неї повинно бути перспективним та актуальним напрямком розвитку бізнесу та багатьох інших технологічних можливостей.

Розробка заданої системи пройшла через велику кількість досліджень та тестувань, щоб забезпечити її надійність та безпеку. Такий застосунок повинен бути актуальним, оскільки має зручний інтерфейс, який зрозумілий як для початківців, так і для досвідчених користувачів криптовалют. Також, в ньому присутня функція безпеки, що використовує логін та пароль, аби користувачі мали надійний захист криптовалютних активів.

Проєкт реалізовано за допомогою Qt framework, Node.js та мови програмування смарт-контрактів Solidity. Робота включає створення власної криптовалюти на основі мережі Blockchain Ethereum стандарту ERC20, написання смарт-контракту для виконання криптовалютних операцій, розробка веб-сервера для обробки транзакцій та GUI, що реалізує функціональність системи для керування криптовалютою.

Особливістю заданого застосунку являється надання користувачам можливості продавати та переказувати криптовалюту без оплати комісії за транзакцію.

Отже, завданням дипломного проєкту є розробка всебічної та зручної для користувачів системи для керування власною створеною криптовалютою, яка включає в собі всі основні операції для взаємодії з токеном та має достатньо надійний захист. Основна реалізована можливість застосунку – це переказувати та продавати токени без оплати комісії. Така система повинна задовольняти зростаючий попит на безпечні та ефективні інструменти управління криптовалютою.

1. АНАЛІЗ ЛІТЕРАТУРИ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Основні поняття та визначення

У першому розділі дипломного проєкту розглядаються основні терміни та визначення, які стосуються теми дослідження. Необхідно надати визначення деяких основних понять, аби краще зрозуміти всю концепцію розробки та використання криптовалют.

Криптовалюта – цифрова валюта, яка використовує децентралізовану платіжну систему для емісії та обліку транзакцій без участі центральної влади. Однією з основних особливостей є збереження даних про транзакції в блокчейні, що є розподіленою базою даних, доступною для всіх учасників мережі [1].

Приватний ключ – це криптографічний ключ, який надає доступ власнику криптовалюти для керування криптовалютою та надає можливість підпису транзакцій. Цей ключ є унікальним, за допомогою нього існує можливість виконувати різноманітні операції з токенами, використовуючи спеціалізовані бібліотеки, такі як Web3 та Ethers. Приватний ключ повинен бути добре захищеним та зберігатися в безпечному місці.

Блокчейн – це розподілена база даних, в якій інформація зберігається у вигляді ланцюжка блоків, що постійно збільшується. Кожен блок містить хеш-посилання на попередній блок, час його створення та дані транзакцій, які знаходяться у відкритому доступі. Для запобігання фальсифікації даних кожен блок містить хеш попереднього блоку, що робить його непідробним [2].

Криптогаманець – це програмна система, яка забезпечує зберігання криптовалюти, її переказ, продаж та купівлю. Вона може зберігати приватний ключ для доступу до своїх активів та дозволяти користувачам виконувати транзакції. Криптовалютні гаманці бувають різними за своєю функціональністю та механізмами захисту даних своїх клієнтів. До того ж, таке сховище зберігання даних може бути представлене як фізичний, тобто апаратний пристрій.

Описані вище поняття надають змогу краще зрозуміти як влаштоване таке програмне забезпечення, подальші визначення будуть стосуватися структури розробки самого токена, який використовується у проєкті.

ERC20 – це стандарт Ethereum, який описує функціональність токенів платформи Ethereum. Криптовалюта, яка створена на основі стандарту ERC20, є сумісною з будь-яким іншим сервісом або застосунком, що використовує той самий стандарт [3].

Смарт-контракт – це програмний код, який самостійно виконується у блокчейні та містить деякі правила та умови, що дозволяють транзакціям автоматично виконуватись. Вони розширюють створення децентралізованих програм та забезпечують основу багатьох проєктів написаних для блокчейну.

Ether (ETH) – рідна криптовалюта мережі Ethereum, яка використовується для оплати комісії за транзакцію та обчислювальні послуги в мережі.

Нижче перераховані засоби, які використовувалися у взаємодії з блокчейном, створенні програмної системи та токена.

Solidity – це спеціальна мова програмування високого рівня, яка використовується у написанні смарт-контрактів для блокчейну Ethereum [10].

Web3 – це бібліотеки та інструменти, які дозволяють розроблювати децентралізовані застосунки на блокчейні. Web3 надає можливість створювати функції, які можуть взаємодіяти зі смарт-контрактами, зчитувати і записувати дані на блокчейні, керувати операціями та виконувати інші взаємодії.

Ethers – бібліотека для розробки децентралізованих застосунків на основі платформи Ethereum. Вона вміщує в себе інтерфейс для взаємодії з Ethereum мережею та смарт-контрактами. Має великий функціонал щодо виконання операцій з криптовалютою.

Qaterial – це інструмент для розробки компонентів користувацького інтерфейсу (UI), що використовується для створення ПЗ з використанням Qt framework [14].

1.2 Основи розробки криптовалют та блокчейну на платформі Ethereum

Розвиток криптовалют та блокчейн технологій революціонізував теперішній світ фінансів та транзакцій, тому що така розробка впроваджує альтернативу традиційним валютам, забезпечуючи найголовніші відмінності між ними саме у захисті та децентралізації.

Блокчейн Ethereum – це розподілена обчислювальна мережа, в якій зберігається база даних всіх транзакцій, які виконуються на платформі. Ethereum блокчейн складається з неймовірної кількості вузлів, що містять копії однакових баз даних. Кожен такий вузол може здійснювати операції з мережею, зберігати дані та створювати нові блоки, які у свою чергу будуть додаватись вже до утвореного заздалегідь ланцюжка.

Ethereum являє собою децентралізовану платформу, яка дозволяє розробляти та розгортати смарт-контракти. Платформа створена на основі технології блокчейн, в основі якої лежить надійне зберігання та передача даних між учасниками. Таким чином, блокчейн працює так, що жоден користувач не має доступу до всієї мережі, а інформація в усій цій екосистемі передається через безліч кількість під’єднаних комп’ютерів у мережі. Як вже зазначалось, одним із важливих елементів Ethereum є криптовалюта Ether (ETH), яка використовується для оплати транзакцій у мережі. Дана платформа є однією із найпопулярніших для створення децентралізованих застосунків, оскільки містить в собі великий функціонал для програмування та розгортання смарт-контрактів.

Детальніше необхідно зупинитись на стандарті токенів мережі Ethereum. Платформа підтримує розробку власних криптовалют на базі ERC20. Цей стандарт має певний перелік правил, яких необхідно дотримуватись для того, щоб токен міг функціонувати та бути розгорнутим у мережі Ethereum. Такі дії мають важливе значення в екосистемі, оскільки за допомогою такого стандарту з’явилась можливість, забезпечуючи загальні правила, для створення та використання різних токенів на блокчейні Ethereum.

Перелік необхідних правил до ERC20 [3]:

- totalSupply: функція, яка визначає обсяг загальної кількості токенів, що будуть створені;
- balanceOf: метод, який повертає баланс певного акаунту (використовуючи адресу користувача) на рахунку;
- transfer: описує метод, який надає можливість власнику криптовалюти переказувати токени на іншу адресу у мережі;
- transferFrom: функція, що надає можливість переказувати криптовалюту з однієї адреси на іншу, якщо є дозвіл від адреси, токени якої переказуються;
- approve: метод, який дозволяє іншим акаунтам отримати дозвіл на переказ токенів з вашого рахунку;
- allowance: метод, який описує можливість надавати деякому адресу у мережі перевіряти суму токенів, які йому дозволяється переказати від власника цих токенів;
- symbol: функція, яка повертає символ токена (ETH);
- name: функція, що повертає назву токена;
- decimals: метод, який повертає кількість десяткових знаків токена;
- events: стандарт вимагає реалізації певних подій, таких як Transfer та Approval, вони повинні бути викликані в той час, коли виконуються методи transfer та approve.

Ethereum має вбудовану мову програмування Solidity, яка дозволяє створювати смарт-контракти. Вони в свою чергу є програмами, які автоматично виконують умови, записані в них. Таким чином, забезпечується безпечна обробка транзакцій [10].

Також, необхідно розуміти, що взаємодія між криптовалютами та блокчейном у мережі Ethereum здійснюється за допомогою універсального протоколу взаємодії, який має назву Ethereum Virtual Machine (EVM).

EVM – це ПЗ, яке виконує написаний код децентралізованих застосунків та контрактів на блокчейні Ethereum. Тобто, все що розробляється на платформі

Ethereum виконуються в контексті EVM. Кожен вузол мережі Ethereum має свою власну копію EVM, що забезпечує їх сумісність та однорідність [4].

Існують два типи облікових записів в Ethereum – це акаунти та контракти. Акаунти контролюються приватним ключем (private key), в той час як контракти розгортаються у мережі та контролюються написаним до них кодом. Обидва типи облікових записів можуть отримувати, зберігати та відправляти ETH та токени, а також взаємодіяти з розгорнутими смарт-контрактами.

Однак, є деякі принципові відмінності між акаунтами та контрактами. Акаунт можна створити безкоштовно, тоді як створення контракту потребує додаткових витрат за зберігання у мережі Ethereum.

Акаунти можуть ініціювати транзакції та проводити угоди лише на переказ ETH та токенів, тоді як контракти можуть надсилати транзакції лише у відповідь на отримання транзакції та виконувати багато різних дій, таких як переказ токенів або навіть створення нового договору [4].

Кожен акаунт складається з криптографічної пари ключів: публічного та приватного, а контракти в свою чергу не мають жодних приватних ключів та контролюються лише логікою написаного коду у смарт-контракті. Адреса контракту зазвичай надається тоді, коли смарт-контракт розгортається у мережі.

На рис. 1 зображено діаграму, яка ілюструє з яких обов'язкових атрибутів повинен складатись Ethereum акаунт [6].

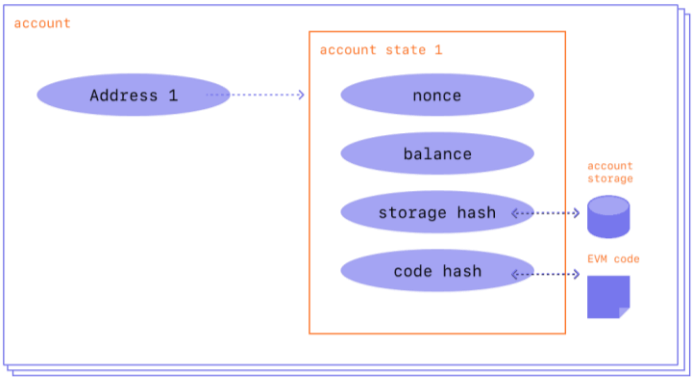

Рисунок 1 – Загальна структура акаунту

Ethereum гаманець має цілий ряд основних компонентів, яких бажано дотримуватися.

Найголовнішим являється приватний ключ користувача. Він вміщає в собі деякий випадково згенерований 256-бітовий довгий код, який дозволяє власнику такого ключа підписувати транзакції. PK повинен зберігатися в спеціальному ключовому файлі та в зашифрованому вигляді в пам'яті комп'ютера або на спеціальному апаратному пристрої, який має назву «холодний гаманець». Також, для надійної безпеки, такий файл доцільно захистити паролем. Приватний ключ повинен надійно зберігатись, адже з його допомогою можна виконувати будь-які операції зв'язані з криптовалютою.

Не менш важливим компонентом є публічна адреса застосунку. Це значення є унікальним та відповідає конкретному акаунту, на якому зберігаються віртуальні активи. Криптовалютна система (акаунт) складається з двох основних компонент, таких як приватний та публічний ключ. В свою чергу, публічний ключ генерується з приватного ключа за допомогою ECDSA (Elliptic Curve Digital Signature Algorithm) алгоритму. Для створення загальнодоступної адреси акаунту, беруться останні 20 байт від хешу публічного ключа, обчисленого за алгоритмом Keccak-256, та додається префікс 0x до початку отриманої послідовності.

Приватний ключ необхідний для підписання повідомлень та транзакцій, тому його вкрай важливо надійно зберігати. На відміну від приватних ключів, з яких можна утворювати публічні ключі, не існує можливості згенерувати приватні ключі з публічних. Це означає, знаючи PK, надається можливість підписувати транзакції та повідомлення, доводити свою власність [6].

Ще одним із елементів захисту сучасних гаманців є seed-фраза. Вона являється набором випадково згенерованих слів, які необхідно запам'ятати або виписати при створенні акаунту. Така фраза зазвичай складається із 12 або 24 слів та використовується для відновлення доступу до застосунку в разі втрати PK. У світі таких програм є важливим елементом безпеки, але не обов'язковим, адже будь-яка особа може отримати доступ до застосунку, знаючи seed-фразу.

Необхідно виконати поглиблений розбір поняття транзакції у мережі Ethereum. Транзакція – це деяка дія, яка запускається акаунтом, власником якого є людина, а не смарт-контракт.

Наприклад, якщо Микита хоче надіслати 5 ETH Владиславу, то з рахунку Микити буде знято 5 ETH, а Владиславу відповідно буде нараховано 5 ETH. Задані маніпуляції відображаються в одній транзакції.

Операції, які впливають на стан EVM, мають бути розповсюджені по всій мережі. Будь-який вузол може передати запит на виконання транзакції до EVM, після передачі буде виконано транзакцію та всі викликані зміни після виконання заданої транзакції будуть розповсюджені всім учасникам мережі Ethereum.

Угоди такого типу потребують плати за газ (комісію) та повинні бути включені у затверджений блок.

Газ – це одиниця виміру, яка описує кількість обчислювальних зусиль, необхідних для виконання певних операцій на цій платформі. Кожна транзакція в мережі Ethereum вимагає обчислювальних ресурсів, тому всі ці операції потребують певної плати. Тобто, газ є оплатою комісії за надані послуги, які полягають у виконанні транзакції в мережі. Важливою особливістю є те, що газ списується незалежно від того, що транзакція була оброблена вдало чи ні.

Наприклад, прості операції по переказу криптовалюти коштують 21 000 одиницю газу. Якщо газу виявилося забагато для оплати транзакції, то решта, яка не була використана, обов’язково повертається до акаунту користувача.

Підтверджена транзакція повинна включати такі компоненти [7]:

- from: адреса відправника, що підписує транзакцію;
- recipient: адреса отримувача;
- signature: ідентифікатор відправника;
- nonce: лічильник, який збільшується при кожній новій транзакції;
- value: кількість ETH, що переказується;
- data: поле, куди можна написати якусь додаткову інформацію;
- gasLimit: максимальний об’єм газу, що може бути витрачено на транзакцію;

- maxPriorityFeePerGas: максимальна ціна споживаного газу;
- maxFeePerGas: максимальна плата за одиницю газу.

Зазвичай, більшість транзакцій, що виконуються, отримують доступ до контракту з акаунту. Смарт-контракти в основному пишуться на мові програмування Solidity, тому вони можуть інтерпретувати свої атрибути даних відповідно до бінарного інтерфейсу програми (ABI).

Для наочності, на рис. 2 зображено приблизний вигляд об’єкту транзакції, яка підписана за допомогою PK.

```
{
  "id": 5,
  "jsonrpc": "2.0",
  "method": "account_signTransaction",
  "params": [
    {
      "from": "0x8674f63feb8dc028739e00a99c43fe20cf7fd99db",
      "gas": "0x55555",
      "maxFeePerGas": "0x1234",
      "maxPriorityFeePerGas": "0x1234",
      "input": "0xabcd",
      "nonce": "0x5",
      "to": "0x07b567b7ed3b7a634680b4c194385abfcb693fe0",
      "value": "0x1460"
    }
  ]
}
```

Рисунок 2 – Об’єкт підписаної PK транзакції

Оплата газу виконується в токенах ETH. Але існує ще дві одиниці вимірювання: wei та gwei. Вони використовуються в Ethereum для вимірювання вартості транзакцій, де gwei дорівнює 0,000000001 ETH (10^{-9} ETH). В свою чергу, wei – це найменша одиниця виміру вартості в Ethereum, вона дорівнює 10^{-18} ETH.

Чим вища вартість газу, тим швидше транзакція буде оброблена мережею. Газова плата розраховується як добуток вартості газу на його кількість, необхідну для виконання операції.

Наведений далі приклад проілюструє механіку використання газу. Якщо задати ліміт газу у розмірі 70 000 для простого переказу ETH, то EVM повинна використати 21 000, тоді рештою назад повернеться 49 000.

З іншого боку, якщо вказати недостатню кількість газу, наприклад, 15 000 для переказу ETH у мережі, то EVM повинна використати 15 000 заданих одиниць, якими користувач намагається оплатити транзакцію, але вона не завершиться. Далі EVM все повертає так, як і було до цієї транзакції. Але оскільки той, хто намагався виконати задану операцію, спожив 15 000 газу, то ця кількість все одно буде використана [8].

Для завершення даного підрозділу, нижче описано ще декілька, важливих для розуміння всієї екосистеми, коротких понять.

Щодо з'єднання усієї платформи Ethereum, можна сказати, вона є децентралізованою мережею комп'ютерів, відомих як вузли, які працюють на ПЗ, що виконує перевірку блоків та даних щодо транзакцій.

Для того, щоб стати вузлом у мережі Ethereum, необхідно запустити на своєму комп'ютері програмне забезпечення, відоме як клієнт.

Вузол – це будь-який екземпляр клієнтського ПЗ в мережі Ethereum, який підключений до інших комп'ютерів, де також працює таке ж саме ПЗ, і разом вони утворюють цілу мережу.

Клієнт – це реалізація Ethereum, яка перевіряє дані відповідно до правил протоколу та забезпечує безпеку мережі.

Також, існує деякий механізм консенсусу Ethereum, за допомогою якого учасники мережі досягають згоди про те, який надалі блок буде додано до блокчейну. Таким чином, вони повинні мати однакову копію бази даних у всіх вузлах мережі, і для цього необхідно мати спосіб підтвердження транзакцій та блоків, який був би стійким до шахрайства та випадкових помилок.

Зараз платформа Ethereum використовує механізм консенсусу, який має назву Proof-of-Stake (PoS) протокол.

Зрозумілою складовою платформи Ethereum є мережі. Вони являють собою різні середовища, від яких можна отримати доступ для програмування, тестування або виробництва смарт-контрактів та інших продуктів.

Кожна мережа в своїй суті є блокчейн-мережею, яка побудована на основі протоколу Ethereum і взаємодіє з головною мережею (Mainnet). Також, існує велика кількість тестових мереж, вони всі мають свій власний набір правил, консенсус-механізм і криптовалюту, що дозволяє програмістам створювати та запускати децентралізовані застосунки, які можуть працювати на тестових мережах, а згодом перенесені на головну. Особливістю таких мереж є те, що вони дозволяють не витрачати реальних коштів на виконання транзакцій.

В даній дипломній роботі використовувалась тестова мережа Sepolia, яка за замовчуванням рекомендується як одна з кращих для розробки застосунків.

Отже, для розробки системи для керування криптовалютою, необхідно мати глибоке розуміння всієї екосистеми, треба добре розуміти стандарти ERC20 для розробки токенів, вивчити основні концепції програмування на цій платформі, а саме смарт-контракти на мові Solidity. Необхідно мати уявлення, яким чином розгортається у мережі смарт-контракт власної криптовалюти та як необхідно задавати параметри для токена [10].

Також, застосунок повинен мати інтерфейс для взаємодії зі смарт-контрактами, які відповідають за виконання операцій з криптовалютою, а саме переказ токенів, перегляд балансу, купівля та продаж криптовалюти.

Крім того, повинна бути впроваджена якісна безпека взаємодії між розроблюваною системою та мережею Ethereum. Оскільки, криптовалюта є децентралізованою системою, дані щодо транзакцій зберігаються на різних вузлах мережі, які можуть бути не надійними та використовуватись у шахрайстві. Тому, розробка безпечних алгоритмів для захисту приватних ключів користувачів та взаємодії з платформою Ethereum має бути в пріоритеті.

1.3 Аналіз існуючих мереж та систем для керування криптовалютою

В цій частині роботи розглянуто важливу роль у розумінні сучасного стану системи криптовалют та пошук інструментів, які можуть бути використані для управління власним випущеним токеном.

Необхідно провести комплексний аналіз різноманітних систем та криптовалюти, що дозволить визначити їхні можливості, переваги та недоліки. Дослідження існуючих рішень у визначеній галузі необхідно для розробки ефективного застосунку, що забезпечить надійне зберігання токенів, який повинен мати безпечне та зручне управління власною криптовалютою.

Аналіз різних гаманців та токенів дозволить знайти необхідні відмінності, які можуть бути застосовані для написання кращого застосунку, а саме: функціональні можливості, основні характеристики, архітектура, приватність, сумісність, вартість та доступність. Щодо функціональних можливостей криптогаманця, то таких може бути достатньо велика кількість, тому необхідно виділити основні з них, такі як створення нових акаунтів, імпортування акаунтів, переказ криптовалюти, перегляд історії транзакцій, підтримка криптовалютних стандартів тощо. Також, важливо проаналізувати рівні безпеки, які включають збереження приватних ключів, аутентифікацію, шифрування даних та захист від сторонніх осіб.

Ознайомлення з існуючими рішеннями, допоможе визначити потенційні можливості для покращення інтерфейсу та зручності використання власної системи, а саме: розробка GUI, підтримка базових операцій з токенами, можлива інтеграція з іншими сервісами та платформами, що забезпечить більшу функціональність та розвиток у подальшому обміні криптовалютами.

Отримані результати аналізу повинні бути основою системи для керування криптовалютою, що розробляється в дипломному проєкті. Аналіз існуючих рішень дозволить знайти найкращі практики та врахувати їх у роботі. Варто відзначити, що аналіз повинен бути детальним та об’єктивним, з урахуванням різних аспектів технологічного, функціонального та безпекового характеру.

Наступні криптовалютні застосунки, які використовуються великою кількістю користувачів для зберігання своїх активів, описані нижче. Очевидно, що всі вони мають дещо схожий функціонал, але мають суттєві відмінності, які впливають на всю роботу застосунку в цілому.

