Одеський національний університет імені І.І.Мечникова
(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних систем та технологій
(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

на здобуття ступеня вищої освіти «бакалавр»
(освітньо-кваліфікаційний рівень)

Використання блокчейн технологій у месенджерах
(назва українською)

Using blockchain technologies in messengers
(назва англійською)

Виконав: студент денної форми навчання
спеціальності 123 Комп'ютерна інженерія.
(шифр і назва напряму підготовки, спеціальності)
Бойченко Владислав Сергійович
(прізвище, ім'я, по-батькові)

Керівник ст. викл. Берков Ю.М.
(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент к.ф.-м.н., доцент Шугайло Ю.Б.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№___ від «___»_____ 2022 р.
Завідувач кафедри
_____ Ю.О. Гунченко
(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК № ___
протокол № __ від «___» ____ 2022 р.
Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)
Голова ЕК
_____ Н.Ф. Казакова
(підпис) (прізвище, ініціали)

Одеса - 2022

АНОТАЦІЯ

Метою роботи є розробка та створення прототипу системи для взаємодії користувача з блокчейном через сучасний месенджер. Для чого був проведений аналіз існуючих і широко використовуваних в даний час месенджерів, у яких можливо реалізувати даний проект. Зокрема розглянуті різні сучасні месенджери як Telegram, WhatsApp, Viber. Окремо було розроблено Smart-Contract для виконання транзакцій в мережі блокчейн. В якості мережі блокчейн було обрано Avalanche C-Chain, для написання інтерфейсу користувача було обрано мову програмування JS та фреймворк на її основі VueJS. В результаті сформульовані основні функції проектованої системи. На підставі цього побудована схема взаємодії користувача с блокчейном та реалізований програмний код для її виконання. Побудову схеми взаємодії користувача с блокчейном проведено в програмі MermaId. Для забезпечення функціонування системи було розроблено керуючий код, з застосуванням середовища програмування WebStorm.

Особливістю розробленої є те, що для взаємодії з блокчейном користувачеві не потрібен браузер та знаходити Web інтерфейс, а достатньо скористуватися месенджером та одразу почати взаємодіяти з блокчейном.

Ключові слова: Блокчейн, Telegram, Smart-Contract, JS, децентралізація

ANNOTATION

The aim of the work is to develop and create a prototype system for user interaction with the blockchain through a modern messenger. For what was the analysis of existing and widely used currently used messengers in which it is possible to implement this project. Separately, Smart-Contract was developed to perform transactions on the blockchain network. As a result, the main functions of the projected system are formulated. The construction of a user interaction scheme with the blockchain was carried out in the MermaId program. To ensure the functioning of the system, a control code was developed using the WebStorm programming environment.

The peculiarity of the developed one is that in order to interact with the blockchain, the user does not need a browser and find a Web interface, but it is enough to use the messenger and immediately begin to interact with the blockchain.

Key words: Blockchain, Telegram, Smart-Contract, JS, decentralisation

ЗМІСТ

Стор.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

SC – Smart-Contract - різновид угоди в формі закодованих математичних алгоритмів, укладення, зміна, виконання і розірвання яких можливі лише з використанням комп'ютерних програм (блокчейн платформ) в рамках мережі Інтернет.

JS – JavaScript - динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript.

HTTP – протокол передачі даних, що використовується в комп'ютерних мережах. Назва скорочена від HyperText Transfer Protocol, протокол передачі гіпертекстових документів. HTTP належить до протоколів моделі OSI 7-го прикладного рівня.

RPC – Виклик віддалених процедур (Remote procedure call) — протокол, що дозволяє програмі, запущеній на одному комп'ютері, звертатись до функцій (процедур) програми, що виконується на іншому комп'ютері, подібно до того, як програма звертається до власних локальних функцій.

EVM – Віртуальна машина Ethereum – це глобальний комп'ютер, який може використовувати будь-яка людина за невелику комісію, яка виплачується в ефірі.

ERC20 Токен – це популярний стандарт для смарт-контрактів на блокчейні Ethereum. Являє собою набір правил, які слід слідувати розробки контракту, відповідального параметри і емісію нового користувальницького токена.