MetaMask (рис. 3) – це система (криптогаманець), яка надає користувачам зручний спосіб для керування їх токенами та взаємодії з децентралізованими застосунками, включаючи блокчейн-платформу Ethereum. MetaMask являє собою потужний та популярний інструмент для поціновувачів криптовалюти [9].

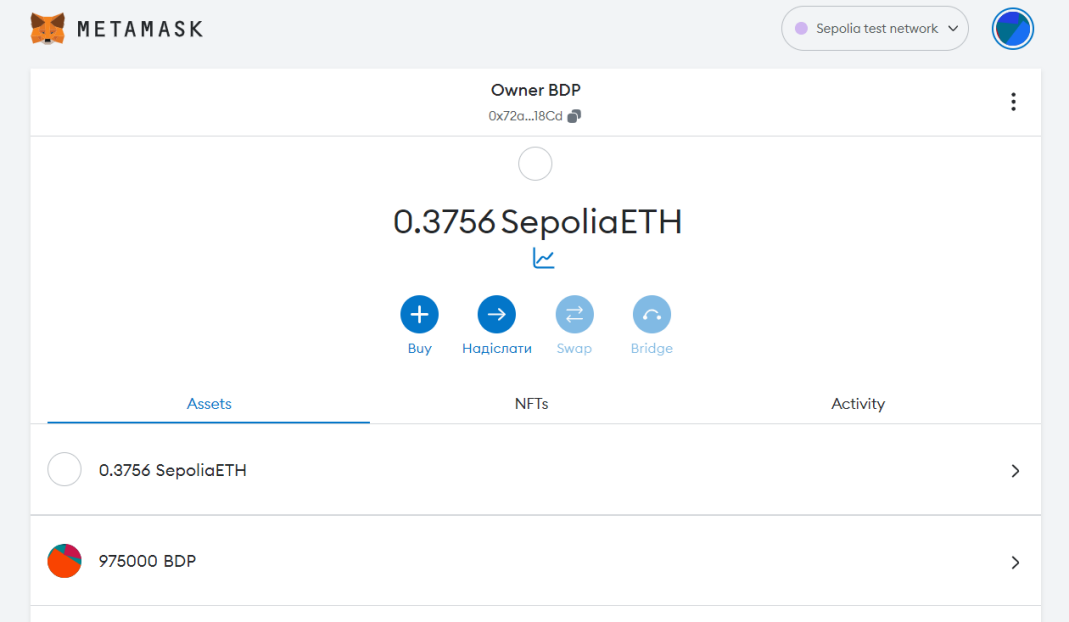

Рисунок 3 – Головне вікно MetaMask

Якщо детальніше розглянути рис. 3, то можна побачити таке поле, як SepoliaETH. Задана характеристика показує кількість ETH на балансі користувача в тестовій мережі Sepolia. Також, зверху надруковано адресу акаунту, яка починається на 0x. Головне вікно гаманця має такі опції, що дозволяють користувачу купити, надіслати, обміняти криптовалюту та переглянути її баланс.

Наприклад, цей гаманець є достатньо функціональним. Він підтримує базові операції для керування криптовалютою, дозволяє створювати та імпортувати акаунти.

Дозволяє користувачам взаємодіяти з децентралізованими застосунками, що базуються на блокчейні.

Надає можливість створювати кілька гаманців в одному застосунку, що є зручним, якщо декому необхідно розподілити свої активи по різним адресам. Звісно, що кожен новий гаманець являє собою новий акаунт, який складається з іншого унікального публічного та приватного ключа.

Захист забезпечений надійними криптографічними способами, які шифруються паролем, який необхідно ввести для того, щоб увійти в гаманець. Аналізуючи застосунок, можна виявити, що приватні ключі зберігаються локально на комп'ютері в зашифрованому вигляді, а не знаходяться на серверах. Такий метод є надійним, тому що існує велика ймовірність того, що приватний ключ може бути вкрадений.

MetaMask підтримує не тільки блокчейн Ethereum, а багато інших мереж, що вказує на зручність такої реалізації. Тому, завдяки інтеграції різних платформ, можна зберігати велику різноманітну кількість токенів на одному акаунті.

Детальніше необхідно розібратись з безпекою гаманця. На рис. 4 зображено приватний ключ акаунту, який розкривається тільки з паролем [9].

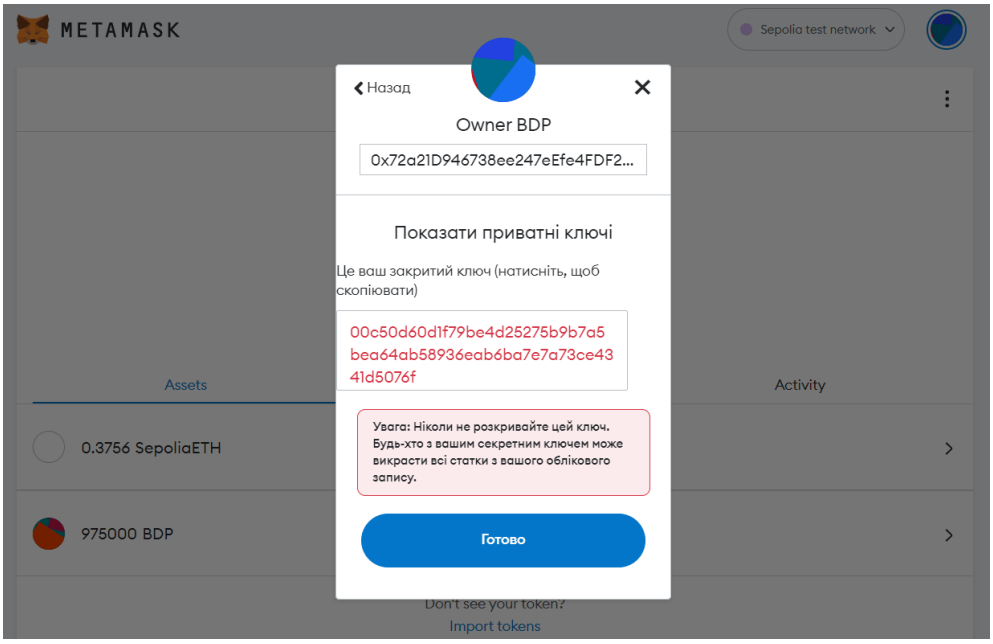

Рисунок 4 – Приватний ключ користувача MetaMask

Як вже зазначалось, MetaMask зберігає ключі локально на пристрої користувача, що забезпечує високий рівень безпеки та контролю, ще й застосунок утворює додатковий ступінь безпеки за допомогою встановлення пароля, що в деякому значенні подвоює безпекові заходи.

Також, кожна транзакція, яка здійснюється в цій системі, повинна бути неодмінно підтверджена, що виключає випадкового виконання якоїсь операції та забезпечує більший контроль над токенами.

Застосунок володіє функцією, яка автоматично перевіряє адресу на яку користувач хоче переказати токени. Працює це таким чином, що система враховує тип адреси та формат, що використовується у мережі, щоб упевнитися, що людина переказує криптовалюту на правильну адресу.

MetaMask інтегрується через розширення в браузер. Використання такого блокчейн-розширення дозволяє підвищити безпеку, оскільки вони перевіряють URL-адреси веб-сайтів та повідомляють попередженням щодо потенційно небезпечних та шахрайських застосунків.

Даний криптогаманець популярний за зручність у використанні, що робить його актуальним серед користувачів. Далі наведено ключові аспекти, які можуть пояснити таку тенденцію.

Досить простий, приємний та зрозумілий інтерфейс, який інтуїтивно впливає на кожного користувача, тому що є примітивним в освоєнні. Інтерфейс має однотонний дизайн та логічну структуру, такі властивості полегшують навігацію та використання функціоналу гаманця.

MetaMask має можливість інтеграції з різними найпопулярнішими сервісами та застосунками, що дозволяє користувачам легко взаємодіяти з ними. Підтвердженням цьому є те, що MetaMask може використовуватись для купівлі токенів практично на всіх децентралізованих біржах, брати участь в криптовалютних заходах та багато іншої функціональності. Наприклад, цей гаманець можна підключити до веб-сайту і виконувати операції, що зв’язані з купівлею товарів та послуг.

Транзакції в MetaMask виконуються швидко, система вміє автоматично розпізнавати опції, які необхідні для відправки транзакцій, тому, значно спрощується процес їх підтвердження та підписання. На рис. 5 показано приклад виконання однієї з транзакцій, яка реалізована в MetaMask [9].

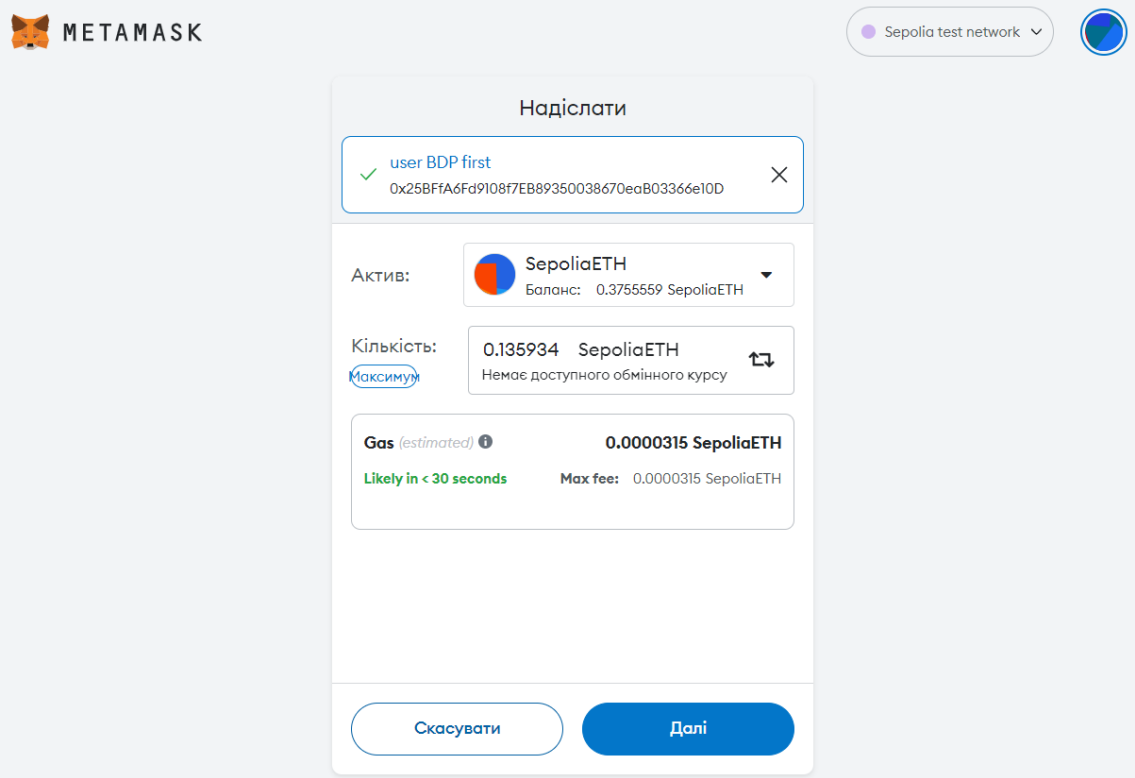

Рисунок 5 – Транзакція переказу токенів у MetaMask

Очевидно, що ця система працює зі смарт-контрактами. Це з'єднання технічно реалізовано в наступних описаних деталях.

Завдяки вбудованій підтримці смарт-контрактів, користувачі можуть надсилати та отримувати криптовалюту, підписувати транзакції, купувати та продавати токени. Вони дозволяють створювати правила та умови для управління криптовалютою. Також, смарт-контракти в MetaMask використовуються для того, щоб брати участь в ініціальних розподілах монет. В основному це використовується для взаємодії з децентралізованими фінансовими протоколами.

Ще один не менш популярний гаманець називається Trust Wallet, схожий функціонал у порівнянні з MetaMask, але також присутні і суттєві відмінності.

Trust Wallet (рис. 6) – це система (криптогаманець), яка надає користувачам надійний, безпечний та зручний спосіб управління криптовалютою та взаємодії з різноманітними блокчейн-мережами. Такий же простий та інтуїтивний інтерфейс, що й у MetaMask, базовий функціонал та неймовірна кількість можливостей привели гаманець до популярності серед криптовалютних поціновувачів.

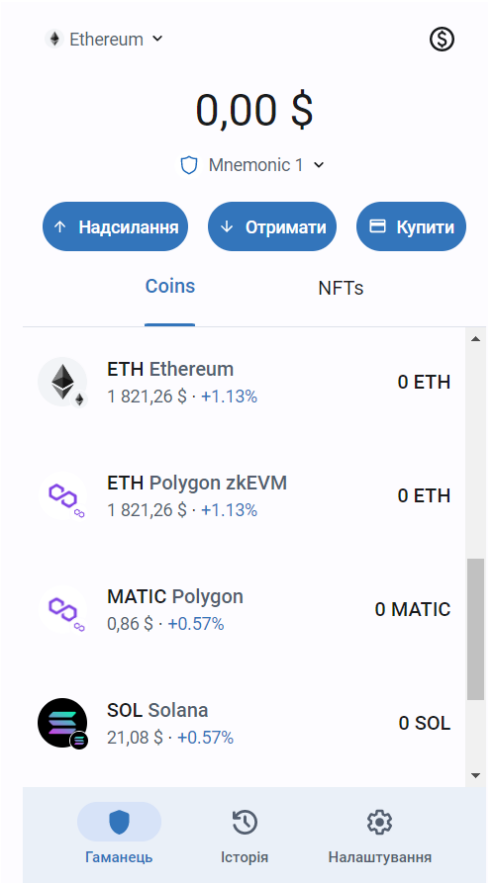

Рисунок 6 – Головна сторінка Trust Wallet

На рис. 6 можна побачити все теж саме, що й у MetaMask, де вже підключена головна мережа платформи Ethereum, можна скористатися операціями купівлі та переказу токенів. Також, помітно, що окрім токенів ETH в системі присутні й інші токени стандарту ERC20.

Оскільки гаманці мають схожі можливості та інтерфейс, то проведено аналіз для порівняння їх функціоналу, щоб було зрозуміло, для яких функцій, яка система повинна підходити краще.

Таким чином, гаманці мають однакові можливості, але не кожний може працювати з різною криптовалютою. Trust Wallet може підтримувати більшість криптовалют, коли MetaMask початково створювався під платформу Ethereum та для підтримки токенів стандарту ERC20. Зрозуміло, що зараз MetaMask здатен підтримувати й деякі інші базові мережі, наприклад, BSC (Binance Smart Chain), але цього недостатньо у порівнянні з Trust Wallet.

Ще однією особливістю є те, що система для керування криптовалютою Trust Wallet має власний токен, який має назву Trust Wallet Token (TWT). Це важливо, тому що прикладна функція TWT полягає в тому, щоб надавати користувачам гаманця, які мають на балансі певну кількість таких токенів, деяких додаткових переваг. TWT не має ніякого відношення до платформи Ethereum, оскільки токен створений на основі блокчейну BSC (Binance Smart Chain) та на базі стандарту BEP20. Як відомо, дана мережа включає стандарт токенів BEP20, який працює аналогічно визначеному стандарту ERC20.

Створення такого роду криптовалюти впроваджується для того, щоб вона брала участь у розвитку самого застосунку. Далі наведено перелік того, який функціонал може надавати такий токен.

В цілому, токен використовується у застосунку та може бути залучений до різноманітних заходів, функцій та послуг. Включаючи, меншу плату за комісію, доступ особливих функцій криптогаманця, можливість брати участь у програмах лояльності тощо.

Власники подібних токенів можуть проголосувати за різні корисні функції, що пов’язані з подальшим розвитком системи, які розробники у майбутньому будуть впроваджувати, відштовхуючись від голосування.

Спільнота, що користується токенами гаманця, може отримувати додаткову винагороду у вигляді тих самих токенів від розробників, за виявлену користувачами активність у застосунку.

Зазвичай, в подібних застосунках існує можливість надавати свою криптовалюту у стейкінг (зберігання активів з подальшим нарахуванням відсотків). Наприклад, можна покласти токени TWT на стейкінг, що дозволить отримувати пасивний дохід за зберігання своїх токенів та активність у застосунку.

З наведених прикладів прослідковується різниця MetaMask та Trust Wallet. Існує ще значна кількість привілей у створенні власного токена для криптогаманця, але було наведено одні із основних.

Не менш важливо проаналізувати блокчейн-платформи, які складають основу кожного криптовалютного гаманця. Для порівняння будемо використовувати найпопулярніші та найбільш визнані блокчейн-мережі Bitcoin та Ethereum. Bitcoin – це децентралізована електронна платіжна система, що була запропонована Сатоші Накамото у 2008 році та реалізована у 2009. Основна особливість Bitcoin полягає в тому, що вона не потребує централізованого управління, а всі транзакції з цифровим підписом між адресами розсилаються до всіх учасників мережі (peer-to-peer), а дані про пересилання біткоїнів знаходяться у скопійованій базі даних. Для забезпечення безпеки та унеможливлення витрати біткоїнів інших осіб, чи повторного використання своїх біткоїнів, використовуються криптографічні методи [1].

Порівняння основних функцій Ethereum та Bitcoin.

Початково, Bitcoin – це перша криптовалюта у світі, які була створена у вигляді цифрової валюти. Він дозволяє проводити безпечні та анонімні фінансові операції (транзакції) між усіма учасниками мережі. У теперішні дні, основним функціоналом Bitcoin являється переказ криптовалюти (BTC) у мережі.

Платформа Ethereum побудована на основі блокчейну, але на відміну від Bitcoin, основний функціонал полягає в наданні певних інструментів для виконання та розробки смарт-контрактів, що використовуються для децентралізованих застосунків. Тобто, на такій платформі можна створювати

різноманітні блокчейн-програми. Таким чином, дана мережа розроблена в більшості своїй для розробок, а не використання.

Розробка у мережах Ethereum та Bitcoin.

На платформі Bitcoin використовується мова програмування Bitcoin Script, вона має обмежений стек та використовується для визначення умов переказу токенів (використовується у блокуванні та розблокуванні транзакцій). Зазвичай для Bitcoin створюють криптовалютні гаманці та платіжні сервіси.

Мережа Ethereum використовує не одну мову програмування: Solidity, Vyper тощо. Найпопулярніша є Solidity, що використовується для розробки смарт-контрактів.

Bitcoin використовує алгоритм консенсусу Proof-of-Work (PoW), де учасники, що підтримують мережу, розв'язують складні математичні задачі для підтвердження та безпеки транзакцій, завдяки чому забезпечується децентралізація.

Ethereum використовує алгоритм консенсусу Proof-of-Stake (PoS), що зменшує енерговитрати та збільшує швидкість обробки транзакцій. Масштабованість блокчейнів Ethereum та Bitcoin також відрізняється.

Bitcoin має досить низьку швидкість для обробки транзакцій, тому що він був спроєктований як цифрова валюта, яка поступається у швидкості Ethereum. Мережа здатна виконувати приблизно 7-10 транзакцій за секунду.

Інша ситуація з мережею Ethereum, яка має перспективу більшого масштабування протоколів, може обробляти велику кількість транзакцій за секунду та розвиває свою пропускну спроможність у мережі.

Очевидно, що варіативність блокчейн-платформ підходить для реалізації різних цілей. Тож, остаточний вибір між Bitcoin та Ethereum залежить від продукту, який створюється. Bitcoin підходить для тих, кому потрібен цифровий актив та засіб платежу, тоді як Ethereum є кращим варіантом для розробників децентралізованих застосунків, розумних контрактів та інших інновацій.

Усі продукти та платформи мають багатий функціонал та добре розроблені. Помітно, що подібні системи та значна кількість існуючих блокчейн-мереж мають схожий функціонал та архітектуру, але все залежить від продукту, який необхідно створити та який буде мати свою реалізовану особливість. Наприклад, дечого не було виявлено після аналізу криптогаманців – це опція для виконання транзакцій, що не беруть плати за комісію (газ). Фактично, неможливо реалізувати таку альтернативу, тому що Ethereum не може виконувати операції без газу, але існують шляхи, де замість користувача комісію буде оплачувати дехто інший, людина до якої буде направлена транзакція для оплати газу.

1.4 Формулювання вимог та обґрунтування теми дипломного проєкту

З проаналізованої інформації попереднього підрозділу зрозуміло, якими якостями, функціоналом та особливостями повинна володіти зрозуміла, зручна, стандартна та надійна програмна система для управління криптовалютою. Надалі, детальніше описана інформація стосується виключно властивості, яка являється унікальною в подібних застосунках.

Створення програмної системи для керування власноруч випущеною криптовалютою з використанням технології блокчейн повинне включати особливі опції, які будуть забезпечувати можливість переказу криптовалюти іншому користувачеві та продаж токенів без оплати комісії (газу).

Таке рішення робить унікальним криптовалютний застосунок, тому що він зберігає єдиний власний токен та ETH, за допомогою якого можна обмінювати власну криптовалюту. Значення такого проєкту може досягати різноманітних ідей, де подібний токен може реалізовувати спеціальне призначення та використання.

Наприклад, таке може бути використане у компанії, де співробітникам виплачують зарплатню у вигляді криптовалюти. У кожної людини буде свій власний криптогаманець та акаунт, куди будуть надходити токени. Взагалі, використання застосунку можна організувати у двох версіях.

В одній версії, системою користується звичайна людина, яка не має відношення ні до якої компанії, а просто зберігає там певну кількість власних токенів. В такому випадку, програма не повинна мати можливості виконання транзакцій без комісії, тому що це немає ніякого сенсу.

В іншій версії, застосунок використовується співробітником компанії, якому виплачують зарплатню спеціальною службовою криптовалютою. Тоді, для того, щоб надати деякі привілеї своїм робітникам та заохочувати їх і надалі користуватись своїм продуктом, власники компанії можуть використовувати для цього криптогаманець, де відсутня комісія за переказ та продаж токенів.

Для того, щоб не відбувалось користувачами зловживання безкоштовними транзакціями, можна ввести їх обмежену кількість на добу, тому що систему доведеться скоріше відключити, якщо кожен буде виконувати по тисячі операцій кожного дня.

Надалі, систему можна розвивати, інтегрувати та масштабувати для більш спеціалізованих цілей та проєктів, де можна використовувати подібний підхід.

Для звичайних користувачів, що не будуть оплачувати газ за виконання транзакцій, також існує своя низка переваг.

Якщо людина часто використовує криптовалюту та виконує багато операцій, то вона буде витрачати значну частку на оплату газу, який не дешевий у мережі Ethereum.

Зрозуміло, що можливість не оплачувати комісію приваблює користувачів. Таким чином, можна швидко та надійно утворювати велику базу клієнтів, які будуть популяризувати та використовувати застосунок.

Оскільки світ криптовалют дуже великий, то існує значна кількість конкурентів, які також можуть мати свій привабливий функціонал. Тому, програма з безкоштовним переказом та продажом токенів відразу стає конкурентоспроможною, адже не кожна компанія готова піти на такі кроки для притоку користувачів, які обиратимуть вигідніший застосунок.

Детальніше про розробку цієї технології описано у наступному розділі. Також, важливо розуміти, що опція виконання транзакцій без оплати газу може мати свої недоліки.

Наприклад, можливе відчутне обмеження швидкості по обробці транзакції, тому що в заданих діях повинно виконуватись більше кроків, ніж у простішій операції з оплатою газу.

Отже, опція виконання транзакцій переказу та продажу криптовалюти, які не використовують газ, має бути вигідною для користувачів, яка забезпечуватиме економію витрат, збільшить привабливість системи для керування криптовалютою, покращить конкурентоспроможність та зручність використання.

2. РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ ДЛЯ КЕРУВАННЯ ВЛАСНОРУЧ ВИПУЩЕНОЮ КРИПТОВАЛЮТОЮ

2.1 Опис загальної архітектури системи

Описані нижче опції визначають загальну архітектуру розробки, в якій показані основні компоненти та зв'язки, що сприяють побудові всієї функціональності системи для керування власною криптовалютою.

Першочергово, система повинна забезпечувати контакт з користувачем, для такої цілі реалізовано GUI. Інтерфейс потрібен для взаємодії функцій системи і користувача. В даному застосунку до цього входить можливість створити та імпортувати акаунт, захистити свої особисті дані паролем, перевірити PK, переглянути баланс токенів, здійснити купівлю, переказ та продаж криптовалюти, оплачуючи комісію, та останні дві операції без комісії.

Ще одним основним компонентом усієї системи є смарт-контракт власного випущеного токена. Він працює з усіма компонентами, оскільки смарт-контракт фактично і є токеном та в ньому закладено базову функціональність керування криптовалютою на стандарті ERC20, яка є доступною на виконання для користувача з GUI.

Веб-сервер виконує роль посередника між Ethereum блокчейном та GUI для виконання криптовалютних операцій. Працює це таким чином, що сервер отримує запит від клієнта, запускає на виконання відповідну свою функцію, яка налагоджує деякі механізми та необхідні властивості, далі звертається до розгорнутого у мережі смарт-контракту та повертає результати користувачу.

Основою усієї системи виступає Ethereum блокчейн. Дана технологія забезпечує захист, децентралізацію та зберігання всіх виконаних транзакцій та операцій у блокчейні.

Таким чином, з усієї описаної структури складається розроблена система для керування власноруч випущеною криптовалютою.

2.2 Розробка смарт-контракту токена на базі стандарту ERC20

У цьому розділі описано процес написання смарт-контракту токена на базі стандарту ERC20, який визначає функції та методи для взаємодії з токенами, використовуючи блокчейн Ethereum.

Вся логіка описаного токена представлена в одному *BDPToken.sol* файлі. На рис. 7 показано імпорт необхідних контрактів та бібліотек, функції яких використовуються в основному коді [18].

```
import { SafeMath } from "../node_modules/@openzeppelin/contracts/utils/math/SafeMath.sol";
import { Address } from "../node_modules/@openzeppelin/contracts/utils/Address.sol";

import { IERC20Internal } from "./EIP-3009/IERC20Internal.sol";
import { EIP3009 } from "./EIP-3009/EIP3009.sol";
import { EIP2612 } from "./EIP-3009/EIP2612.sol";
import { EIP712 } from "./EIP-3009/EIP712.sol";

import { Ownable } from "./Ownable.sol";
```

Рисунок 7 – Імпорт контрактів та бібліотек

Нижче наведені пояснення щодо кожного контракту:

- *SafeMath.sol*: забезпечує базові та безпечні математичні операції, а саме: додавання, віднімання, ділення, множення. Методи використовуються для запобіганню переповнень та від'ємних значень;
- *Address.sol*: використовується для взаємодії зі спеціальними адресами Ethereum, надає можливість перевірити адресу на існування, переробити її у інший вигляд тощо;
- *IERC20Internal.sol*: представляє собою інтерфейс, який визначає стандартні методи для внутрішнього використання у контракті токена на базі ERC20;
- *EIP3009.sol*: реалізовує стандарт ERC3009, функціональність якого використовується для захищених переказів токенів з використанням дозволу або відхилення транзакцій;

- *EIP2612.sol*: даний стандарт використовується для підтвердження (підпису) транзакцій;
- *EIP712.sol*: смарт-контракт надає можливість створення та перевірки доменного роздільника, який задіяний для підписання транзакцій;
- *Ownable.sol*: використовується для визначення власника токена (контракту), надає методи для виконання операцій, які доступні тільки для власника.

Далі продемонстровано конструктор для смарт-контракту створюваного токена на рис. 8, він викликається у момент, коли контракт розгортається у мережі Ethereum [18].

```
constructor(
    string memory tokenName,
    string memory tokenVersion,
    string memory tokenSymbol,
    uint8 tokenDecimals,
    uint256 tokenTotalSupply
){
    _name = tokenName;
    _version = tokenVersion;
    _symbol = tokenSymbol;
    _decimals = tokenDecimals;

    DOMAIN_SEPARATOR = EIP712.makeDomainSeparator(tokenName, tokenVersion);
    _owner = msg.sender;
    _mint(_owner, tokenTotalSupply);
}
```

Рисунок 8 – Конструктор для смарт-контракту токена

З коду зрозуміло, що конструктор необхідний, щоб встановити основні параметри токена (ім'я, символ, версію десяткові знаки, загальний обсяг) та визначити власника контракту.

Важливим є те, що за допомогою методу *EIP712.makeDomainSeparator* створюється *DOMAIN_SEPARATOR*, який необхідний для реалізації стандарту EIP712, який задіяний у безпечному підписі транзакцій.

Наступний код описує реалізацію важливих функцій, які необхідні для базового токена стандарту ERC20.

На рис. 9 показана функція *_transfer*, яка виконує переказ токенів між користувачами [18].

Спочатку в методі перевіряється, щоб ні один із рахунків *sender* та *recipient* не являвся нульовим адресом. Якщо рахунки є дійсними, то зменшується баланс *sender* (відправника) та збільшується рахунок *recipient* (отримувача) на задану кількість токенів *amount*. В кінці викликається подія, яка вказує на те, що все пройшло вдало.

```
function _transfer(
    address sender,
    address recipient,
    uint256 amount
) internal override {
    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");

    _balances[sender] = _balances[sender].sub(
        amount,
        "ERC20: transfer amount exceeds balance"
    );
    _balances[recipient] = _balances[recipient].add(amount);
    emit Transfer(sender, recipient, amount);
}
```

Рисунок 9 – Функція *_transfer*

Наступний код (рис. 10) описує функцію *balanceOf*, яка повертає баланс заданого рахунку [18].

```
function balanceOf(address account) external view returns (uint256) {
    return _balances[account];
}
```

Рисунок 10 – Функція *balanceOf*

Подальший метод *_approve* на рис. 11 описує встановлення дозволу на переказ токенів якомусь іншому акаунту [18].

Тіло методу починається з перевірки адресів на те, що вони не являються нульовими. Далі встановлюється дозвіл на переказ певної кількості *amount* (токенів) з *owner* (власника) до *spender* (отримувача), зберігання заданих дозволів виконується в мапі *_allowances*, де ключами являються адреси власників, а значеннями є мапи, в яких ключами являються адреси отримувачів, а значеннями є дозволена на переказ кількість токенів. В кінці знову викликається подія, що сигналізує про успішне завершення операції.

```
function _approve(
    address owner,
    address spender,
    uint256 amount
) internal override {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
}
```

Рисунок 11 – Метод *_approve*

Наведений нижче метод *transferFrom* на рис. 12 використовується для передачі токенів з одного рахунку на інший за допомогою заздалегідь сформованих дозволів [18].

В даному випадку виконується виклик функцій, які вже були описані вище. Спочатку *_transfer* для переказу токенів, потім встановлюється нове значення дозволу для відправника (*sender*) та поточного виконавця (*msg.sender*). Кількість дозволених токенів визначається через віднімання переданої кількості токенів (*amount*) зі значення дозволених, далі викликається функція *_approve*, яка повинна занести новий дозвіл для залишку.

```
function transferFrom(
    address sender,
    address recipient,
    uint256 amount
) external returns (bool) {
    _transfer(sender, recipient, amount);
    _approve(
        sender,
        msg.sender,
        _allowances[sender][msg.sender].sub(
            amount,
            "ERC20: transfer amount exceeds allowance"
        )
    );
    return true;
}
```

Рисунок 12 – Метод *transferFrom*

Весь описаний код надає необхідну логіку для пересилання токенів, перегляду балансу та встановлення дозволів на переказ токенів.

2.3 Реалізація операції обміну власних токенів на Ether та навпаки

В даному підрозділі фактично описано функції купівлі та продажу власного випущеного токена у мережі Ethereum. Реалізовано це таким чином, що існують дві окремі функції обміну власної криптовалюти на токени мережі та навпаки. Така розробка в системі забезпечує зручний та безпечний механізм конвертації між двома активами, де ETH являється головним, універсальним та використовуваним токеном мережі, що дозволяє користувачам ефективно керувати своєю криптовалютою та здійснювати обмін відповідно до їхніх потреб.

Для початку, необхідно мати два методи, де один із них повертає поточну ціну токена, а інший встановлює нову. Дані методи можна побачити на рис. 13, вони мають назву *getPricePerBDP* та *setPricePerBDP*. Важливо, що в методу *setPricePerBDP* встановлений модифікатор *onlyOwner*, це означає, що заданий код може викликати тільки власник контракту (токена).

```
function getPricePerBDP() external view returns(uint256) {
    return pricePerBDP;
}

function setPricePerBDP(uint256 newPricePerBDP) external payable onlyOwner {
    require(newPricePerBDP != 0, "BDPToken.setPricePerBDP: Price must be more than zero!");
    pricePerBDP = newPricePerBDP;
}
```

Рисунок 13 – Методи отримання та встановлення ціни токена

Тепер, треба описати функцію, яка дозволяє купувати власні випущені токени у системі, вона наведена на рис. 14 та має свої особливі характеристики.

```
function buy(uint256 amountToBuy) external payable {
    uint256 amountEtherInWei = msg.value;
    require(amountToBuy <= this.balanceOf(_owner),
        "BDPToken.buy: Don't have enought tokens in reserve");
    require(amountToBuy >= (10**this.decimals() ).div(pricePerBDP),
        "BDPToken.buy: Purchase amount is less than 1 wei ");
    uint256 TotalCost = amountToBuy.mul(pricePerBDP);
    TotalCost = TotalCost.div(10**18);
    require(amountEtherInWei >= TotalCost,
        "BDPToken.buy: Don't received enough ether to buy requested amount of BDP");
    _transfer(_owner, msg.sender, amountToBuy);
    if(amountEtherInWei > TotalCost) {
        uint256 refund = amountEtherInWei.sub(TotalCost);
        payable(msg.sender).transfer(refund);
    }
    emit BuyToken(msg.sender, amountToBuy, pricePerBDP);
}
```

Рисунок 14 –Алгоритм купівлі токенів

Нижче наведено короткий опис коду.

Змінна *amountEtherInWei* зберігає Ether у значенні wei (*msg.value*), який надійшов з транзакцією від відправника (*msg.sender*), як оплата за токени.

Наступна перевірка визначає, що власник контракту (*_owner*) зберігає достатню кількість токенів на своєму акаунті для продажу.

Ще одна вимога перевіряє *amountToBuy* (токени для купівлі) на валідність, щоб запрошена кількість на купівлю була рівною, як мінімум, одному wei.

Далі обчислюється вся вартість токенів до змінної *TotalCost* для купівлі в значенні wei. Відбувається це шляхом множення всієї кількості на ціну одного токена системи (*pricePerBDP*), а потім виконується операція коригування вартості wei до ETH діленням на 10^{18}, тому що один ETH складається з 18 десяткових знаків.

Наступна перевірка визначає, що для виконання операції було передано достатню кількість *amountEtherInWei* для придбання токенів.

Далі використовується вже відома функція *_transfer*, яка переказує зазначену кількість токенів (*amountToBuy*) від власника контракту (*_owner*) на адресу з якої надійшов запит на купівлю токенів (*msg.sender*).

Кінцева умова необхідна для того, щоб перевірити, чи не було надано більшу кількість ETH у транзакцію, ніж повинно було. Якщо так сталось, то на адресу відправника буде надіслано назад вирахувану різницю (*refund*).

В кінці викликається подія (*BuyToken*), яка реєструє у блокчейні купівлю токенів, передаючи адресу покупця (*msg.sender*), кількість токенів (*amountToBuy*) та ціну за токен (*pricePerBDP*).

Треба зазначити, що ETH, який був надісланий з транзакцією для купівлі токенів, тепер зберігається на рахунку контракту.

Наступною є функція продажу власних випущених токенів (рис. 15), вона дещо схожа на вже описану по купівлі, але має свої особливі відмінності.

Перша вимога необхідна дня того, щоб упевнитись, що на рахунку відправника транзакції (*msg.sender*) достатньо токенів для продажу (*amountToSell*).

Наступна використовується, щоб перевірити *amountToSell* (токени для продажу) на валідність, вони не повинні мати значення меншого від найменшої одиниці (*pricePerBDP*).

```
function sell(uint256 amountToSell) external payable {
    require(this.balanceOf(msg.sender) >= amountToSell,
        "BDPToken.sell: You don't have enought tokens.");
    require(amountToSell >= (10**this.decimals()).div(pricePerBDP),
        "BDPToken.sell: So little sum to sell!");
    uint256 _allowance = this.allowance(msg.sender, address(this));
    uint256 weiAmount = amountToSell.mul(pricePerBDP);
    weiAmount = weiAmount.div(10**this.decimals());
    require(_allowance >= amountToSell,
        "BDPToken.sell: Check the token allowance");
    require(address(this).balance >= weiAmount,
        "BDPToken.sell: Contract haven't enought ethereum. Try call function later!");
    this.transferFrom(msg.sender, _owner, amountToSell);
    payable(msg.sender).transfer(weiAmount);
    emit SellToken(msg.sender, amountToSell, pricePerBDP);
}
```

Рисунок 15 – Алгоритм продажу токенів

На подальшому кроці в змінну *_allowance* заноситься значення дозволу від власника токенів (*msg.sender*) на передачу токенів до контракту (*address(this)*).

Далі розраховується вартість суми токенів для продажу, яка буде переказана на рахунок контракту, вона повинна бути збережена у змінній *weiAmount*. Відбувається це шляхом множення всієї кількості на ціну одного токена системи (*pricePerBDP*), а потім здійснюється коригування вартості до відповідної величини діленням на $10^{decimals}$, тому що один токен системи складається з певної кількості *decimals* (десяткові знаки).

Наступним кроком перевіряється, чи є значення дозволених на переказ токенів (*_allowance*) смарт-контрактом з рахунку відправника не меншим за суму, яку необхідно обміняти (*amountToSell*).

Ще одна вимога перевіряє те, щоб на рахунку смарт-контракту (*address(this).balance*) знаходилось достатньо ETH за суму (*weiAmount*), яку необхідно виплатити користувачу за токени.

Використовуючи вже описаний метод (*transferFrom*), відбувається переказ токенів (*amountToSell*) від відправника транзакції (*msg.sender*) на рахунок власника контракту (*_owner*).

Для завершення операцій, необхідно заплатити ETH за куплені токени. Тому, відправник транзакції (*msg.sender*) виконує переказ визначеної суми за токени (*weiAmount*) на адресу смарт-контракту.

Остання дія (*SellToken*) використовується для реєстрації у блокчейні продажу токенів, передаючи адресу продавця (*msg.sender*), кількість проданих токенів (*amountToBuy*) та ціну за токен (*pricePerBDP*).

2.4 Розробка операцій переказу та продажу, які не стягують комісії

Основною ідеєю програмної системи є виконання транзакцій переказу та продажу токенів без оплати газу. Для такої задачі було розроблено методи *gassLessSell* та *gassLessTransfer* з використанням авторизації. Останній зображено на рис. 16.

```
function gassLessTransfer(
    address from,
    address to,
    uint256 value,
    uint256 validAfter,
    uint256 validBefore,
    bytes32 nonce,
    uint8 v,
    bytes32 r,
    bytes32 s)
external {
    require(this.balanceOf(from) >= value,
        "BDPToken.gassLessTransfer: User from don't have enought BDP on account!");
    transferWithAuthorization(from, to, value, validAfter, validBefore, nonce, v, r, s);
    emit Transfer(from, to, value);
}
```

Рисунок 16 – Код методу *gassLessTransfer*

Метод приймає такі параметри:

- from: адреса з якої переказуються токени;
- to: адреса на яку надсилаються токени;
- value: сумарна кількість токенів для переказу;

- validAfter: час з якого починається період дійсності авторизації;
- validBefore: час з якого закінчується період дійсності авторизації;
- nonce: ID транзакції;
- v, r, s: необхідні параметри підпису для підтвердження авторизації.

Якщо описувати кожний рядок тіла методу, то спочатку перевіряється наявність на рахунку відправника (*this.balanceOf(from)*) достатньої кількості токенів (*value*).

Основною є функція *transferFromWithAuthorization*, яка дозволяє переказати токени без оплати газу відправнику, використовуючи авторизаційні параметри, замість нього газ сплатить власник контракту. Функція виконується так, що проводить перевірку часу дійсності авторизації, перевіряє підпис та обробляє переказ токенів з адреси (*from*) на адресу (*to*).

Генерується подія *Transfer* для реєстрації у блокчейні виконаного переказу токенів, передаючи адресу відправника (*from*), адресу отримувача (*to*) та кількість токенів (*value*).

На рис. 17 представлено функцію *gassLessSell*, яка використовує метод авторизації для виконання транзакції. Вона має ідентичні вхідні параметри, що й *gassLessTransfer.* Якщо порівнювати цю функцію із вже описаною *sell*, то помітно, що вони схожі, але метод *gassLessSell* має свої особливі відмінності, які описані нижче.

Спочатку йде перевірка на те, що адреса користувача (*_to*), яка надійшла зовні, дорівнює адресі власника контракту (*_owner*), на яку повинні бути переказані токени, що продаються.

Подальший функціонал ідентичний функції *sell,* але в даному випадку відсутня змінна *_allowance,* яка визначає значення дозволених на переказ токенів смарт-контрактом з рахунку відправника. Вона не є необхідною, оскільки додаткову безпеку визначає функція *transferFromWithAuthorization,* яка також виконує однакову роль, що й у функції *gassLessTransfer.*

```
function gassLessSell(address from, address to, uint256 value, uint256 validAfter,
    uint256 validBefore, bytes32 nonce, uint8 v, bytes32 r, bytes32 s)
external {
    require(to == _owner,
        "BDPToken.gassLessSell: Signed transaction must be sent to owner address");
    require(this.balanceOf(from) >= value,
        "BDPToken.gassLessSell: You don't have enought tokens.");
    require(value >= (10**this.decimals()).div(pricePerBDP),
        "BDPToken.gassLessSell: So little sum to sell!");
    uint256 weiAmount = value.mul(pricePerBDP);
    weiAmount = weiAmount.div(10** this.decimals());
    require(address(this).balance >= weiAmount,
        "BDPToken.gassLessSell: Contract haven't enought ethereum. Try call function later!");
    transferWithAuthorization(from, _owner, value, validAfter, validBefore, nonce, v, r, s);
    payable(from).transfer(weiAmount);
    emit SellToken(msg.sender, value, pricePerBDP);
}
```

Рисунок 17 – Код методу *gassLessSell*

2.5 Реалізація налаштувань застосунку, використовуючи Qt framework

В даному підрозділі розглянуто реалізацію функцій застосунку для керування власноруч випущеною криптовалютою на основі Qt framework, який являється потужним інструментом для написання кросплатформних застосунків з використанням C++ та QML (Qt Meta-Object Language) [16].

QML є декларативною мовою, що використовується в Qt framework для написання GUI. Вона є однією з основних мов, які можуть бути використані для розробки застосунків у Qt framework.

Деяку основну частину коду на Qt framework описано далі. На рис. 18 зображено імпорт необхідних модулів для розробки [16]:

- *QtQuick* та *QtQuick.Controls* є модулями для написання UI;
- *QtWebSockets* являється модулем для взаємодії з веб-сокетами;
- *AccountHandler* містить функціонал для обробки облікових записів;
- *Qaterial* є бібліотекою створення матеріального дизайну інтерфейсу [14];
- *Abi* являється модулем, що містить ABI для взаємодії зі смарт-контрактом;
- *Web3* – це модуль, що надає інтерфейс для роботи з технологією блокчейн.

```
import QtQuick 2.15
import QtQuick.Controls 2.15
import QtWebSockets 1.15
import AccountHandler 1.0
import Qaterial 1.0 as Qaterial
import "abi.js" as Abi
import "web3.js" as Web3
```

Рисунок 18 – Імпорт модулів

На наступному коді (рис. 19) створюється об'єкт вікна застосунку та ініціалізуються основні змінні для взаємодії з контрактом.

```
Qaterial.ApplicationWindow {
    id: window
    width: 800
    height: 600
    visible: true
    title: qsTr("Bakalavra diploma project")
    readonly property string _BDP_TOKEN_ADDRESS_: "0x97e47E3Bc5C0BFc1423dCb7d6c3dA713f2C5f33f"
    readonly property var _BDP_TOKEN_ABI_: Abi.ABI
    property string _WAITING_TEXT_: "Transaction was sent. Waiting for confirmation..."
```

Рисунок 19 – Вікно застосунку та змінні

Даний код створює об’єкт головного вікна системи, використовуючи клас *Qaterial.ApplicationWindow* з бібліотеки *Qaterial*. Вікно має певні поля, а саме: *id*, *width*, *height*, *visible*, *title* [14].