DEFI – це нова фінансова технологія, заснована на безпечних розподілених системах блокчейн, подібних до тих, що використовуються криптовалютами. Система усуває контроль банків та установ над грошима, фінансовими продуктами та фінансовими послугами.

NFT – це унікальні токени блокчейна, які означають право власності на актив.

DAPP – децентралізовані сервер-клієнтські додатки, які працюють у децентралізованих системах на зразок Ethereum

APR – Річна відсоткова ставка відноситься до річних відсотків, отриманих від суми, яка стягується з позичальників або виплачується інвесторам. APR виражається у відсотках, який представляє фактичну річну вартість коштів протягом терміну кредиту або доходу, отриманого від інвестицій.

Майнер – Программа для генерации (майнинга) криптовалюты та обробки транзакцій.

Транзакція – (трансакція) це здійснення закінчених дій стосовно визначеного об'єкта, що переводить цей об'єкт з одного поточного стану в інший.

Крипто гаманець – це програма чи засіб, який дозволяє зберігати криптовалюту. Фізично гаманець не містить ніяких монет чи токенів — замість цього в ньому зберігаються секретні коди, що дають вам право користуватися вашою криптовалютою.

Solidity – об'єктно-орієнтована та предметно-орієнтована мова програмування розумних контрактів для платформи Ethereum.

Ethereum – криптовалюта та платформа для створення децентралізованих онлайн-сервісів (DAPPs) на базі блокчейна, що працюють на базі розумних контрактів. Реалізована як єдина децентралізована віртуальна машина.

WalletConnect – це відкритий протокол для безпечного зв'язку між гаманцями та Dapps за допомоги web3.

ВСТУП

З розвитком та розповсюдженням технологій в суспільстві зростає попит на розподілене програмне забезпечення. Блокчейн-технології сьогодні стають дедалі популярнішими. Найбільш відома завдяки фінансовій сфері, технологія розподілених реєстрів сьогодні використовується в медицині, на виборах та в інших різних областях. Це можливо завдяки перевагам, які дає використання блокчейна, таким як безпека та прозорість. Сьогодні ця технологія використовується і в освіті. Зокрема, дедалі популярніша тема видачі цифрових дипломів, заснованих на блокчейні.

У фінансовій сфері є багато прикладів для використання блокчейна в основному для інвестування, так як дана технологія має багато переваг. Одні з ключових – перевірюваність та незмінність. Один раз написаний SC не може бути змінений, а кожен бажаючий може переглянути його вихідний код, щоб переконатися які саме дії можуть бути виконані завдяки ньому. А це дає перевагу в тому, що всі проінвестовані кошти будуть використані як передбачалося користувачем. Для цього немає потреби не довіряти SC, так як один і той самий код буде однаково працювати у всіх користувачів. Достатньо відкрити інтерфейс для користувача, щоб отримати актуальну інформацію. Це дозволяє швидко отримувати актуальну та достовірну інформацію. Крім того, блокчейн завдяки своїй децентралізації унеможливлює підробку даних. Принципи, на яких ґрунтується технологія розподіленого реєстру, не дають жодної можливості непомітно виправити дані користувача. Це дозволяє значно знизити рівень шахрайства в фінансовій сфері.

Процес створення будь якого програмного забезпечення для використання технологій блокчейну залежить від вибору конкретного типу блокчейн мереж та мови програмування. Коли всі вимоги зібрані, можна почати розробляти структуру проекту та почати описувати її складові.

Метою роботи є розробка та створення прототипу сайту для виконання транзакцій в мережі блокчейн за допомогою сучасного месенджера.

Завданнями роботи є огляд предметної області, формулювання функціональних вимог до програмного забезпечення, вибір мереж, створення прототипа SC для реєстрування дій в блокчейні та вибір месенджера для інтеграції, що задовольняють сформульованим вимогам, розробка та налагодження програмного коду.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

Існує велика кількість взаємодії з блокчейном в залежності від потреб, вимог задачі та обраної блокчейн мережі.

Одна з можливих причин використання даного прототипу є збільшення кількості користувачів розподіленими системами та забезпечення для зручного користування. Розглянемо певну схему прототипу взаємодії з розподіленими блокчейн системами в сучасних месенджерах.

Для взаємодії з блокчейн мережами часто використовують Web технології. JS дозволяє використовувати HTTP запити, а вони в свою чергу дозволяють робити запити в блокчейн мережі по технології RPC. Розглянемо компоненти прототипу для виконання роботи (табл. 1.1).

Таблиця 1.1 – Компоненти прототипу для взаємодії з блокчейн мережами через сучасні месенджери

Компонент	Значення
Блокчейн мережа на основі EVM - Avalanche	Блокчейн мережа для виконання та обробки запитів за допомогою
Месенджер Telegram	Сучасний месенджер з підтримкою ботів
Інтерфейс користувача	WEB частина яка буде відповідати за відображення та взаємодію з блокчейном
Крипто гаманець	Потрібен для ідентифікації користувача за допомогою асиметричного шифрування

Ця система пропонує ряд функцій:

1) Staking - Перерахування ERC20 токенів на SC, що не дає змогу надалі користуватися ними, але за час, коли вони будуть заблоковані, будуть нараховуватися нагороди.
2) Unstaking - Повернення своїх коштів на свій рахунок
3) Отримання нагород - нарахування на рахунок користувача нагороди фіксований відсоток в залежності від часу та суми утримання ERC20 токенів на рахунку SC.
4) Перевірка нарахованих нагород за поточний відрізок часу(у секундах)
5) Перегляд інформації на взаємодія з блокчейном через месенджер

Розглянувши систему можна виділити дуже зручні та вдалі функції прототипу для взаємодії з розподіленою системою блокчейном. Наприклад інтеграція різноманітної логіки взаємодії з блокчейном с месенджера.

Крім стэйкінга ERC20 Токенів можна використовувати подібним чином систему різну бізнес логіку як для DEFI систем так і для систем NFT.

Існує велика кількість проектів на розподіленої технології блокчейн які можна теж інтегрувати в подібну систему взаємодії з розподіленою системою блокчейн для підвищення кількості користувачів.

Ще однією цікавою перевагою даної системи є її використання з різними системами блокчейна без суттєвої зміни коду, що надає перевагу в масштабування комерційних та некомерційних проектів.

Подібна система не потребує додаткових трудовитрат для взаємодії, так к використовую всі знайомі користувачеві технології для роботи з блокчейном.

2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Постановка задачі

Після дослідження предметної області, огляду існуючих технологій для взаємодії з розподіленою системою блокчейном визначимо загальні задачі для реалізації дипломного проекту. Для системи використання блокчейн технологій в сучасних месенджерах було обрано наступні функції:

1) Отримання даних користувача в блокчейн мережі
    1) Можливість запиту на отримання поточного стану рахунку користувача а також результати минулих взаємодій з блокчейном;
2) Можливість під’єднання крипто гаманця:
    1) Під’єднання крипто гаманця користувача для подальших здійснення транзакцій в блокчейні;
3) Можливість взаємодії з блокчейн мережею:
    1) Перерахування ERC20 Токенів на SC для подальшого отримання нагород;
    2) В разі необхідності виконання транзакції для довіри SC на користування ERC20 Токенами поточним SC;
    3) Вивід ERC20 Токенів зі SC які були раніше перераховані;
    4) Вивід нагороди в ERC20 Токенів, які буди нараховані за утримання коштів на SC;
4) Можливість виконання всіх взаємодій через сучасний месенджер;

Приведений набір функцій дозволить користувачу розширити способи використання блокчейна. Основною функцією є взаємодія з блокчейн мережею за допомогою сучасного месенджера. Функції взаємодії можуть бути корисними для швидкої перевірки стану користувача в блокчейн мережі.

2.2 Огляд компонентів та створення системи

Наступною важливою частиною проектування є вибір компонентів майбутньої системи та побудова схеми взаємодії обраних компонентів. Побудову схеми системи проведено в програмі MermaId(для побудови діаграм та схем).

Вибір конкретних компонентів має велике значення для побудови схеми та взаємодії компонентів. Для даної системи було обрано сучасний месенджер Telegram, блокчейн мережу Avalanche, та DAPP було розроблено на сучасній мові програмування JS з використанням сучасного фреймворку VUE.JS (рис. 2.1). Обраний набір компонентів найчастіше використовується для в сучасній розробці додатків на основі блокчейна.