Ще нижче описано змінні, які мають наступні властивості:

- *_BDP_TOKEN_ADDRESS_* містить адресу контракту для взаємодії з ним;
- *_BDP_TOKEN_ABI_* вміщає ABI для взаємодії з контрактом.

У наступних рядках (рис. 20) ініціалізуються *_BDP_TOKEN_CONTRACT_* та *web3* змінні. Остання створює новий екземпляр об’єкту *Web3.Web3* та привласнює йому URL вузла блокчейну для під’єднання.

Властивості *account_handler* задається шлях до директорії, де знаходяться облікові записи користувачів системи. Кожний користувач у своєму файлі повинен мати логін, зашифрований PK та пароль (необов’язково).

Змінній *_BDP_TOKEN_CONTRACT_* привласнюється новий екземпляр контракту токена, використовуючи *web3.eth.Contract* з такими аргументами, як адреса контракту (*_BDP_TOKEN_ADDRESS_*) та ABI (*_BDP_TOKEN_ABI_*). Також, змінна встановлює *defaultChain* (ланцюжок за замовчуванням) у значенні sepolia (тестова мережа Ethereum) [12].

```
web3 = new Web3.Web3("https://sepolia.infura.io/v3/184884ee16e34afab71bf6c655e81075");
account_handler.setAccountsPath("C:/Users/lnew7/Desktop/wallet/wallet_files/accounts")
_BDP_TOKEN_CONTRACT_ = new web3.eth.Contract(_BDP_TOKEN_ABI_, _BDP_TOKEN_ADDRESS_)
_BDP_TOKEN_CONTRACT_.defaultChain = "sepolia"
```

Рисунок 20 – Налаштування мережі

Наступний приклад коду містить три функції та об'єкт *Timer*, вони використовуються для динамічного відображення балансу ETH та токена BDP (власний випущений токен системи) у застосунку.

Перша функція *get_eth_balance* (рис. 21), її вхідним параметром є кількість ETH на рахунку користувача у значенні wei (*wei_balance*). Далі сума у wei конвертується до значення в номіналі ETH та округлюється до 5 знаків після коми (*_eth_balance_*). Змінній *_ETH_AMOUNT_* привласнюється *_eth_balance_* після парсингу у тип float. *_ETH_AMOUNT_* буде використовуватись у подальших операціях з криптовалютою, а *_eth_balance_* виводить на екран інформацію щодо балансу ETH користувача.

```
function get_eth_balance(wei_balance) {
    var _eth_balance_=(wei_balance/1000000000000000000).toFixed(5)
    _ETH_AMOUNT_=parseFloat(_eth_balance_)
    eth_balance.text=_eth_balance_+" ETH"
}
```

Рисунок 21 – Вивід балансу ETH

Друга функція *get_bdp_balance* (рис. 22) працює аналогічно, як попередня.

```
function get_bdp_balance(wei_balance) {
    var _bdp_balance_=(wei_balance/1000000000000000000).toFixed(5)
    _BDP_AMOUNT_=parseFloat(_bdp_balance_)
    bdp_balance.text=_bdp_balance_+" BDP"
}
```

Рисунок 22 – Вивід балансу BDP

Третя функція *get_cryptowell* (рис. 23), її функціонал схожий на попередні, але має свою відмінність. Призначення полягає в обробці курсу конвертації ETH до BDP. Отримана вхідним параметром вартість у wei (*wei_currency*) перетворюється в суму в значенні BDP та округлюється до 5 знаків після коми.

```
function get_cryptowell(wei_currency) {
    var cryptowell = (1000000000000000000/wei_currency).toFixed(5)
    _BDP_CURRENCY_=parseFloat(cryptowell)
    bdp_currency.text="1 ETH = "+cryptowell+" BDP"
}
```

Рисунок 23 – Вивід курсу обміну ETH до BDP

Останній, об’єкт *Timer* (рис. 24) працює з усіма трьома описаними функціями та отримує значення їхніх балансів.

```
Timer {
    id: showing_balances
    running: menu.visible
    repeat: true
    interval: 3000
    onTriggered: () ⇒  {
        web3.eth.getBalance(_ACCOUNT_.address).then(get_eth_balance)
        _BDP_TOKEN_CONTRACT_.methods.balanceOf(_ACCOUNT_.address).call().then(get_bdp_balance)
        _BDP_TOKEN_CONTRACT_.methods.getPricePerBDP().call().then(get_cryptowell)
    }
}
```

Рисунок 24 – Отримання балансів

Timer виконується з інтервалом в 3 секунди та запускає функцію *onTriggered*. Вже в тілі цього блоку виконуються асинхронні запити на

отримування балансу ETH, BDP та курсу конвертації ETH до BDP, використовуючи *_BDP_TOKEN_CONTRACT_.methods* (прописані методи контракту токена) та *web3.eth.getBalance* (метод отримання балансу з модуля web3) [12]. Далі всі отримані значення передаються до відповідних функцій, які були описані вище.

2.6 Розробка криптовалютних операцій та взаємодія з веб-сервером

Даний підрозділ розкриває деталі функцій операцій з криптовалютою у взаємодії з веб-сервером. Для такої задачі було використано WebSocket протокол. На рис. 25 розглянуто код об'єкту *WebSocket*, який задіяний для з'єднання з сервером, використовуючи адресу *ws://localhost:1011*. При отриманні відповіді (*onTextMessageReceived*) у вигляді повідомлення (*message*) виконується функція, яка активує заблоковане меню та викликає функцію *define_option(message)*.

```
WebSocket {
    id: wallet_web_socket
    active: true
    url: "ws://localhost:1011"
    onTextMessageReceived: (message) ⇒ {
        menu.enabled = true
        define_option(message)
    }
}
```

Рисунок 25 – Об'єкт *WebSocket*

Далі наведено функцію *define_option* (рис. 26), яка приймає параметр *message* для обробки відповідного повідомлення з веб-сервера В тілі задіяна змінна *current_option*, що визначає, яка з трьох визначених операцій була оброблена (*transferring_address*, *buying_amount*, *selling_amount*). Після цього перевіряється *message* на помилку та встановлюється відповідний текст до змінної *current_option*, який виводиться на екрані після виконання операції.

```
function define_option(message) {
    var current_option
    switch (listView.currentIndex) {
        case 1: current_option = transferring_address; break;
        case 2: current_option = buying_amount; break;
        case 3: current_option = selling_amount; break;
    }
    if (message === "DONE")
        current_option.helperText = "Transaction was successfull!"
    else if (message === "ERROR") {
        current_option.helperText = ""
        current_option.setError("Something went wrong!")
    }
}
```

Рисунок 26 – Функція *define_option*

Подальший код описує функції, які необхідні для виконання операцій з криптовалютою та взаємодії з веб-сервером.

Функція *sell* (рис. 27) використовується для продажу BDP токенів. Параметри, які надходять у функцію визначають кількість токенів BDP для продажу (*amount*) та PK користувача (*accountPrivateKey*).

Суть функції полягає у тому, щоб надіслати запит до сервера на обробку заданих даних (*wallet_web_socket.sendTextMessage*). В повідомленні знаходяться такі дані: команда, яка вказує на продаж токенів (*sell*), PK користувача (*accountPrivateKey*) та сума токенів для продажу (*parseFloat(amount)*).

Після того, як повідомлення було надіслано на сервер, змінній *selling_amount* привласнюється значення змінної *_WAITING_TEXT_* (Transaction was sent. Waiting for confirmation...) та функція блокує меню, що унеможливлює подальше користування застосунком, поки не буде оброблена транзакція.

```
function sell(accountPrivateKey, amount) {
    wallet_web_socket.sendTextMessage('{"command":"sell","accountPrivateKey":"'+
                                      accountPrivateKey+'","sum":"'+parseFloat(amount)+'"}')
    selling_amount.helperText = _WAITING_TEXT_
    menu.enabled = false
}
```

Рисунок 27 – Операція продажу токенів

Функція *buy* (рис. 28) працює аналогічним чином, що й функція *sell*. Єдина відмінність в тому, що в цьому випадку надсилається команда (*buy)* та змінній *buying_amount* привласнюється значення *_WAITING_TEXT_* у повідомленні.

```
function buy(accountPrivateKey, amount) {
    wallet_web_socket.sendTextMessage('{"command":"buy","accountPrivateKey":"'+
                                      accountPrivateKey+'","sum":"'+parseFloat(amount)+'"}')
    buying_amount.helperText = _WAITING_TEXT_
    menu.enabled = false
}
```

Рисунок 28 – Операція купівлі токенів

Функція *gasLessSell* (рис. 29) також працює аналогічно, що й функція *sell,* з різницею, де команда до сервера має значення (*gasLessSell)*.

```
function gasLessSell(accountPrivateKey, amount) {
    wallet_web_socket.sendTextMessage('{"command":"gasLessSell","accountPrivateKey":"'+
                                      accountPrivateKey+'","sum":'+parseFloat(amount)+'}')
    selling_amount.helperText = _WAITING_TEXT_
    menu.enabled = false
}
```

Рисунок 29 – Операція продажу токенів без газу

Функція *transfer* (рис. 30) використовується для переказу BDP токенів та має свої відповідні параметри: PK акаунту (*accountPrivateKey*), адреса на яку переказуються токени (*addressTo*) та загальна кількість для переказу (*amount*).

Спочатку виконується перевірка, використовуючи об’єкт *web3*, що параметр *addressTo* являється валідним адресом у технології блокчейн.

Наступним кроком формується запит до сервера, який включає в себе такі дані: команду (*transfer*), PK (*accountPrivateKey*), адресу отримувача (*addressTo*) та суму токенів (*amount*).

У наступній частині, змінній *transferring_address* привласнюється значення *_WAITING_TEXT_* та блокується меню застосунку.

```
function transfer(accountPrivateKey, addressTo, amount) {
    if (!web3.utils.isAddress(addressTo)) {
        transferring_address.setError("Such address does not exist!")
        return
    }
    wallet_web_socket.sendTextMessage('{"command":"transfer","accountPrivateKey":"'+accountPrivateKey+
                                      '","addressTo":"'+addressTo+'","sum":'+parseFloat(amount)+'}')
    transferring_address.helperText = _WAITING_TEXT_
    menu.enabled = false
}
```

Рисунок 30 – Операція переказу токенів

Функція *gasLessTransfer* (рис. 31) працює ідентично, що й функція *transfer* з відмінністю в тому, що команда до сервера має значення (*gasLessTransfer)*.

```
function gasLessTransfer(accountPrivateKey, addressTo, amount) {
    if (!web3.utils.isAddress(addressTo)) {
        transferring_address.setError("Such address does not exist!")
        return
    }
    wallet_web_socket.sendTextMessage('{"command":"gasLessTransfer","accountPrivateKey":"'+accountPrivateKey+
                                      '","addressTo":"'+addressTo+'","sum":'+parseFloat(amount)+'}')
    transferring_address.helperText = _WAITING_TEXT_
    menu.enabled = false
}
```

Рисунок 31 – Операція переказу токенів без газу

2.7 Обробка операцій з токеном на веб-сервері за допомогою Node.js

Наступною частиною є розробка функціональності для обробки операцій з токеном на веб-сервері. Для здійснення таких операцій використовується програмний кросплатформний інструмент Node.js, який взаємодіє з технологією блокчейн Ethereum [17].

Наведений код на рис. 32 виконує необхідний імпорт бібліотек та модулів, здійснює налаштування для взаємодії з мережею Ethereum та смарт-контрактом власного випущеного токена BDP.

Описання кожного імпортованого модуля [19]:

- WebSocketServer: модуль для створення веб-сервера через веб-сокети;
- Web3: модуль, який виконує взаємодію з технологією блокчейн [12];

- ethers: модуль для взаємодії з платформою та мережею Ethereum [11];
- BigNumber: модуль для обробки чисел великої точності в Ethereum;
- BDPArtifacts: JSON-файл, який містить ABI смарт-контракту BDP токена;
- ethJsUtil: модуль для роботи з підписами транзакцій Ethereum;
- dotenv: модуль для використання змінних середовища з *.env* файлу.

```
let WebSocketServer = require('ws')
let {ethers, BigNumber} = require("ethers");
const BDPArtifacts = require('../build/contracts/BDPToken.json');
var  ethJsUtil = require( "../node_modules/ethereumjs-util");
const Web3 = require("../node_modules/web3")
require("../node_modules/dotenv").config();
const web3 = new Web3();
const abi = BDPArtifacts.abi;
const network = process.env.ETHEREUM_NETWORK;
const provider = new ethers.providers.JsonRpcProvider(process.env.INFURA_ENDPOINT);
const addressContract = process.env.CONTRACT_ADDRESS;
```

Рисунок 32 – Імпорт модулів та встановлення необхідних налаштувань

Наступним кроком створюється екземпляр *web3*, отримується значення *abi* (ABI*)* токена BDP (*BDPArtifacts.abi*), заноситься адреса контракту зі змінної середовища (*CONTRACT_ADDRESS*) до константи *addressContract* та відбувається налаштування мережі (*network*) та провайдера (*provider*). Константі *network* привласнюється назва тестової мережі Ethereum (sepolia), а константа *provider* створює JSON RPC провайдер, який з'єднується з Infura (вузол), використовуючи змінну середовища *INFURA_ENDPOINT* (URL вузол блокчейну).

На черзі код, який описує важливу функцію *signTransferAuthorization*, вона використовується для підпису авторизованої транзакції в операціях, де користувач не оплачує газ. Функція зображена на рис. 33 та повертає підписану авторизовану транзакцію [18].

```
function signTransferAuthorization(from, to, value, validAfter, validBefore,
                                   nonce, domainSeparator, privateKey) {
    return signEIP712(
        domainSeparator,
        TRANSFER_WITH_AUTHORIZATION_TYPEHASH,
        ["address", "address", "uint256", "uint256", "uint256", "bytes32"],
        [from, to, value, validAfter, validBefore, nonce],
        privateKey
    );
}
```

Рисунок 33 – Функція *signTransferAuthorization*

Подальший функціонал (рис. 34) створює об'єкт сервера *wss*, використовуючи модуль WebSocketServer, де значення порту (*port*) дорівнює 1011, на якому сервер буде прослуховувати з'єднання.

```
const wss = new WebSocketServer.WebSocketServer({port: 1011});
wss.on("connection", (ws) => {
    console.log("New client connected!");
    ws.on("message", async(data) => {
        console.log("Sending...")
        let request = JSON.parse(data)
        var response;
        switch (request.command) {
            case 'transfer': response = await transfer(request.accountPrivateKey,
                request.addressTo, request.sum); break;
            case 'buy': response = await buy(request.accountPrivateKey, request.sum); break;
            case 'sell': response = await sell(request.accountPrivateKey, request.sum); break;
            case 'gasLessSell': response = await gasLessSell(request.accountPrivateKey, request.sum); break;
            case 'gasLessTransfer': response = await gasLessTransfer(request.accountPrivateKey,
                request.addressTo, request.sum); break;
        }
        ws.send(response);
    });
    ws.on("close", () => {
        console.log("The client disconnected!");
    });
    ws.onerror = () => {
        console.log("Some error occurred!")
    }
});
console.log("The WebSocket server is running on port 1011");
```

Рисунок 34 – Алгоритм взаємодії з *WebSocketServer*

Обробка подій сервера починається з привітання нового клієнта (*"New client connected!"*), який був під'єднаний. Після цього сервер налаштовується на

очікування отримування повідомлень (*message*), закриття з'єднання з клієнтом (*close*) та обробку помилок (*onerror*).

Якщо аналізувати функцію обробки повідомлення, яка отримує дані від клієнта (*"message", async(data)*), то спочатку до змінної *request* парситься отримане повідомлення (*data*) у JSON форматі. Потім конструкція *switch* перебирає значення властивості *request.command* та дає на виконання відповідну функцію, таку як *transfer*, *buy*, *sell*, *gasLessSell, або gasLessTransfer*, передаючи туди інші необхідні аргументи, які включаються до поля *request.* Всі результати виконання відповідної функції зберігаються до змінної *response*, яку сервер надсилає у відповідь клієнту (*ws.send(response)*).

Нижче наведено функцію виконання операції продажу криптовалюти на сервері, на якій проілюстровано та роз'яснено основні вимоги написання операцій, що взаємодіють з блокчейном Ethereum, а обсяг усіх функцій можна переглянути у додатку з повним кодом.

Код на рис. 35 представляє асинхронну функцію *sell*, яка призначена для продажу BDP токенів у системі.

Функція приймає такі параметри, як PK користувача, що хоче продати токени (*signerPrivateKey*) та кількість токенів (*amountToSell*). Описуючи коротко, спочатку перевіряється значення *amountToSell,* переводиться у відповідний формат номіналу wei (*ethers.utils.parseUnits(amountToSell.toString(), "ether")*) та привласнюється даній змінній.

Далі створюється (*ethers.Wallet*) необхідний об'єкт акаунту користувача (*signer*) за допомогою його PK (*signerPrivateKey*) та провайдеру (*provider*) [11].

Обов'язково треба отримати значення кількості транзакцій, які вже були виконанні користувачем (*nonceBase*) для того, щоб можна було виконати функцію *getNonce*, що повертає кількість транзакцій, збільшуючи значення (*nonce*) з кожним викликом. Необхідність такої функції полягає в тому, що треба задавати поле (*nonce)* при надсиланні транзакції у мережу.

```
const sell = async(signerPrivateKey, amountToSell) => {
    try {
        amountToSell = ethers.utils.parseUnits(amountToSell.toString(), "ether");
        const signer = new ethers.Wallet(signerPrivateKey, provider);
        let nonceBase = provider.getTransactionCount(signer.getAddress());
        let offsetNonce = 0;
        function getNonce() {
            return nonceBase.then((nonce) => (nonce + (offsetNonce++)));
        }
        var BDP = new ethers.Contract(addressContract, abi, signer);
        var allowance = await BDP.allowance(signer.address, addressContract, {nonce: getNonce()});
        if (BigNumber.from(allowance).lt(BigNumber.from(amountToSell))) {
            try {
                var tr1 = BDP.increaseAllowance(addressContract, amountToSell, {nonce: getNonce()});
                var tx1 = await signer.sendTransaction(tr1, {nonce: getNonce()});
                console.log("Miners perform a transaction...");
                console.log(`https://${network}.etherscan.io/tx/${tx1.hash}`);
                const receipt = await tx1.wait();
                console.log(`Block is mined in ${receipt.blockNumber}`);
            } catch (e) {
                console.log(error.message);
                return "ERROR";
            }
        }
        var transaction = BDP.sell(amountToSell, {nonce: getNonce(), gasLimit: 3000000});
        var sendTransactionPromise = await signer.sendTransaction(transaction);
        console.log("Miners perform a transaction...");
        console.log(`https://${network}.etherscan.io/tx/${sendTransactionPromise.hash}`);
        const receipt1 = await sendTransactionPromise.wait();
        console.log(`Block is mined in ${receipt1.blockNumber}`);
        return "DONE";
    }
    catch(error) {
        console.log(error.message);
        return "ERROR";
    }
}
```

Рисунок 35 – Алгоритм функції *sell* зі сторони сервера

Наступним кроком є створення екземпляру контракту BDP токена (*ethers.Contract*), використовуючи такі значення, як *addressContract*, *abi* та *signer*.

Потім отримується поточне значення дозволених токенів (*allowance*) для продажу активу користувача в самому смарт-контракті. В умові відбувається перевірка, що якщо значення дозволених токенів менше за *amountToSell*, виконується відправка транзакції (*sendTransaction*) на збільшення дозволу за допомогою функції контракту *increaseAllowance*, яка приймає такі аргументи, як *addressContract*, *amountToSell, {nonce: getNonce()}*.

Після описаного блоку виконується операція по продажу криптовалюти (*sell*), яка викликається зі смарт-контракту токена BDP з такими аргументами: *amountToSell, {nonce: getNonce(), gasLimit: 3000000}.* Значення *gasLimit* описує, що транзакція має ліміт вартості газу, який дорівнює 3000000. Така кількість обрана для забезпечення достатньої суми газу на виконання операції продажу криптовалюти.

Далі викликається метод *signer.sendTransaction*, який надсилає підписану транзакцію (*transaction*) у мережу Ethereum на обробку та отримує результат (*sendTransactionPromise*), який повинен дочекатися отримання підтвердження транзакції. Наступний метод *sendTransactionPromise.wait()* очікує виконання транзакції та отримує новий блок даних, який буде занесено до змінної *receipt1*, що містить інформацію щодо підтвердженої транзакції.

В кінці функції повертається рядок, що уточнює успішне виконання операції продажу токенів BDP, але у випадку виникнення помилки на будь-якому кроці, функція повертає рядок, який сигналізує про невдачу.

На рис. 36 зображено частину коду функції *gasLessTransfer*, що забезпечує основну відмінність транзакції, яка виконується без оплати газу для користувача.