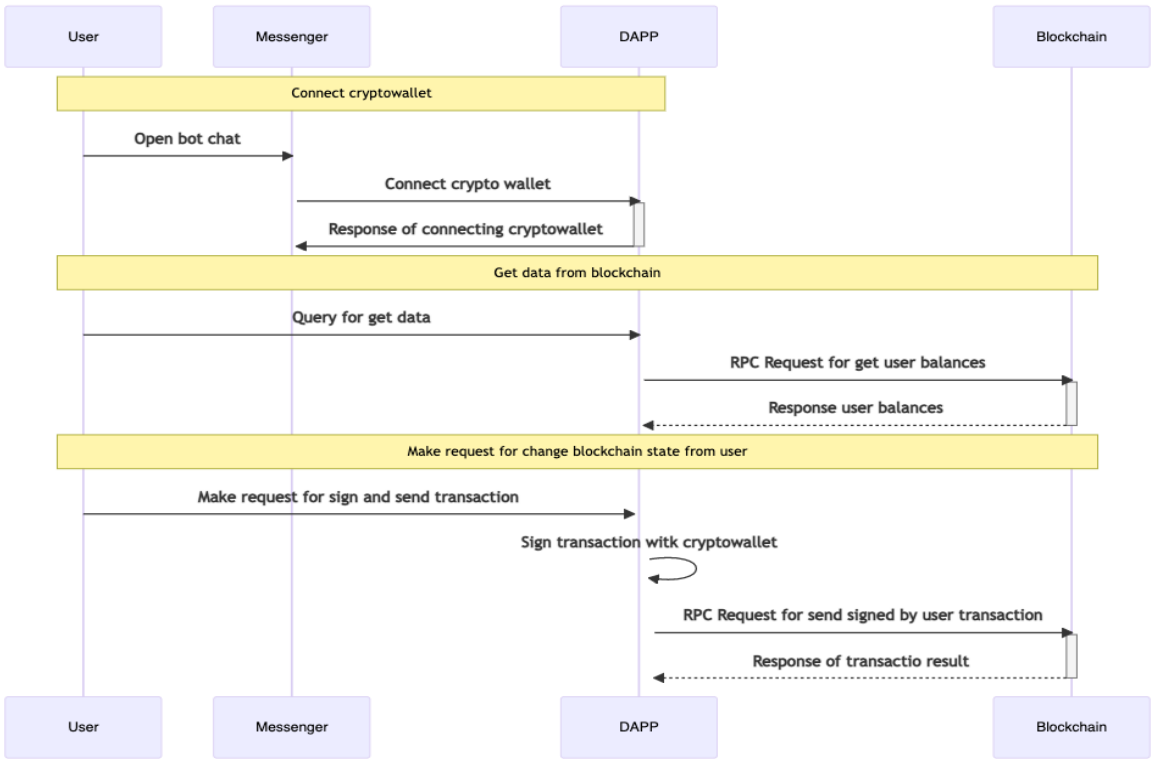

Рисунок 2.1 – Схема взаємодії компонентів системи

Найголовнішим компонентом проекту є блокчейн Avalanche, який на своєму боці обробляє всі заклики користувача на отримання поточних даних а також запис нових за логікою описаною в SC.

Компонент DAPP відповідає за приємне відображення для користувача інтерфейсу для взаємодії, а також відповідає за зручну взаємодію зі блокчейном за допомогою відображення потрібних даних, роблячи певні запити до розподіленої системи блокчейн, а також за допомогою елементів інтерфейсу для виконання запитів на блокчейн для зміни його стану.

Також потрібно зазначити, що за зміну стану в блокчейні потрібно перераховувати, в залежності від того яку інформацію потрібно змінювати, певну кількість токенів блокчейна, це комісія, яку повинен сплатити користувач майнерам для обробки його транзакції. Для цього у користувача повинен бути підключений крипто гаманець, на якому, на момент виконання транзакції повинна бути достатня кількість коштів.

Запити на отримання інформації безкоштовні та не потребують оплати від користувача, також крім простого отримання інформації дозволяється викликати функції SC теж безкоштовно, але при умові, що ці функції після своєї роботи не змінюють стану блокчейна.