Покрокове виконання такого коду:

- створення екземпляру контракту *BDP*, використовуючи аргументи *addressContract*, *abi*, *owner_signer* (акаунт власника токена);
- заповнення змінної *domainSeparator* значенням, яке надходить з методу контракту *BDP.DOMAIN_SEPARATOR* з аргументом *nonce* (ID транзакції);
- визначення параметрів для переказу (*transferParams*) з такими властивостями: *from* (адреса користувача, який хоче переказати токени), *to* (адреса отримувача), *value* (кількість токенів), *validAfter* та *validBefore*;
- виконується деструктуризація об’єкту *transferParams* в окремі змінні;
- відбувається виклик раніше описаної функції *signTransferAuthorization*, вона необхідна для підписання операції переказу, використовуючи такі аргументи: *from*, *to*, *value*, *validAfter*, *validBefore*, *nonce*, *domainSeparator* та

userSignerPrivateKey (PK користувача, що хоче здійснити переказ). Результати виконання заносяться до констант *v*, *r*, *s*, які зберігають отриманий підпис;

- на даному етапі можна створювати об'єкт транзакції (*transaction*) з отриманим підписом користувача. Транзакція представляє метод із контракту (*BDP.gassLessTransfer*) з такими параметрами для переказу, як *from*, *to*, *value*, *validAfter*, *validBefore*, *nonce*, *v*, *r*, *s*, *{nonce: getNonce()};*
- сформована транзакція надсилається у мережу Ethereum від власника контракту (*owner_signer.sendTransaction*).

```
var BDP = new ethers.Contract(addressContract, abi, owner_signer);
var domainSeparator = await BDP.DOMAIN_SEPARATOR({nonce: getNonce()});
const transferParams = {
    from: await user_signer.getAddress(),
    to: addressTo,
    value: amountToTransfer,
    validAfter: 0,
    validBefore: MAX_UINT256,
};
const { from, to, value, validAfter, validBefore } = transferParams;
const { v, r, s } = signTransferAuthorization(from, to, value, validAfter, validBefore,
                                              nonce, domainSeparator, userSignerPrivateKey);
var transaction = BDP.gassLessTransfer(from, to, value, validAfter, validBefore,
                                       nonce, v, r, s,{nonce: getNonce()});
var sendTransactionPromise = await owner_signer.sendTransaction(transaction);
```

Рисунок 36 – Фрагмент коду функції *gasLessTransfer*

2.8 Функціонал акаунтів та використання алгоритмів шифрування

Далі необхідно зосередитись на функціоналі щодо акаунтів користувачів. В даному підрозділі увагу зосереджено на створенні, авторизації та перевірці вже існуючого акаунту до системи.

Створення акаунту користувача зображено на рис. 37, метод *createAccount* класу *Account_Handler* має параметри, які складаються з рядків *login*, *private_key* та *password.* Спочатку створюється текстовий файл в директорії акаунтів (*account_path*), назва якого відповідає логіну користувача. Файл відкривається в

режимі запису (*WriteOnly*) і вносить РК власника акаунту у зашифрованому вигляді (*encrypt*), що використовує пароль (*password*) для шифрування. Результат шифрування представляє собою байтовий масив даних. Метод використовується як для імпорту, так і для створення нових акаунтів у систему.

```
void Account_Handler::createAccount(const QString &login, const QString &private_key,
                                    const QString &password) {
    QString account_path=path+"/"+login+".txt";
    QFile data_reader;
    data_reader.setFileName(account_path);
    data_reader.open(QFile::WriteOnly);
    data_reader.write(encrypt(private_key, password, vector_phrase));
    data_reader.close();
}
```

Рисунок 37 – Метод *createAccount*

Авторизація акаунту користувача показана на рис. 38, метод *authorize* класу *Account_Handler* включає параметри, які складаються з рядків *login* та *password.* Даний метод призначений для авторизації користувача та має схожий функціонал з попереднім, але з деякими відмінностями. В даному випадку, файл акаунту відкривається тільки для читання (*ReadOnly)* та зчитує (*readAll*) усі дані, що потім зберігаються в константі *private_key* у вигляді байтового масиву (*QByteArray*). Після закриття файлу викликається функція *decrypt*, яка повинна розшифрувати отримані дані (*private_key*) за допомогою пароля (*password*). Результат операції конвертується в рядок (*QString*) та повертається з методу.

```
QString Account_Handler::authorize(const QString &login, const QString &password) {
    QString account_path=path+"/"+login+".txt";
    QFile data_reader;
    data_reader.setFileName(account_path);
    data_reader.open(QFile::ReadOnly);
    const QByteArray private_key=data_reader.readAll();
    data_reader.close();
    return QString(decrypt(private_key, password, vector_phrase));
}
```

Рисунок 38 – Метод *authorize*

Перевірка акаунту користувача на існування зображена на рис. 39, метод *doesExistAccount* класу *Account_Handler* має один єдиний параметр (*login*). Кінцевим результатом виконання є повернення *true*, якщо акаунт вже створено у директорії та *false*, якщо ні.

```
bool Account_Handler::doesExistAccount(const QString &login) {
    QString account_path=path+"/"+login+".txt";
    QFile data_reader;
    data_reader.setFileName(account_path);
    if (data_reader.exists())
        return true;
    return false;
}
```

Рисунок 39 – Метод *doesExistAccount*

Наступні методи застосовуються у шифруванні (*encrypt*) РК та його розшифруванні (*decrypt*), використовуючи алгоритм AES-256, що містить реалізацію основних операцій шифрування, таких як додавання ключа раунду, заміна байтів, зсув рядків, змішування стовпців тощо [15].

Метод *encrypt* зображено на рис. 40, на вхід він отримує текст (*text*), який необхідно зашифрувати, ключ (*passphrase*) та *vector_phrase* в якості параметрів. Тіло методу складається з виклику функції *QAESEncryption::Crypt* та повернення результату її виконання у вигляді масиву байтів (*QByteArray*).

Покрокове роз'яснення аргументів функції *QAESEncryption::Crypt*:

- *AES_256*: використання алгоритму AES з ключем довжиною 256 бітів;
- *CBC*: режим шифрування Cipher Block Chaining;
- *text.toLocal8Bit()*: перетворення тексту (*text*) на масив байтів;
- *hash(passphrase.toLocal8Bit(), QCryptographicHash::Sha256)*: виконання хешування ключа (*passphrase*), використовуючи алгоритм SHA-256 (Secure Hash Algorithm);
- *hash(vector_phrase.toLocal8Bit(), QCryptographicHash::Md5)*: виконання хешування *vector_phrase* за допомогою алгоритму MD5 (Message Digest 5).

```
QByteArray Account_Handler::encrypt(const QString &text, const QString &passphrase, const QString &vector_phrase) {
    return QAESEncryption::Crypt(QAESEncryption::AES_256, QAESEncryption::CBC, text.toLocal8Bit(),
                                 QCryptographicHash::hash(passphrase.toLocal8Bit(), QCryptographicHash::Sha256),
                                 QCryptographicHash::hash(vector_phrase.toLocal8Bit(), QCryptographicHash::Md5));
}
```

Рисунок 40 – Метод шифрування даних

Метод *decrypt* представлено на рис. 41, він отримує зашифрований текст (*text*), який необхідно розшифрувати, ключ (*passphrase*) та *vector_phrase* в якості вхідних параметрів. Метод схожий за функціоналом на *encrypt* та складається з виклику функції *QAESEncryption::Decrypt.*

```
QByteArray Account_Handler::decrypt(const QByteArray &text, const QString &passphrase, const QString &vector_phrase) {
    return QAESEncryption::RemovePadding
            (QAESEncryption::Decrypt(QAESEncryption::AES_256, QAESEncryption::CBC, text,
                                     QCryptographicHash::hash(passphrase.toLocal8Bit(), QCryptographicHash::Sha256),
                                     QCryptographicHash::hash(vector_phrase.toLocal8Bit(), QCryptographicHash::Md5)));
}
```

Рисунок 41 – Метод розшифрування даних

Для наочності, необхідно розглянути як описані методи взаємодіють в самому застосунку. Представлений код описує три кнопки (*OutlineButton*) в GUI, які виконують функціонал акаунту.

Перша зображена на рис. 42 та має назву *Create account*, вона призначена для створення облікового запису користувача в застосунку. Поле *enabled* використовує функцію *enable* для визначення стану кнопки в залежності від вже введених даних (*creating_login, encryption_with_password_for_registering, creating_password*), що вказує на те, чи є можливість натиснути на кнопку в цей момент часу.

Після натискання на кнопку (*onClicked*) створюється новий акаунт користувача (*_ACCOUNT_*) у мережі Ethereum, використовуючи функцію *web3.eth.accounts.create()* [12]. Далі виконується *account_handler.createAccount* функція, що створює обліковий запис у застосунку за допомогою введеного логіну (*creating_login.text*), приватного ключа створеного акаунту під обліковий

запис (*_ACCOUNT_.privateKey*) та введеного паролю (*creating_password.text*). Завершуючи всі виконані дії, виконується перехід до меню, використовуючи функцію *come_back*.

```
Qaterial.OutlineButton {
    text: "Create account"
    enabled: enable(creating_login,
                    encryption_with_password_for_registering, creating_password)
    onClicked: () ⇒ {
        _ACCOUNT_=web3.eth.accounts.create()
        account_handler.createAccount(creating_login.text,
                                      _ACCOUNT_.privateKey, creating_password.text)
        come_back(creating_window, creating_login, creating_password,
                  encryption_with_password_for_registering, menu)
    }
}
```

Рисунок 42 – Кнопка створення акаунту

Друга кнопка зображена на рис. 43 та має назву *Log in*, вона призначена для входу в застосунок за допомогою вже створеного акаунту користувача. Властивість *enabled* використовується ідентично кнопці *Create account.* При натисканні кнопки (*onClicked*) запускається функція-обробник, яка виконує деякий код.

```
Qaterial.OutlineButton {
    text: "Log in"
    enabled: enable(login, using_password_for_logging_in, password)
    onClicked: () ⇒ {
        var decrypted_private_key=account_handler.authorize(login.text, password.text)
        if (decrypted_private_key.length≠=66||!web3.utils.isHex(decrypted_private_key)) {
            using_password_for_logging_in.checked=true
            password.setError("Incorrect password!")
        }
        else {
            _ACCOUNT_=web3.eth.accounts.privateKeyToAccount(decrypted_private_key)
            come_back(log_in_window, login, password, using_password_for_logging_in, menu)
        }
    }
}
```

Рисунок 43 – Кнопка авторизації акаунту

Відбувається створення змінної *decrypted_private_key*, в яку заноситься розшифрований PK користувача в результаті виконання функції авторизації (*account_handler.authorize*), використовуючи *login* та *password*. Наступною дією є перевірка PK на валідність, тобто PK повинен бути довжиною в 66 символів та записаний у шістнадцятковому форматі (*web3.utils.isHex*). Якщо умова виконана, то створюється об'єкт акаунту користувача (*_ACCOUNT_*) в мережі Ethereum за допомогою розшифрованого PK (*web3.eth.accounts.privateKeyToAccount*) та відбувається перенаправлення до меню (*come_back*) [12]. В іншому випадку, об'єкт не створюється та виводиться помилка *"Incorrect password!"*.

Третя кнопка показана на рис. 44 та має назву *Import*, вона призначена для імпорту вже створеного акаунту в застосунок. В даному випадку, поле *enabled* не використовує зовнішньої функції та використовується для визначення стану кнопки в залежності від того, чи правильно введені всі необхідні дані (*import_login*, *import_private_key*, *import_password*).

```
Qaterial.OutlineButton {
    text: "Import"
    enabled: (!import_login.errorState&&import_login.length)&&
             (!import_private_key.errorState&&import_private_key.length)&&
             (!encryption_with_password_for_importing.checked||
              (!import_password.errorState&&import_password.length))
    onClicked: () => {
        _ACCOUNT_=web3.eth.accounts.privateKeyToAccount(import_private_key.text)
        account_handler.createAccount(import_login.text, _ACCOUNT_.privateKey, import_password.text)
        come_back(import_window, import_login, import_password,
                  encryption_with_password_for_importing, menu)
    }
}
```

Рисунок 44 – Кнопка імпорту акаунту

Натискаючи на кнопку (*onClicked*), спочатку створюється об'єкт акаунту користувача (*_ACCOUNT_*) на платформі Ethereum, який використовує імпортований PK (*import_private_key*) та *web3.eth.accounts.privateKeyToAccount* функцію [12]. Далі виконується вже відома дія *account_handler.createAccount*, яка створює обліковий запис у застосунку через введений логін (*import_login.text*),

імпортований PK (*_ACCOUNT_.privateKey*) та новий пароль користувача (*import_password.text*).

Таким чином, у всіх трьох випадках в коді використовуються функції та змінні від бібліотек web3 та account_handler, де web3 містить функціонал для взаємодії з мережею Ethereum, а account_handler має логіку задіяну з акаунтами.

3. ТЕСТУВАННЯ СИСТЕМИ

3.1 Розгортання смарт-контракту токена в тестовій мережі Ethereum

Останній розділ дипломного проєкту повинен описувати тестування системи на працездатність та ефективність. Даний підрозділ містить інформацію про процес розгортки смарт-контракту токена на платформі Ethereum. Для такої цілі було використано IDE (інтегроване середовище розробки) Remix, вона дозволяє виконувати процес розробки, компіляції, розгортки та тестування смарт-контракту в зручний та легкий для програміста спосіб [13].

Після налаштування та завантаження розробленого смарт-контракту токена до середовища розробки, повинна виконуватись компіляція файлу, що перевірить правильність синтаксису та переведе написаний код у машинний. На рис. 45 зображено вікно взаємодії для виконання заданої операції. На ньому спостерігається компілятор та його версія, розроблений смарт-контракт для компіляції з розширенням .sol та кнопка для запуску даного процесу.

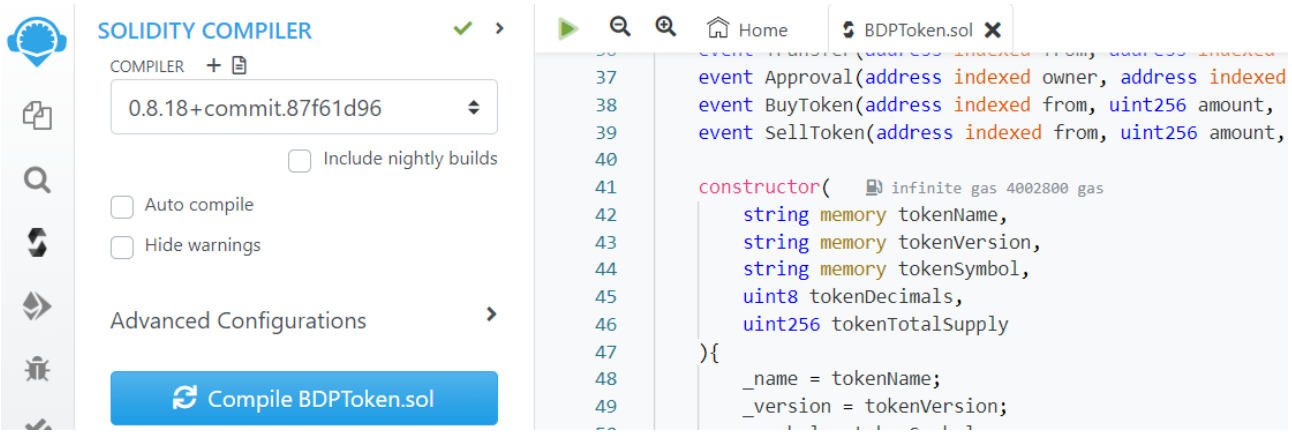

Рисунок 45 – Компіляція смарт-контракту токена

По закінченню компіляції, необхідно розгорнути смарт-контракт на платформі Ethereum. Для цього використовується тестова мережа sepolia та акаунт власника контракту.

На рис. 46 представлено виконання процесу розгортки. На зображенні фігурує провайдер MetaMask, через який здійснюється підключення до мережі sepolia. Нижче наведено адресу, яка ініціює розгортання скомпільованого смарт-контракту та згодом стане його власником. Під опцією *DEPLOY* спостерігаються базові налаштування токена, які будуть згенеровані конструктором, а саме: назва (*TOKENNAME*), версія (*TOKENVERSION*), символ (*TOKENSYMBOL*), десяткові знаки (*TOKENDECIMALS*) та загальна кількість (*TOKENTOTALSUPPLY*) випущених токенів. Праворуч прослідковується відправка транзакції, що ініціює розгортання смарт-контракту. Для виконання такої операції необхідно оплатити газ тестової мережі та натиснути кнопку підтвердження.

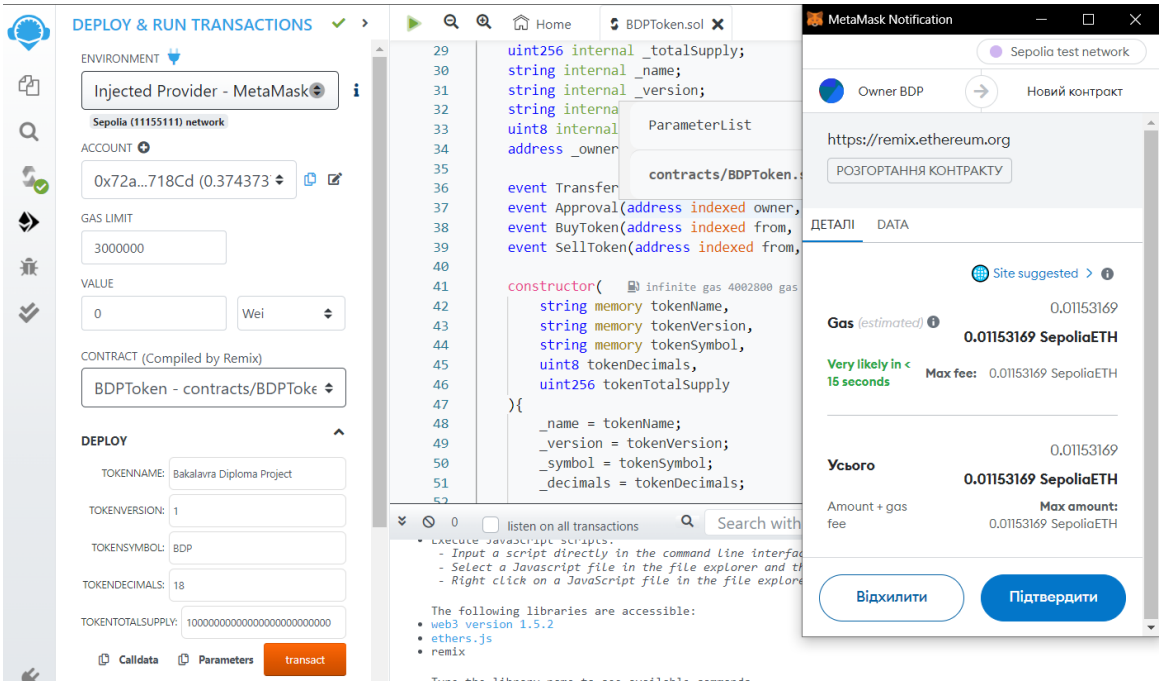

Рисунок 46 – Розгортання смарт-контракту токена

Якщо транзакція була відправлена вдало, то результатом виконаних дій повинна бути адреса розгорнутого смарт-контракту та інші важливі параметри у мережі, за допомогою якої відбувається взаємодія з ним.

На рис. 47 зображено частину методів контракту вже після успішно виконаних налаштувань, де спостерігаються деякі транзакції та їх результати.

В даному випадку було викликано такі методи контракту: *transfer*, *allowance*, *balanceOf* та *decimals*. Наприклад, *transfer* виконує переказ токенів на адресу користувача (*recipient*), де кількість самих токенів (*amount*) представлена з десятковими знаками. Якщо перевести у звичайний формат, то повинно бути переказано 50.5 токенів BDP. Вдале виконання операції підтверджує транзакція, що представлена поруч з методами контракту.

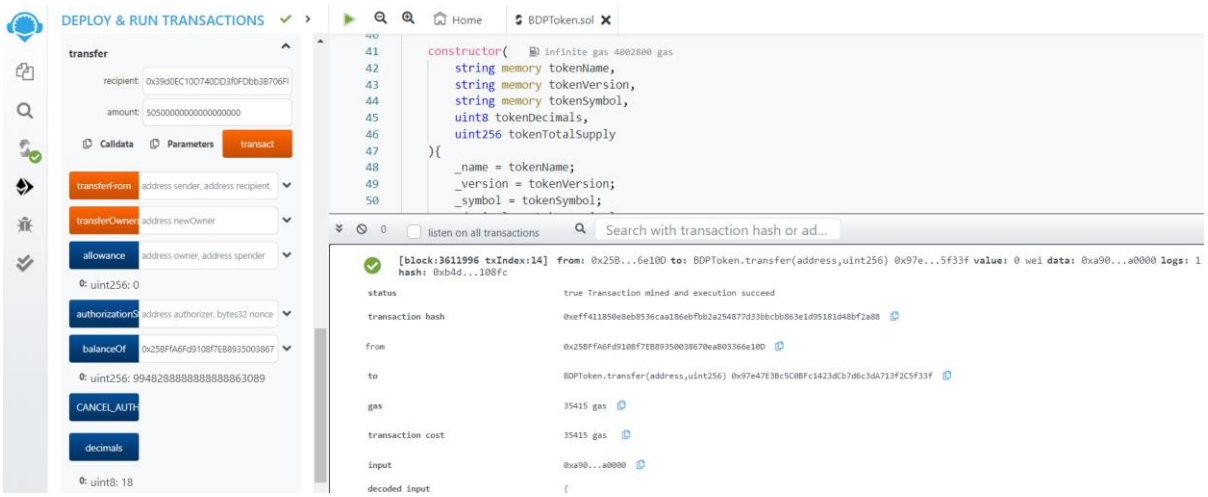

Рисунок 47 – Частина методів та виконана транзакція *transfer* контракту

3.2 Перевірка результатів та тестування функціональності системи

Даний підрозділ повинен містити перевірку всіх сценаріїв використання розробленої системи. Всі операції з криптовалютою мають виконуватись успішно та повинні прослідковуватись не тільки в розробленому застосунку, а й у інших аналогах, наприклад, MetaMask.

Також, створення та імпорт акаунту має функціонувати без помилок та вдало шифрувати РК в обліковому записі. Приватні ключі акаунтів, що зберігаються локально, мають бути захищені та розшифровуватись безперешкодно для виконання авторизації користувача.

На рис. 48 зображено початкове вікно програми, що відображає описану взаємодію з акаунтами.

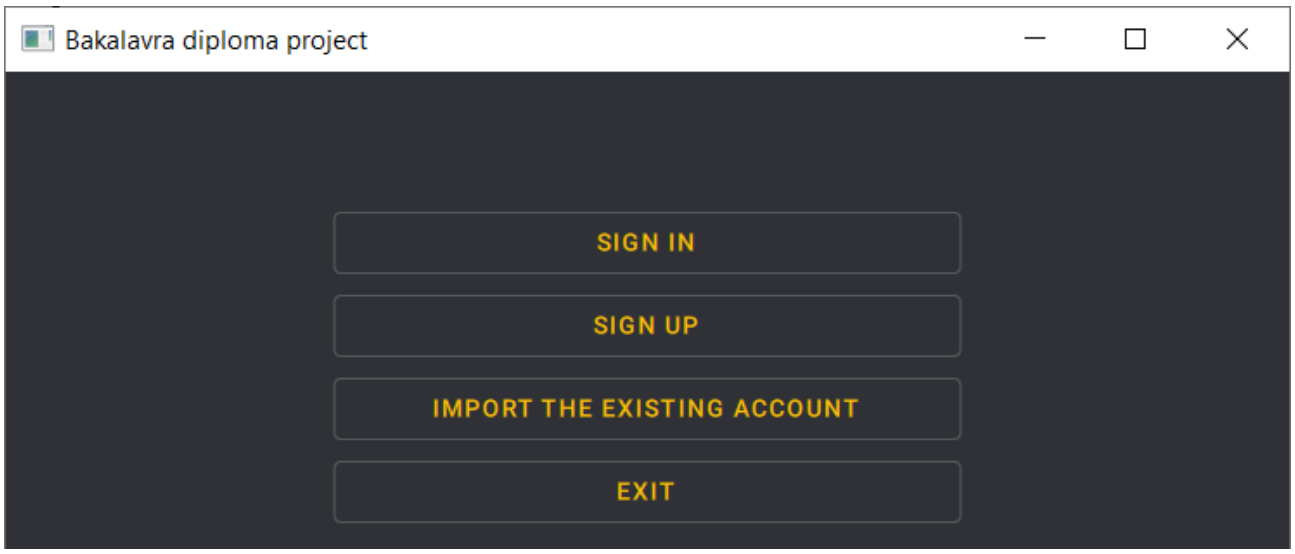

Рисунок 48 – Початкове вікно програми

Опція *SIGN IN* призначена для авторизації користувача, *SING UP* виконує функцію реєстрації облікового запису, *IMPORT THE EXISTING ACCOUNT* для імпорту акаунту з мережі Ethereum та *EXIT* для виходу із застосунку.

На рис. 49 представлено процес авторизації користувача (*SING IN*). Для того, щоб увійти в застосунок, необхідно ввести правильний логін (*Login*). Якщо обліковий запис був захищений паролем (*Password*), то його теж треба заповнити. Аби впевнитись у коректності введеного пароля можна натиснути на значок ока.

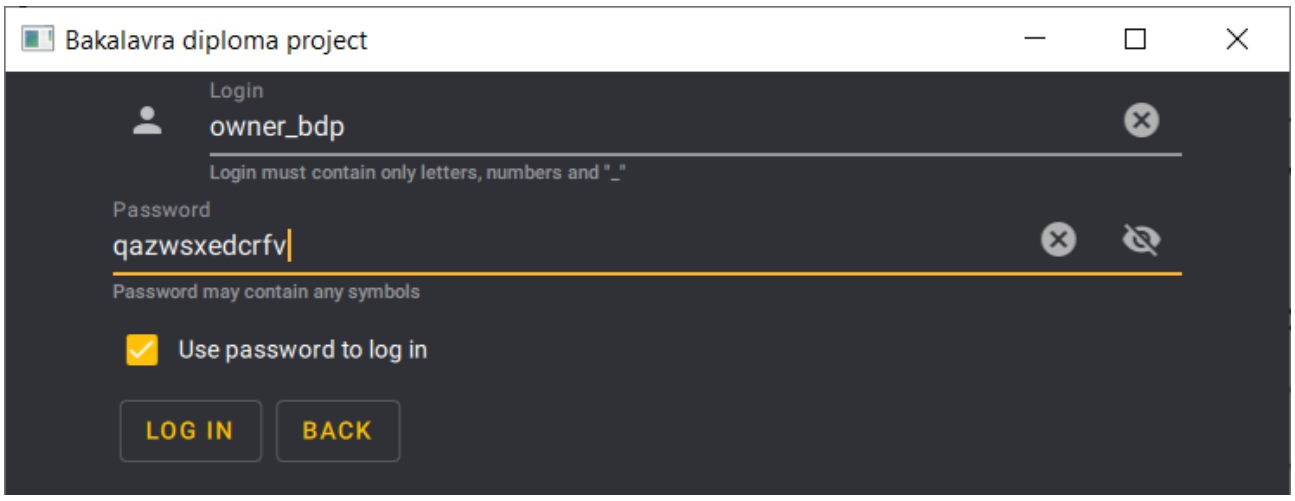

Рисунок 49 – Авторизація користувача

Створення облікового запису в програмі (рис. 50) схоже за інтерфейсом до авторизації. В даному випадку також необхідно заповнити логін (*Login*) та за необхідності пароль (*Password*) для надійного захисту акаунту, що будуть використовуватись у подальшому.

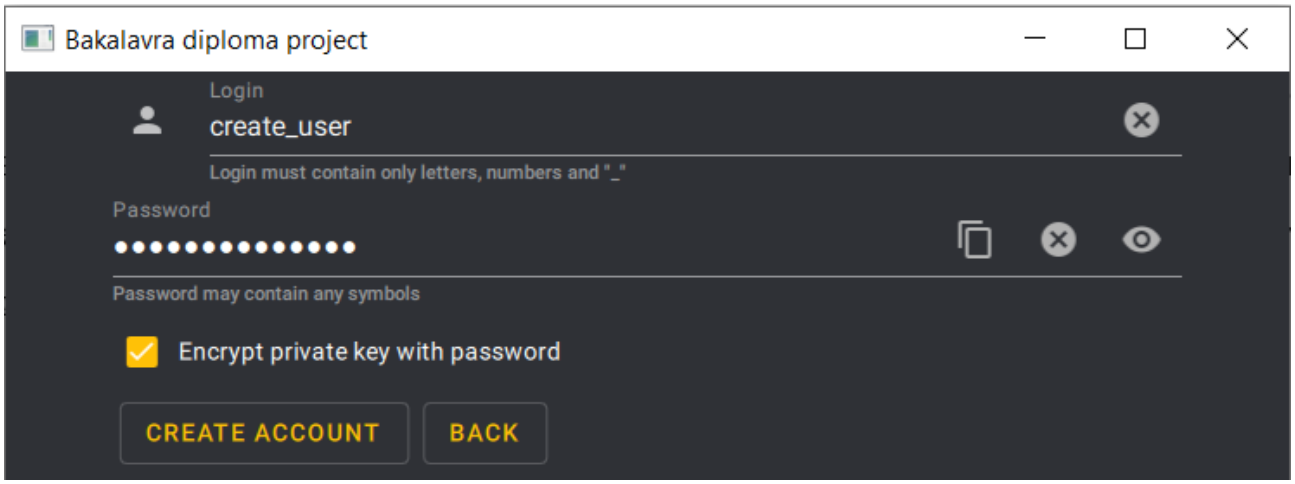

Рисунок 50 – Створення облікового запису

Опція імпортування (рис. 51) надає можливість зареєструвати в системі будь-який вже створений деінде акаунт мережі Ethereum.

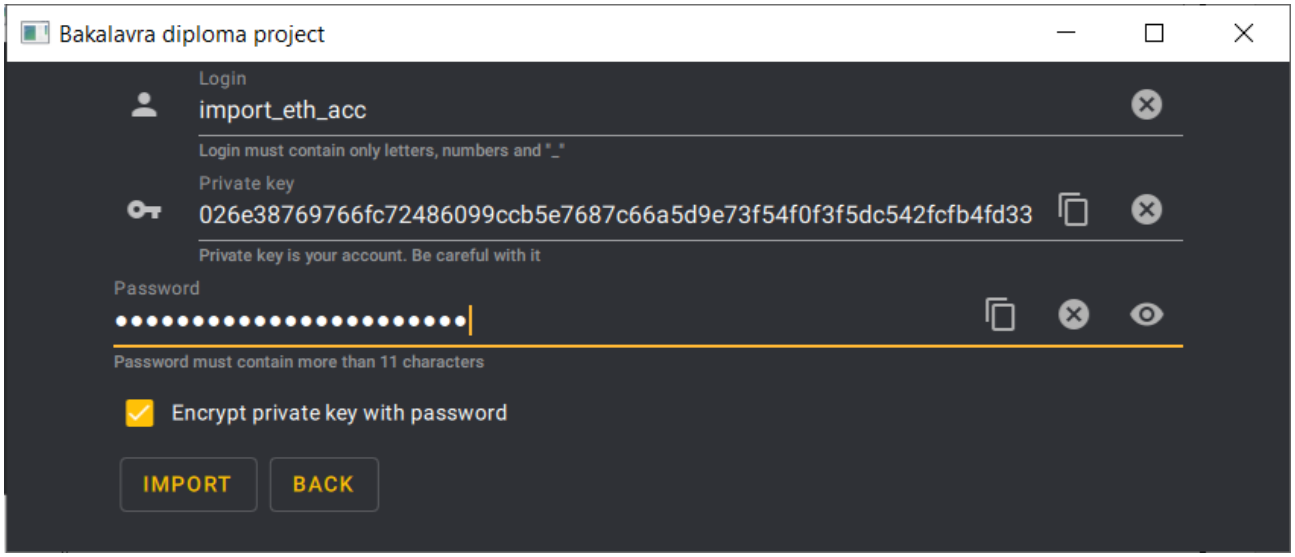

Рисунок 51 – Імпортування існуючого акаунту

Інструкція по імпорту ідентична створенню, але в цьому випадку ще необхідно ввести PK акаунту (*Private key*), який користувач хоче зареєструвати під обліковим записом в даному застосунку.

Якщо вхід в програму було виконано успішно, то повинно з'явитись головне вікно (*Settings*) з інформацією, яка відображає публічну адресу акаунту (можна скопіювати), його PK (можна приховати та скопіювати), курс обміну, кількість ETH та BDP токенів на балансі користувача. На рис. 52 зображено приклад, як таке вікно має виглядати.

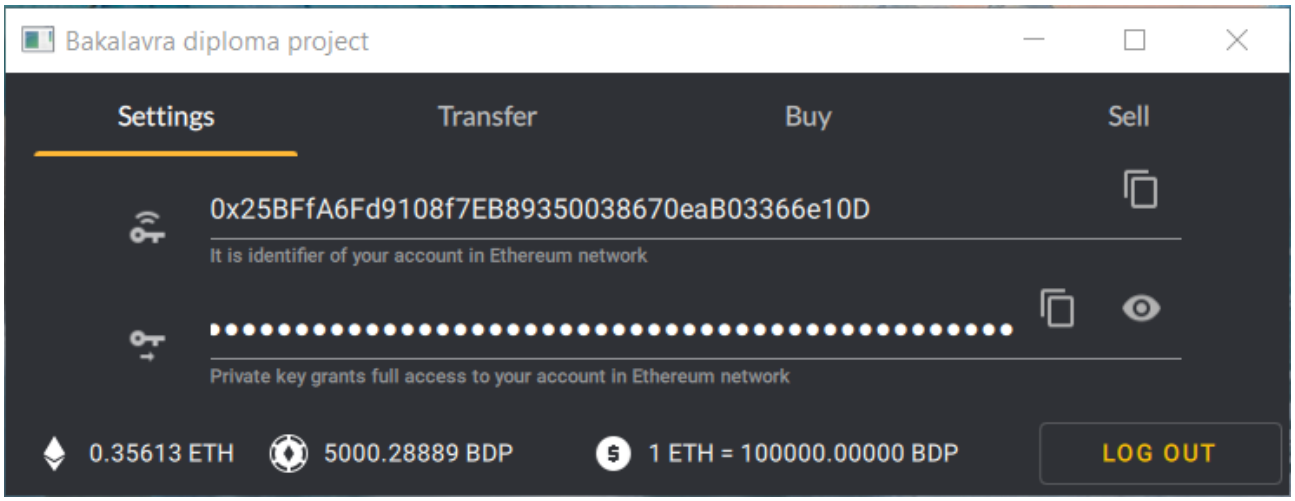

Рисунок 52 – Головне вікно програми (*Settings*)

На кожній вкладці в системі повинна відображатись інформація щодо курсу обміну та балансу користувача, який оновлюється динамічно. Також, з кожного вікна є можливість перейти в будь-яке інше із запропонованих (*Settings*, *Transfer*, *Buy*, *Sell*) та вийти з облікового запису (*LOG OUT*).

На рис. 53 представлена вкладка *Transfer* (переказ токенів), для виконання цієї операції визначені такі поля: *Address* (адреса отримувача), *Amount of BDP* (кількість BDP токенів) та *Without the commission* (прапорець для позначення переказу без використання газу). Комісія списується в токенах ETH, після налаштування всіх параметрів необхідно натиснути кнопку *TRANSFER*.

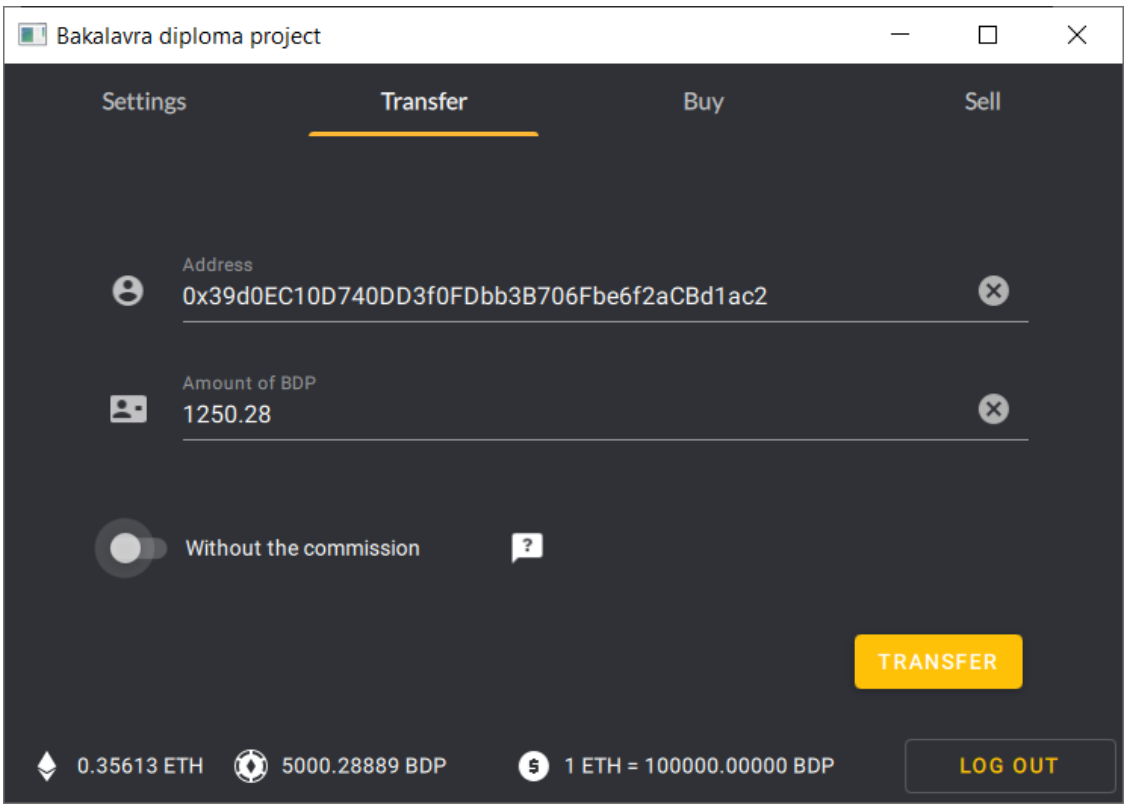

Рисунок 53 – Операція *Transfer*

На рис. 54 представлено запуск веб-сервера, який виконує обробку транзакцій за допомогою взаємодії з Infura провайдером, який виконує з'єднання з блокчейном Ethereum. Ще нижче вказується, що клієнт був під'єднаний.

```
PS C:\Users\lnew7\Desktop\server old 2\scripts> node server
The WebSocket server is running on port 1011
New client connected!
```

Рисунок 54 – Запуск сервера

Рис. 55 містить операцію переказу токенів на стороні сервера, де він виконує логування його дій. Можна спостерігати, що спочатку відбувається формування та відправка транзакції. Потім сервер надає посилання на неї, де можна переглянути деталі щодо виконаної операції в мережі Ethereum та номер блоку, в який транзакція була включена.

```
Sending...
Miners perform a transaction...
https://sepolia.etherscan.io/tx/0x73fc6295cdc632ca138b6bbfe14aad51d9bf23862232f833dd6ac8b7c6547654
Block is mined in 3615812
```

Рисунок 55 – Транзакція переказу на сервері

Для підтвердження виконаних дій необхідно перевірити результат в застосунку MetaMask, що був описаний як аналогічний застосунок у першому розділі роботи. На рис. 56 представлено успішність операції. Спостерігається, що адреса акаунту співпадає та решта BDP токенів відповідає кількості, що повинна була залишитись на балансі користувача.

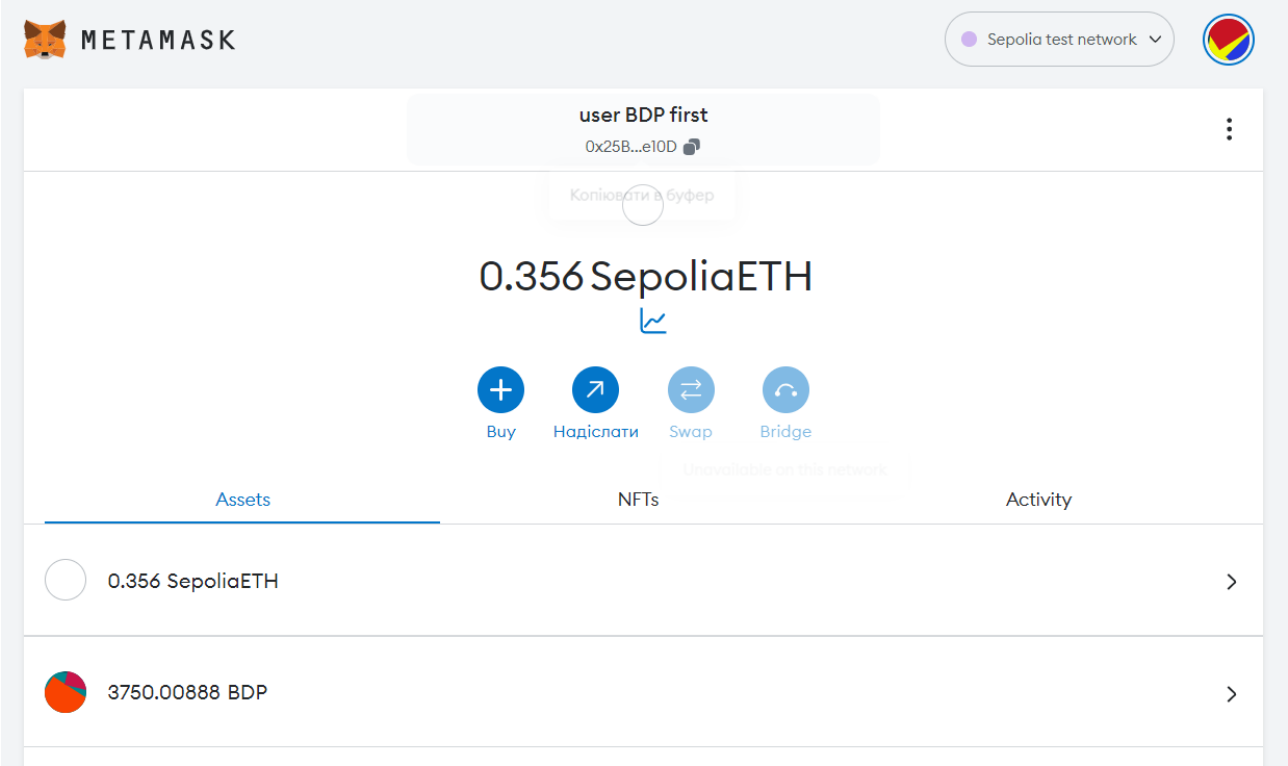

Рисунок 56 – Перевірка успішності операції переказу в MetaMask

В свою чергу, необхідно переконатись, що токени дійсно надійшли на баланс іншого користувача. На рис. 57 показано, що кількість BDP токенів дорівнює тому самому значенню, яке було відправлене.

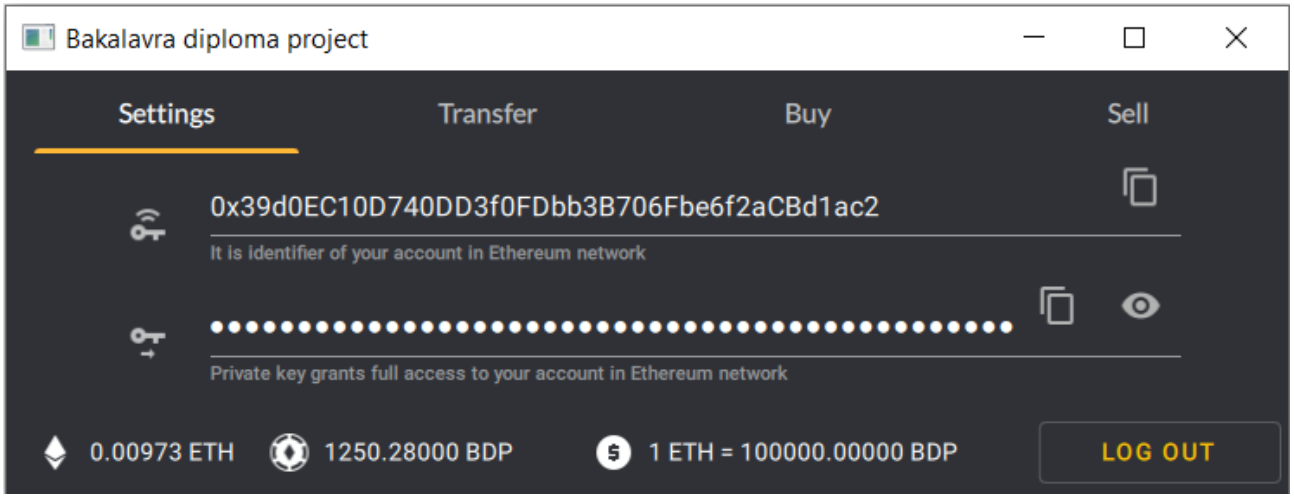

Рисунок 57 – Отримання надісланих токенів

Наступна операція зображена на рис. 58 та має те саме призначення, що й попередня, але здійснюється без використання комісії (газу), якщо активувати відповідний прапорець. Переказ токенів виконується між тими самими адресами, що й у минулій операції.