3 РОЗРОБКА СИСТЕМИ

Фінальною частиною розробки є написання програмного коду, що забезпечує функціонування системи. Написання програмного коду відбуватиметься в середі програмування WebStorm. Описавши взаємодію усіх компонентів системи, можна приступати до написання коду для різних його компонентів, а саме коду логіки для SC, налаштування Telegram бота та написати проект прототип для Web частини на JS використовуючи сучасний фреймворк VUE.JS.

3.1 Опис коду для компоненті SC

Почнемо з коду SC на мові програмування Solidity, так як спираючись на нього будуть підлаштовуватись наступні компоненти системи. Для початку визначаємося з основними функціями(табл. 3.1) всі інші функції є приватними та виконують призначення для математичних обчислень.

Основна ідея даної бізнес логіки це нарахування фіксованого проценту річних (APR) на суму ERC20 Токенів, які перерахував користувач для подальшої виплати нагород. В бізнесі цей метод нагород має за мету стимулювання користувачів блокувати визначений ERC20 Токен для зменшення його волатильності на ринку.

Таблиця 3.1 – Функції SC та їх значення

Функція	Опис
stake	Потрібна для перерахування коштів від користувача до поточного SC, а також збереження кількості для подальшого повернення ERC20 Токенів користувачеві і нарахування відсотків як нагороду.
unstake	Потрібна для знаття ERC20 Токенів зі SC на рахунок користувача кількості, яку він вкаже, але не більше ніж стільки, скільки він раніше перерахував.
withdrawRewards	Потрібна для зняття нагороди на рахунок користувача в ERC20 Токені зі спеціального аккаунту, на якому знаходиться баланс ERC20 Токену для видачі нагород усім користувачам
getUserRewards	Потрібна для отримання значення, скільки на поточний час користувач може отримати нагороди якщо наважиться
getUserData	Потрібна для отримання значень скільки користувач на даний момент перерахував коштів на адресу SC

Розробка даного програмного коду велася на останній версії Solidity 0.8.14.

```
    contract FixedStaking is ReentrancyGuard {
            using Address for address;
            using SafeERC20 for IERC20;

        struct UserData {
            uint256 balance;
            uint256 storedReward;
            uint256 lastStakedTime;
        }
        uint256 constant SECONDS_IN_YEAR = 31556952;
        uint256 percentPreccision = 1000000; // 100%
        uint256 rewardPercent = 150000; // 15% (1% = 10000)
        address treasury =
0xC6BdFA7e694db5621CdcB7242d1931FC586f6d1d;
        address stakingToken =
0xa2c854071fd85a3880319EAb04De1a25271042C2;
        address rewardToken =
0xa2c854071fd85a3880319EAb04De1a25271042C2;
        mapping (address => UserData) userData;
        mapping (address => uint256) savedReward;
        uint256 totalPaidRewards;

        constructor(
            address _stakingToken,
            address _rewardToken,
            address _treasury
            ) {
            require(_stakingToken.isContract(), "Staking:
staking token is not a contract address");
            require(_rewardToken.isContract(), "Staking: reward
token is not a contract address");
            require(_treasury != address(0), "Staking: treasury
is zero address");
            stakingToken = _stakingToken;
            rewardToken = _rewardToken;
            treasury =_treasury;
        }
```

```
        function getUserRewards(address user) public view
returns(uint256 amount){
            return _getUserRewards(user) +
userData[msg.sender].storedReward;
        }

        function getUserData(address user) public view
returns(UserData memory){
            return userData[user];
        }

        function stake(uint256 amount) public {

            IERC20(stakingToken).safeTransferFrom(msg.sender,
address(this), amount);
            if(userData[msg.sender].lastStakedTime == 0){
                userData[msg.sender].lastStakedTime =
block.timestamp;
            }else{
                _updateRewardsForUser(msg.sender);
            }
            userData[msg.sender].balance =
userData[msg.sender].balance + amount;
        }

        function unstake(uint256 amount) public {
            _updateRewardsForUser(msg.sender);
            userData[msg.sender].balance =
userData[msg.sender].balance - amount;
            IERC20(stakingToken).safeTransfer(msg.sender,
amount);
        }

        function withdrawRewards() public nonReentrant {
            _updateRewardsForUser(msg.sender);
            uint256 amountForWithdraw =
userData[msg.sender].storedReward;
            userData[msg.sender].storedReward = 0;
            totalPaidRewards = totalPaidRewards +
amountForWithdraw;
            IERC20(rewardToken).safeTransferFrom(treasury,
msg.sender, userData[msg.sender].storedReward);
        }

        function _updateRewardsForUser(address user) private {
```

```
            if(userData[user].lastStakedTime > 0){
                userData[user].storedReward =
userData[user].storedReward + _getUserRewards(user);
                userData[user].lastStakedTime = block.timestamp;
            }
        }
        function _getUserRewards(address user) private view
returns(uint256 amount) {
            if(userData[user].lastStakedTime == 0){
                return 0;
            }
            uint256 rewardPerSecond = userData[user].balance *
rewardPercent / percentPreccision / SECONDS_IN_YEAR;
            uint256 timeLeft = block.timestamp -
userData[user].lastStakedTime;
            amount = timeLeft * rewardPerSecond;
        }
    }
```