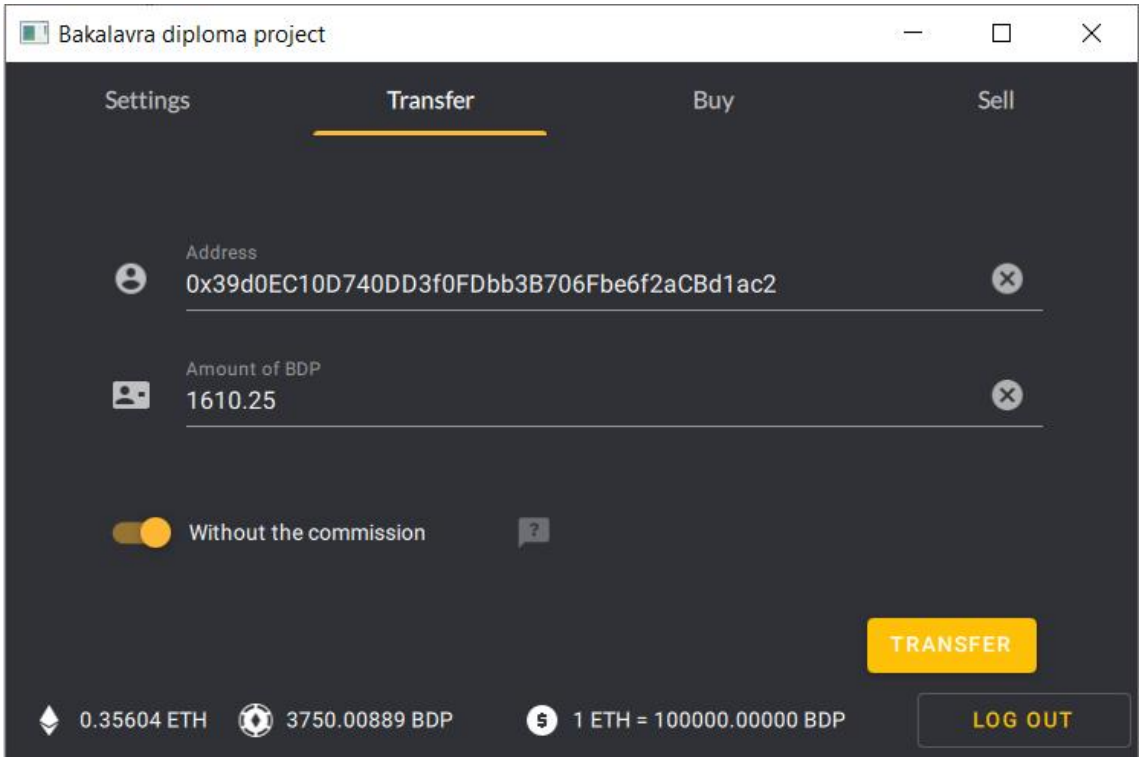

Рисунок 58 – Операція *Transfer* без газу

На рис. 59 показано логування дій сервера, який надав нове посилання на виконану транзакцію в мережі Ethereum.

```
Sending...
Miners perform a transaction...
https://sepolia.etherscan.io/tx/0x978ac171dff877edfdc77bdf38e13d32c23616e0ec903f27d0a9175c0cb3d3bf
Block is mined in 3624962
```

Рисунок 59 – Транзакція переказу без використання газу на сервері

Необхідно перевірити, чи дійсно операція була виконана без оплати комісії. На рис. 60 представлено баланс ETH та BDP токенів, які оновили свої значення відразу після здійсненої транзакції переказу. Якщо прослідкувати за кількістю ETH, то сума цієї криптовалюти залишилася незмінною. Щодо токенів BPD, то вони відповідають тому значенню, що повинно було залишитись після операції.

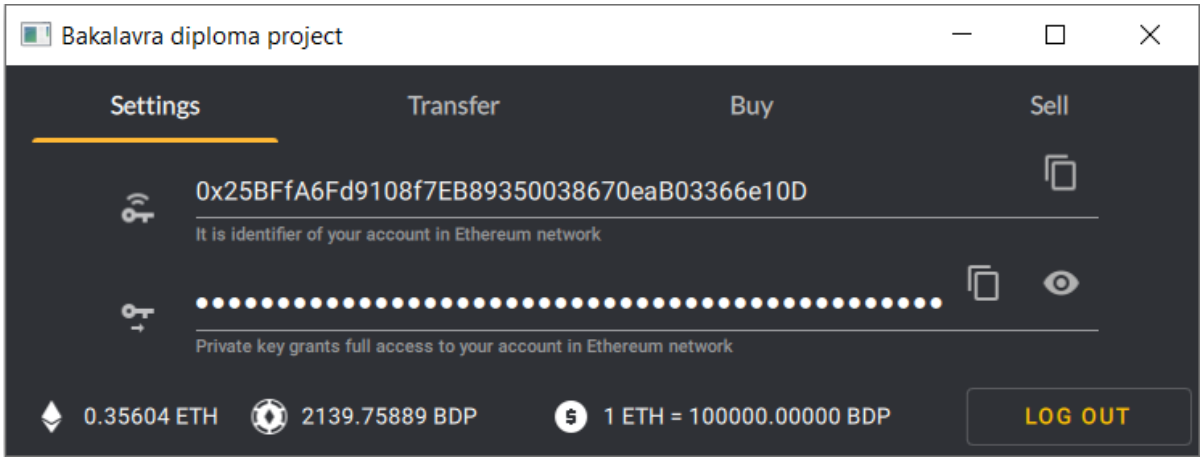

Рисунок 60 – Оновлена кількість токенів після переказу без комісії

В даному випадку, успішність виконаної операції перевіряється по адресі отримувача токенів в застосунку MetaMask. На рис. 61 чітко прослідковується, що на балансі користувача знаходиться правильна кількість токенів BDP. Якщо порахувати суму токенів від виконаних операцій переказу, то отримаємо, що 1250.28 + 1610.25 = 2860.53 BDP.

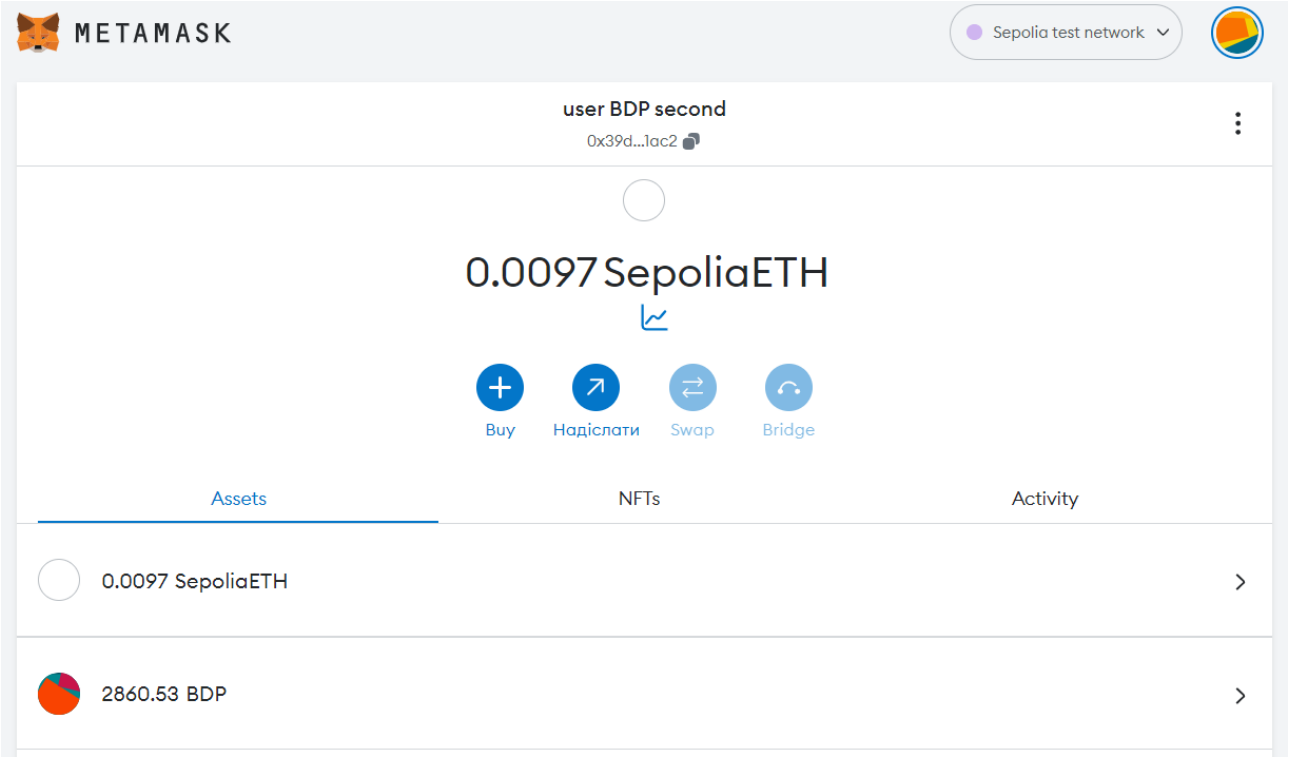

Рисунок 61 – Баланс отримувача токенів після операції переказу без комісії

Далі виконується тестування операції купівлі криптовалюти. На рис. 62 зображено вкладку *Buy* та поле *Amount of BDP* (кількість BDP токенів), виконання відбувається на тому обліковому записі (first_bdp), з якого здійснювався переказ токенів.

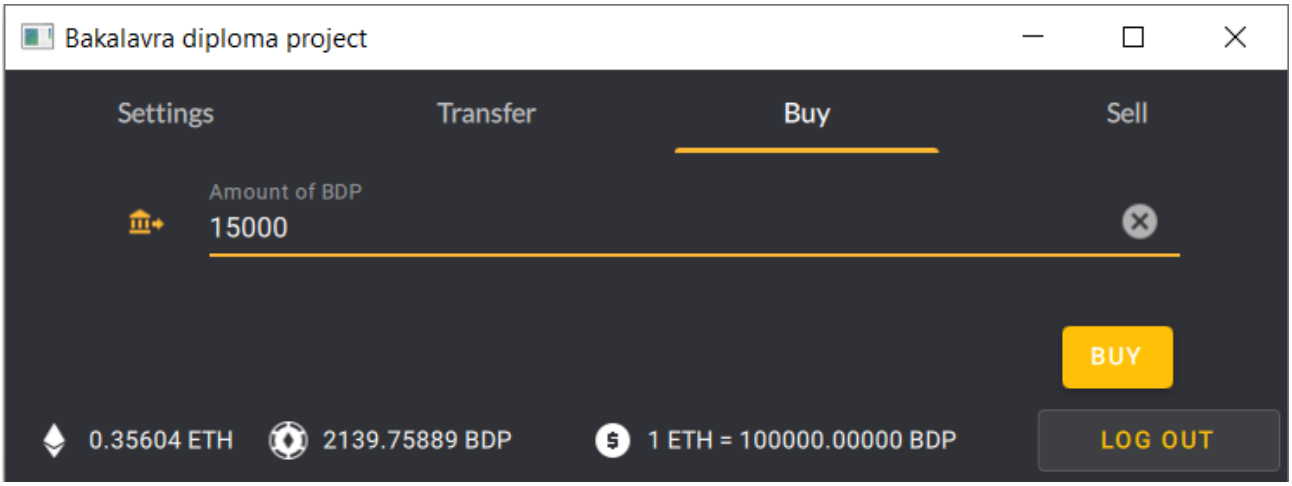

Рисунок 62 – Операція *Buy*

Виконання транзакції вважається успішним, якщо купівля криптовалюти пройшла по зазначеному курсу обміну та оновлений баланс є правильним. На рис. 63 перевіряються отримані значення, де кількість ETH дорівнює 0.20579 та сума BDP токенів становить 17139.75889. Визначені результати є вірними, тому що значення ETH повинно було зменшитись на 0.15 по курсу обміну з деякою додатковою комісією, а кількість BDP токенів збільшитись на 15000.

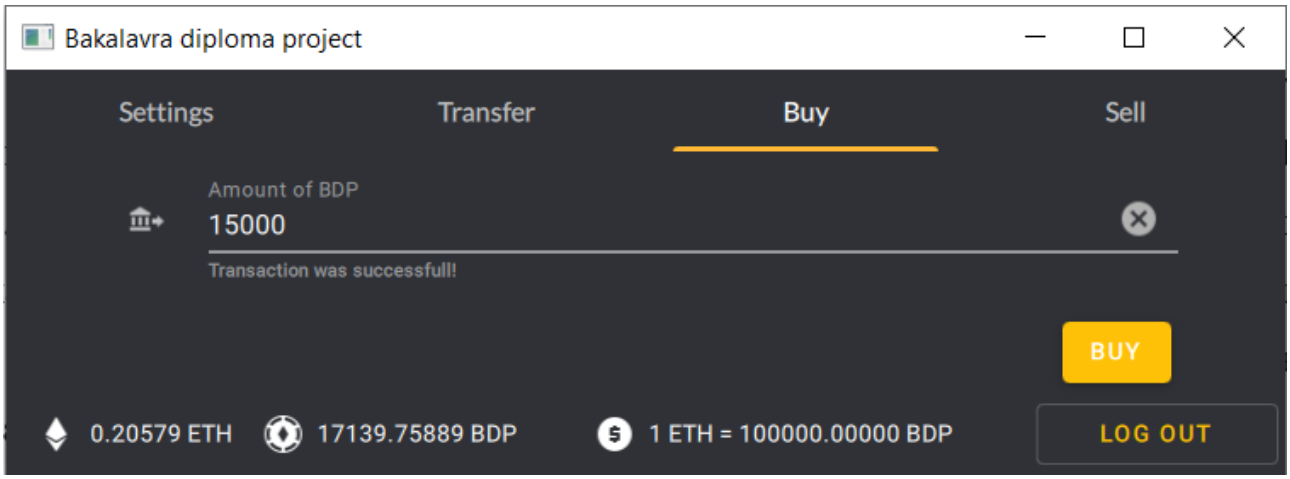

Рисунок 63 – Результати операції *Buy*

Посилання на успішно відправлену та оброблену транзакцію на платформі Ethereum зображено на рис. 64 (веб-сервер).

```
Sending...
Miners perform a transaction...
https://sepolia.etherscan.io/tx/0x510383a031d0e8c6b28b74143f98ad09efd2030f4284254f4efefb9d1b049ab2
Block is mined in 3625381
```

Рисунок 64 – Посилання на транзакцію операції *Buy*

Для того, щоб переконатись, що транзакція була виконана успішно та результати збережені у мережі Ethereum, необхідно перевірити баланс акаунту в застосунку MetaMask (рис. 65).

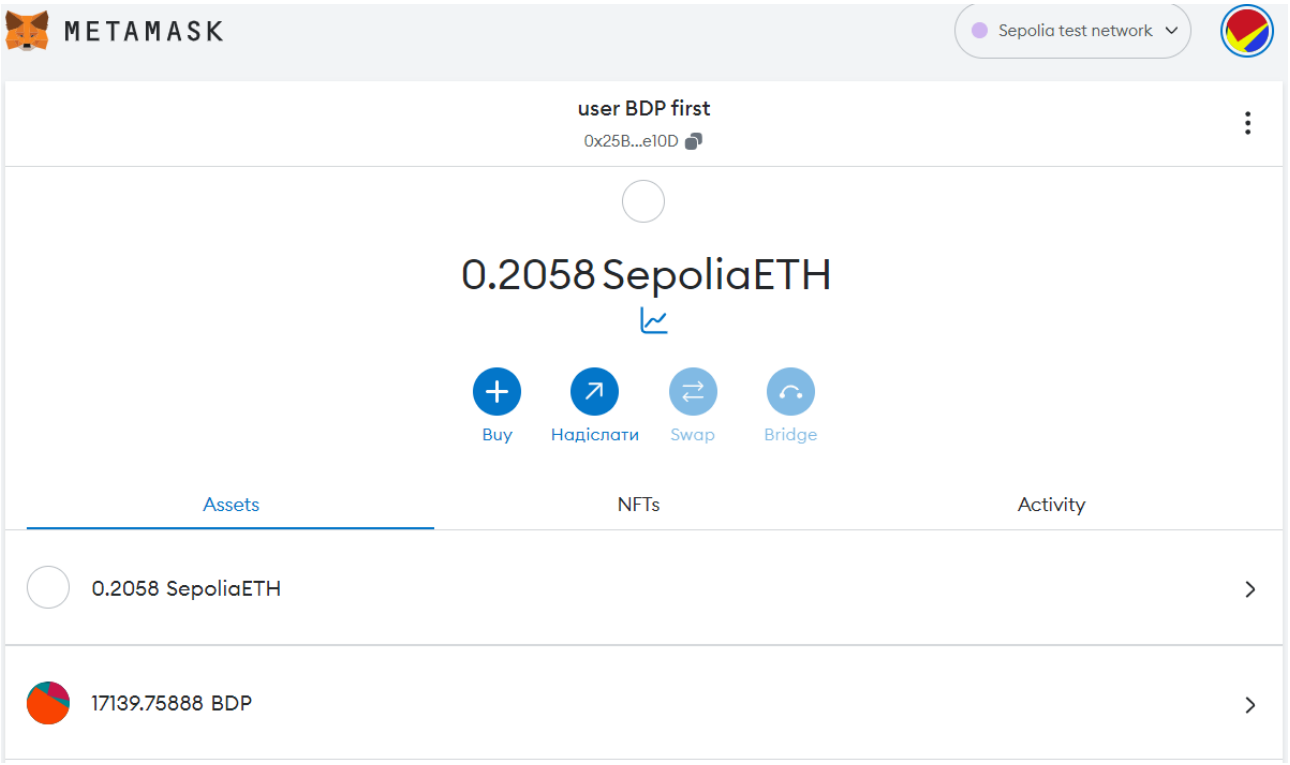

Рисунок 65 – Перевірка результатів операції *Buy* в MetaMask

Важливо зазначити, що витрачений ETH повинен бути надісланий на баланс контракту токена BDP. На рис. 66 зображено загальну кількість на його рахунку, де до цього знаходилось 0.025 ETH та було додано ще 0.15 ETH.

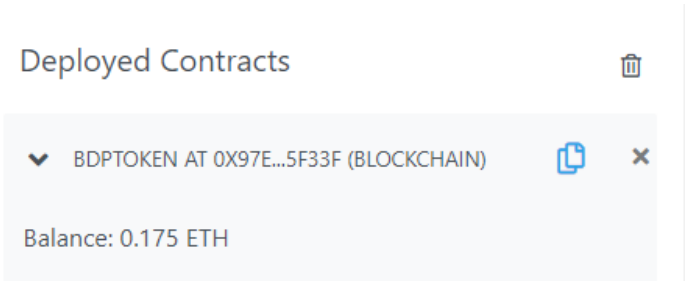

Рисунок 66 – Баланс контракту BDP токена після операції *Buy*

Подальше тестування виконується на вкладці *Sell* (продаж). На рис. 67 відбувається продаж зі списанням комісії, на ньому спостерігається поле *Amount of BDP* (кількість BDP токенів) та прапорець *Without the commission*

(встановлення продажу без оплати комісії, яка списується в токенах ETH). Виконання операцій проводиться на тому самому обліковому записі (first_bdp), що й купівля токенів.

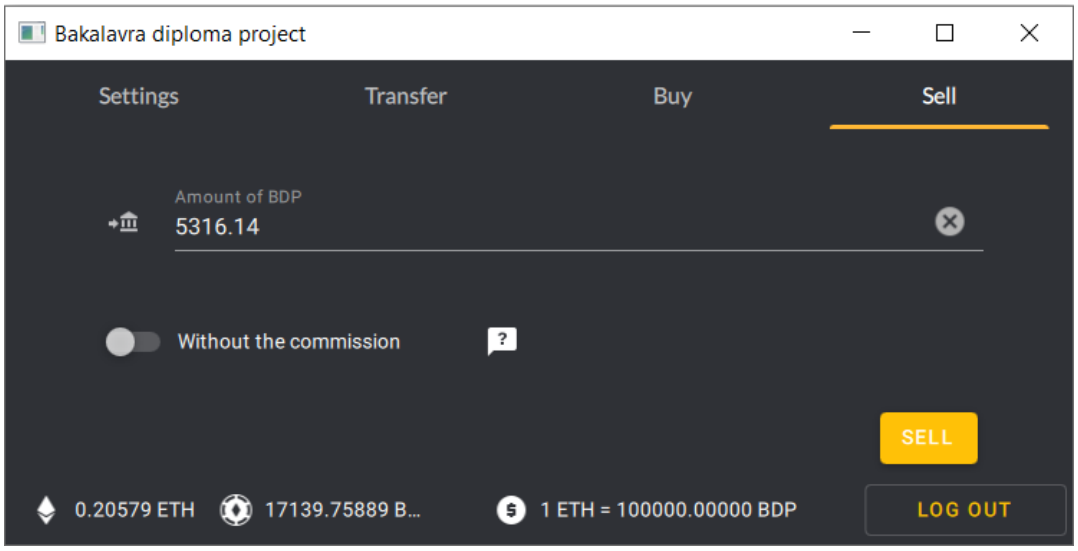

Рисунок 67 – Операція *Sell*

На рис. 68 представлено вдале виконання операції продажу криптовалюти зі списанням комісії. Враховуючи курс обміну, значення ETH збільшено правильно, але нарахована кількість не являється ідеально точною по курсу, тому що було використано газ для здійснення транзакції. Щодо проданих токенів, то їх сума відповідає дійсності, адже 17139.75 - 5316.14 = 11823.61 BDP.