3.2 Опис коду для компоненті DAPP

Програмний код для DAPP виконаний за допомогою сучасної мови програмування JS та за допомогу фреймворка VUE.JS для зручної побудови Web додатків.

DAPP буде складатися з однієї сторінки, де будуть знаходитися 3 основні блоки:

1. Блок для підключення крипто гаманця за допомогою популярної і зручної системі WalletConnect
2. Блок для відображення даних користувача таких як поточний рахунок обраного ERC20 Токену, поточна кількість перерахованих на SC ERC20 Токенів, а також кількість на поточний момент нагород, котру користувач може вивести з системи.
3. Блок взаємодії за SC з наступними елементами управління:
    - Stake - для зарахування ERC20 Токенів на SC вод користувача
    - Unstake - для повернення раніше зарахованих ERC20 Токені на рахунок користувача

- Withdraw - для отримання користувачем нагород в ERC20 Токені

В даному додатку підтримується оновлення інформації без перезавантаження за рахунок реактивного оновлення елементів.

Код візуалізації першого блоку відповідає за відображення кнопки для під'єднання крипто гаманця до додатку, а саме встановлення зв'язку з гаманцем та збереження сесії входу. Код виглядає наступним чином:

```
<template>
  <el-button type="success"
@click="connectWallet">WalletConnect</el-button>
</template>

<script>
import { mapActions, mapGetters } from 'vuex';

export default {
  name: 'ConnectButton',
  computed: mapGetters([
    'connectStatus',
    'getProvider',
  ]),
  methods: mapActions([
    'connectWallet',
    'connectProvider',
  ]),
};
</script>
```

Код логіки, зберігання та обробка станів для встановлення зв'язку та збереження сесії відповідає за зручне розподілення інформації, яка пов'язана крипто гаманцем та взаємодією з ним. Також присутня логіка для налаштування підключення, а саме налаштування мережі для під'єднання крипто гаманця. Код виглядає наступним чином:

```
import WalletConnectProvider from '@walletconnect/web3-
provider';
import { Web3Provider } from '@ethersproject/providers';

// initial state
```

```
    const stateData = () => ({
      mode: process.env.NODE_ENV || 'development',
      connected: false,
      chainId: 0,
      provider: null,
      signer: null,
      account: null,
    });
    const getters = {
      connectStatus: (state) => state.connected,
      getProvider: (state) => state.provider,
      getSigner: (state) => state.signer,
      getAccount: (state) => state.account,
    };
    const actions = {
      async connectWallet({
        state,
        commit,
      }) {
        const walletConnectProvider = new
WalletConnectProvider({
          rpc: {
            43114: 'https://api.avax.network/ext/bc/C/rpc',
          },
        });

        const { connector } = walletConnectProvider;
        const accounts = await walletConnectProvider.enable();
        const provider = new
Web3Provider(walletConnectProvider);
        const signer = provider.getSigner();
        commit('setChainId', connector.chainId);
        commit('setSigner', Object.freeze(signer));
        commit('setAccount', accounts[0]);
        commit('setProvider',
Object.freeze(walletConnectProvider));
        commit('setConnect', true);
      },
    };

    // mutations
    const mutations = {
      setConnect(state, status) {
        state.connected = status;
      },
```

```
    setProvider(state, payload) {
      state.provider = payload;
    },
    setChainId(state, payload) {
      state.chainId = payload;
    },
    setSigner(state, payload) {
      state.signer = payload;
    },
    setAccount(state, payload) {
      state.account = payload;
    },

};

export default {
  state: stateData,
  getters,
  actions,
  mutations,
};
```