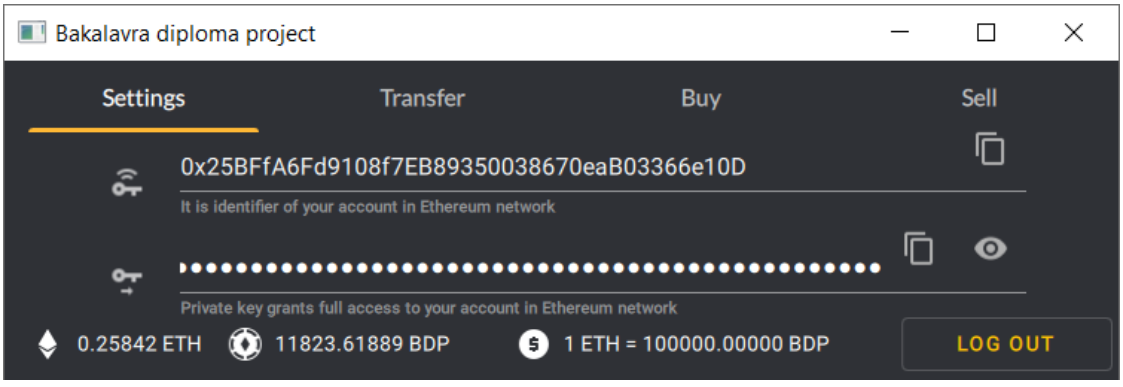

Рисунок 68 – Результати операції *Sell* зі списанням комісії

Сформовану та відправлену транзакцію в мережу Ethereum можна переглянути на веб-сервері (рис. 69).

```
Miners perform a transaction...
https://sepolia.etherscan.io/tx/0xb9f0d976003b62a1d635ac666619e547680c154dcf5c25317e8575b5f98028a3
Block is mined in 3628499
```

Рисунок 69 – Посилання на транзакцію операції *Sell*

На рис. 70 зображено акаунт того самого користувача в застосунку MetaMask, щоб переконатись в збереженні змінених даних у мережі Ethereum.

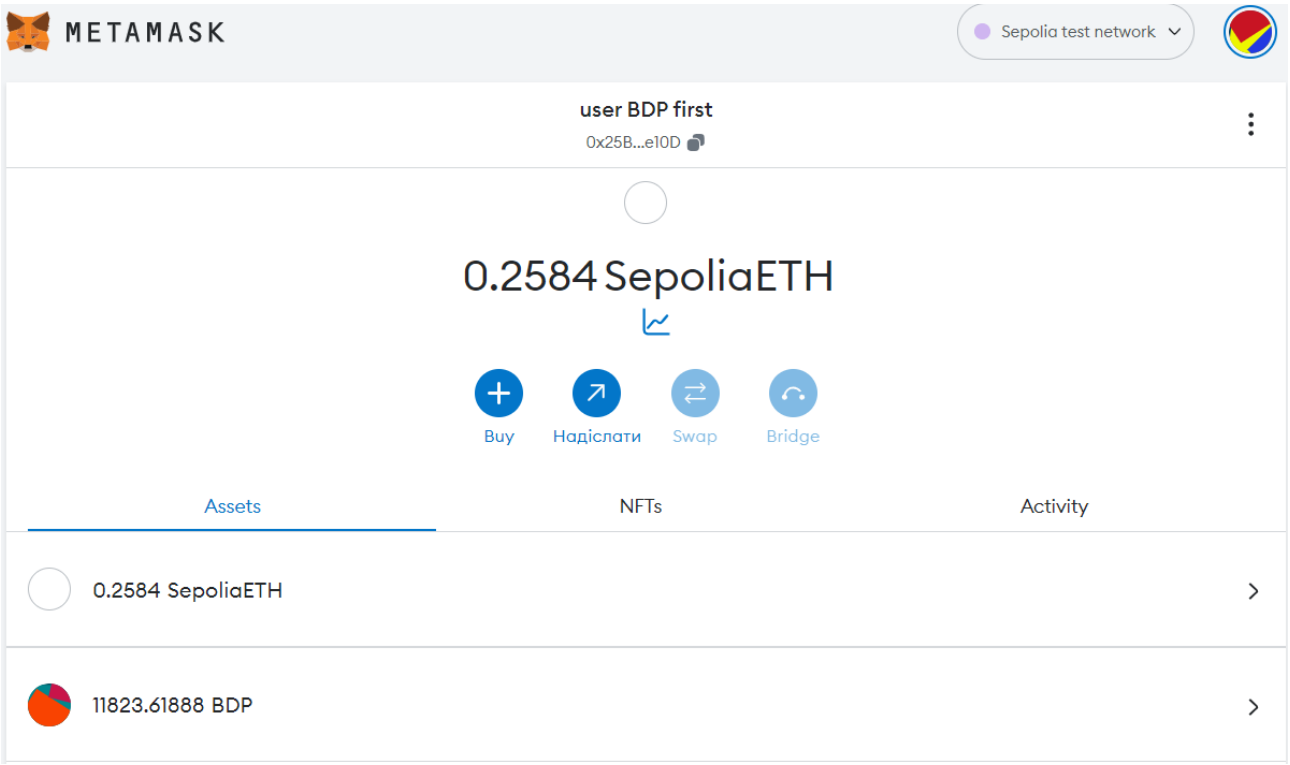

Рисунок 70 – Перевірка результатів операції *Sell* (з комісією) в MetaMask

Як і у випадку купівлі криптовалюти, продаж також взаємодіє з рахунком контракту токена BDP. На рис. 71 представлено решту, що залишилась після виконання операції *Sell*, оскільки комісію оплачує користувач, то на балансі контракту чітко отримано вирахувану кількість ETH. Враховуючи, що після

операції *Buy* на рахунку встановилось 0.175 ETH, а по курсу продаж токенів обійшовся в 0.0531614 ETH. Виходить, що 0.175 - 0.0531614 = 0.1218386 ETH.

Рисунок 71 – Баланс контракту BDP токена після операції *Sell* (з комісією)

Залишилась остання операція для тестування – це продаж токенів без оплати комісії користувачем. На рис. 72 зображено вкладку для виконання (*Sell*), де прапорець *Without the commission* активований. Операція виконується на тому ж самому обліковому записі (first_bdp).

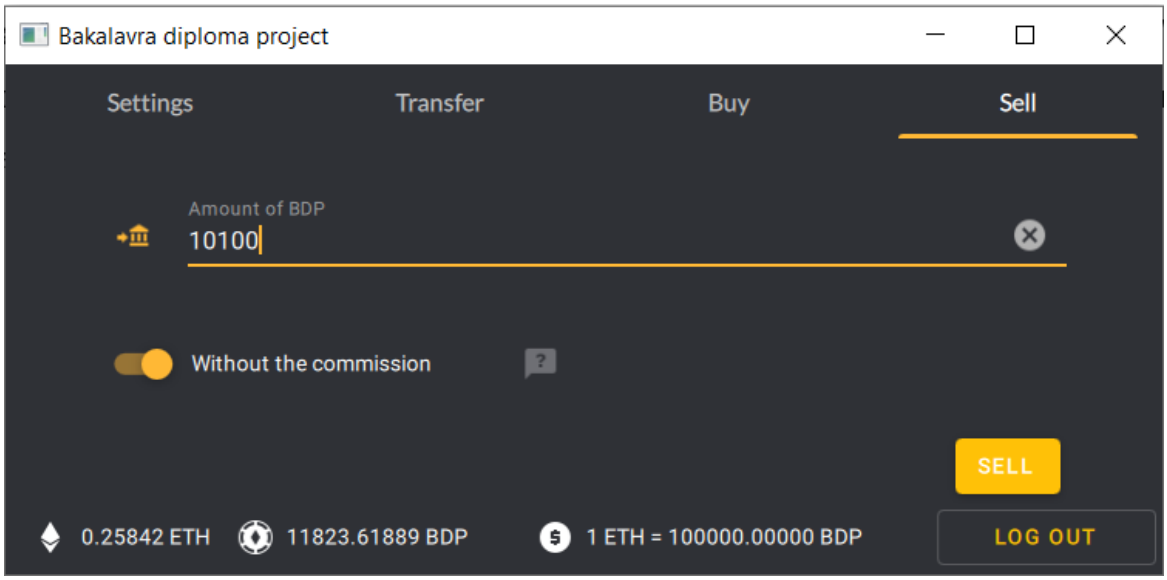

Рисунок 72 – Операція *Sell* без оплати комісії

На рис. 73 спостерігається оновлена кількість криптовалюти на балансі після продажу BDP токенів. По курсу обміну 10100 BDP = 0.101 ETH. Якщо

перевіряти рахунок, то можна помітити, що рівно в таких кількостях було додано користувачу ETH та віднято BDP токенів.

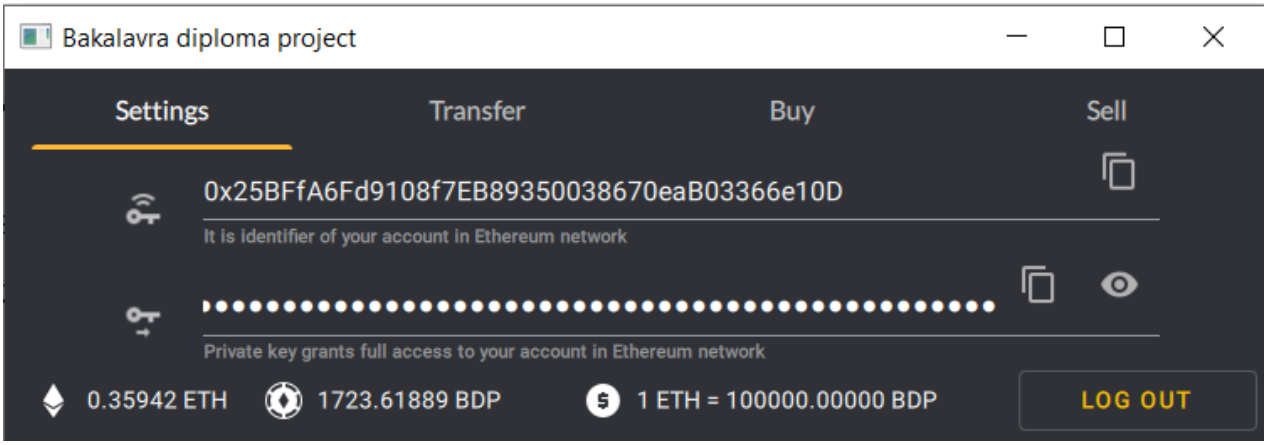

Рисунок 73 – Результати операції *Sell* без оплати комісії

Щодо переконання в тому, що транзакція дійсно була збережена у мережі, знову можна звернутись до MetaMask та переглянути рахунки акаунту (рис. 74).

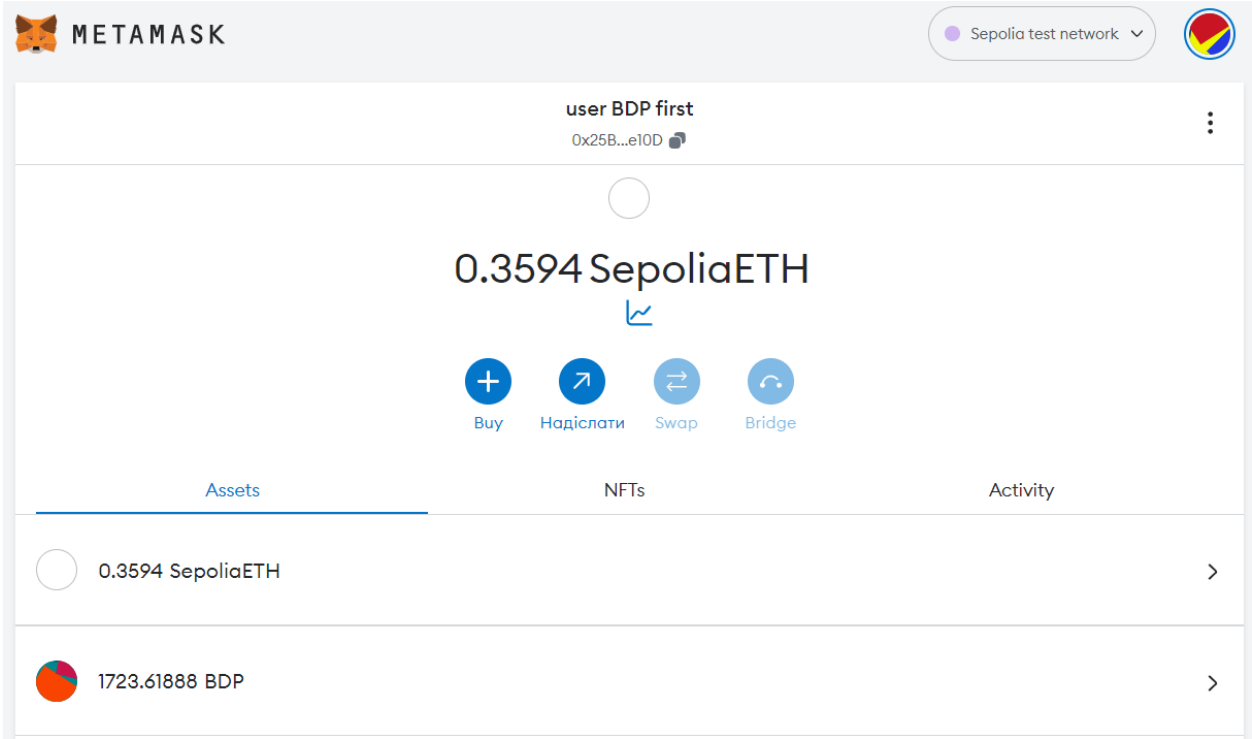

Рисунок 74 – Перевірка результатів операції *Sell* (без комісії) в MetaMask

Нова оброблена транзакція веб-сервером зображена на рис. 75, де можна знайти посилання на неї в мережі.

```
Sending...
Miners perform a transaction...
https://sepolia.etherscan.io/tx/0x86a0a77cf110b790f84368de413dceedfc5e1aa7f02ac0aa3db44a23209c3453
Block is mined in 3629248
```

Рисунок 75 – Посилання на транзакцію операції Sell (без комісії)

Щодо балансу ETH на контракті токена, то все виконується ідентично, що й у продажі з оплатою комісії. На рис. 76 представлено оновлений рахунок, де 0.1218386 - 0.101 = 0.0208386 ETH. Таким чином, комісія повинна бути оплачена власником контракту, тому що з балансу користувача та контракту токена нічого зайвого не списано.

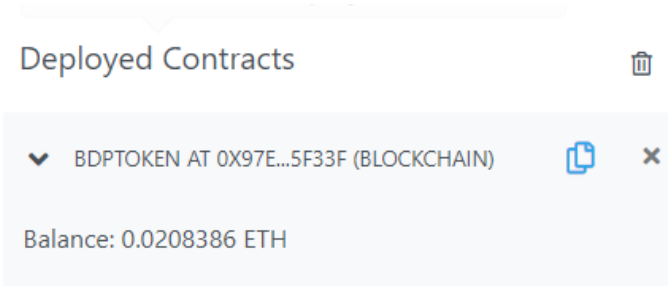

Рисунок 76 – Баланс контракту BDP токена після операції *Sell* (без комісії)

Отже, тестування функціоналу системи для керування криптовалютою виконано успішно, помилок не виявлено, всі сценарії здійснення операцій перевірені та кожна окрема частина програми працює правильно.

Також, якщо переглянути вміст файлу, який слугує обліковим записом користувачу, то можна впевнитись, що PK вкрасти неможливо, адже він надійно зберігається в зашифрованому вигляді. На рис. 77 зображено приклад такого акаунту, де даний обліковий запис (first_bdp) в основному використовувався для тестування функціональності програми в підрозділі.

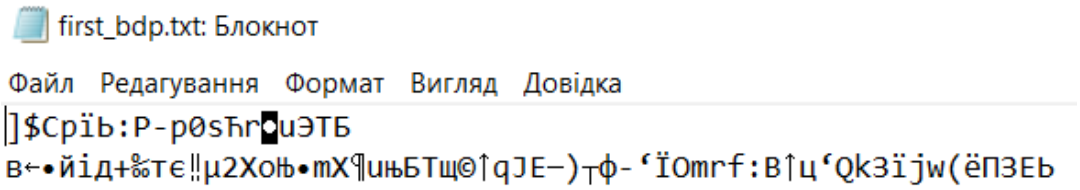

Рисунок 77 – Файл облікового запису

3.3 Оцінка продуктивності та швидкодії системи

Даний підрозділ призначений для виведення оцінки розробленій системі. Необхідно проаналізувати програму на ефективність, визначити часові показники по виконанню транзакцій та перевірити зручність для публічного використання тощо.

Якщо вимірювати швидкість виконання транзакції для наявних операцій з криптовалютою в застосунку, то вона оброблюється досить швидко та ефективно, що в основному займає не більше 10 секунд на підтвердження транзакції.

Щодо масштабованості, то робота системи забезпечує надійний безперебійний зв’язок при збільшенні кількості користувачів.

Використання газу залежить від завантаженості мережі Ethereum. Якщо надходить значна кількість транзакцій на виконання, то й відповідно комісія буде дещо збільшена.

Система не зберігає ніяких додаткових даних користувача, вся його інформація знаходиться в блокчейні, доступ до якого відбувається швидко за допомогою приватного ключа в обліковому записі, який надійно захищений з використанням алгоритмів шифрування даних.

Таким чином, розроблений проєкт оснащений базовим функціоналом для керування криптовалютою, забезпечує швидке реагування програми, безперебійну обробку транзакцій, сучасний та приємний інтерфейс користувача.

ВИСНОВКИ

Програмна система для керування власноруч випущеною криптовалютою з використанням технології блокчейн, яка є результатом дипломного проєкту, створена для зберігання реалізованого токена на рахунку користувача та має функціонал з можливістю здійснення базових операцій з криптовалютою, таких як перегляд балансу, купівля, продаж та переказ токенів, де останні дві опції можуть виконуватись без оплати комісії за транзакцію.

Після вивчення теоретичного матеріалу та аналізу подібних систем для керування криптовалютою, що досліджено в першому розділі дипломного проєкту, визначено доцільність розробки такої системи, тому що вона забезпечена більш розширеною функціональністю у вигляді виконання транзакцій без оплати комісії, що є відсутнім у всіх інших аналогів.

Розроблена програмна система складається з декількох незалежних частин, що описані вже в другому розділі проєкту, вони відповідають за свої конкретні задачі, які разом забезпечують повноцінне виконання операцій з криптовалютою, та можуть бути розширені додатковими функціями.

Провівши тестування, яке представлено в третьому розділі пояснювальної записки, можна підсумувати, що поставлена задача виконана та мета роботи досягнута. Результати перевірки функціональності системи повністю підтверджують її працездатність та відповідність визначеним вимогам.

Основні реалізовані функції програмної системи:

- безпечне зберігання створеного токена;
- переказ, продаж, купівля та перегляд балансу криптовалюти;
- можливість не оплачувати комісію за переказ та продаж токенів;
- сучасний інтерфейс користувача;
- шифрування приватного ключа паролем;
- створення, авторизація та імпорт акаунту.

Отже, розроблена програмна система для керування власноруч випущеною криптовалютою з використанням технології блокчейн являється вдалим результатом виконання дипломного проєкту. Вона має гарний потенціал для подальшого розвитку, розширення та вдосконалення різноманітних інструментів. Наприклад, токен можна інтегрувати не тільки з блокчейном Ethereum, та з іншими ефективними, що користуються попитом, криптовалютними мережами. Також, дана розробка може бути цінною для впровадження інноваційних рішень в сфері криптовалютних технологій.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction / Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, Steven Goldfeder, Princeton University Press, July 19, 2016. 336 с.
2. D. W. Kravitz, J. Cooper. Securing user identity and transactions symbiotically: IoT meets blockchain / D. W. Kravitz, J. Cooper. // 2017 Global Internet of Things Summit (GIoTS). – 2017.
3. ERC20 token standard : URL: https://ethereum.org/uk/developers/docs/standards/tokens/erc-20/ (дата звернення: 13.03.2023).
4. Ethereum Virtual Machine (EVM) : URL: https://ethereum.org/uk/developers/docs/evm/ (дата звернення: 13.03.2023).
5. Bogner A, Chanson M, Meeuw A. A decentralised sharing app running a smart contract on the ethereum blockchain. In Proceedings of the 6th International Conference on the Internet of Things 2016 Nov 7. - p. 160-178.
6. Ethereum accounts : URL: https://ethereum.org/uk/developers/docs/accounts/ (дата звернення: 14.03.2023).
7. Transactions : URL: https://ethereum.org/uk/developers/docs/transactions/ (дата звернення: 14.03.2023).
8. Gas and fees : URL: https://ethereum.org/uk/developers/docs/gas/ (дата звернення: 15.03.2023).
9. MetaMask : URL: https://metamask.io/ (дата звернення: 16.03.2023).
10. Solidity language : URL: https://docs.soliditylang.org/en/v0.8.20/ (дата звернення: 16.03.2023).
11. Ethers.js : URL: https://docs.ethers.org/v5/ (дата звернення: 14.04.2023).
12. Web3.js : URL: https://web3js.readthedocs.io/en/v1.2.1/ (дата звернення: 14.04.2023).
13. Remix Ethereum IDE : URL: https://remix-ide.readthedocs.io/en/latest/ (дата звернення: 05.05.2023).
14. Qaterial. Collection of Material Components : URL: https://github.com/OlivierLDff/Qaterial (дата звернення: 15.03.2023).
15. Advanced Encryption Standard (AES) : URL: https://www.geeksforgeeks.org/advanced-encryption-standard-aes/ (дата звернення: 15.03.2023).
16. Qt framework : URL: https://doc.qt.io/ (дата звернення: 10.04.2023).
17. Node.js : URL: https://nodejs.org/en/docs (дата звернення: 21.04.2023).
18. EIP-3009 : URL: https://github.com/CoinbaseStablecoin/eip-3009 (дата звернення: 08.04.2023).
19. npm Docs : URL: https://docs.npmjs.com/ (дата звернення: 22.04.2023).