Код візуалізації даних користувача таких як поточний рахунок обраного ERC20 Токену, поточна кількість перерахованих на SC ERC20 Токенів, а також кількість на поточний момент нагород, котру користувач може вивести з системи. Усі актуальні дані з блокчейна отримуються автоматично та оновлюються на сторінці. Код виглядає наступним чином:

```
<div class="info">
      <div class="item">
        <span class="title">APR</span>
        <div class="valueBlock">
          <span class="value">15</span>
          <span class="symbol">%</span>
        </div>
      </div>
      <div class="item">
        <span class="title">Balance</span>
        <div class="valueBlock">
          <span class="value">{{balance}}</span>
          <span class="symbol">{{ getTokenInfo.symbol
}}</span>
        </div>
```

```
          </div>
          <div class="item">
            <span class="title">Staked</span>
            <div class="valueBlock">
              <span class="value">{{ staked }}</span>
              <span class="symbol">ONU</span>
            </div>
          </div>
          <div class="item">
            <span class="title">Rewards</span>
            <div class="valueBlock">
              <span class="value">{{ rewards }}</span>
              <span class="symbol">ONU</span>
            </div>
          </div>
        </div>
```

Код логіки, зберігання та обробка станів для отримання актуальної інформації користувача з блокчейну, а також стан за замовченням поки не будуть отриманні актуальні дані. Код виглядає наступним чином:

```
    export default new Vuex.Store({
      state: {
        stakingContractAddress:
'0xCB9cE4D38370D78360C9496De01ED68966A93e15',
        stakeTokenAddress:
'0xa2c854071fd85a3880319EAb04De1a25271042C2',
        userInfo: {
          balance: '0',
          allowance: '0',
          staked: '0',
          rewards: '0',
        },
        userInfoUpdated: new Date().getTime(),
        tokenInfo: {
          name: '',
          symbol: '',
          decimals: 18,
        },
      },
      getters: {
        getStakingContractAddress: (state) =>
state.stakingContractAddress,
```

```
        getStakeTokenAddress: (state) =>
state.stakeTokenAddress,
        getUserInfo: (state) => state.userInfo,
        getAllowance: (state) => state.userInfo.allowance,
        getBalance: (state) => state.userInfo.balance,
        getTokenInfo: (state) => state.tokenInfo,
        getUserUpdated: (state) => state.userInfoUpdated,
      },
      mutations: {
        setAllowance: (state, payload) => {
          state.userInfo.allowance = payload;
        },
        setBalance: (state, payload) => {
          state.userInfo.balance = payload;
        },
        setUserUpdated: (state) => {
          state.userInfoUpdated = new Date().getTime();
        },
        setUserInfo: (state, payload) => {
          state.userInfo = { ...state.userInfo, ...payload };
          state.userInfoUpdated = new Date().getTime();
        },
        setUserStaked: (state, payload) => {
          state.userInfo.staked = payload;
          state.userInfoUpdated = new Date().getTime();
        },
        setUserRewards: (state, payload) => {
          state.userInfo.rewards = payload;
          state.userInfoUpdated = new Date().getTime() + 1;
        },
        setTokenInfo: (state, payload) => {
          state.tokenInfo = payload;
        },
      },
      modules: {
        wallet,
        transactions,
      },
      strict: debug,
      plugins: debug ? [createLogger()] : [],
    });
```

Код візуалізації блоку для взаємодії з блокчейном за допомогою кнопок для виклику певного функціоналу, а саме представлені кнопки для виклику Stake, Unstake та Withdraw. Код виглядає наступним чином:

```
<div class="actions">
      <el-button type="success"
@click="stakeCall">Stake</el-button>
      <el-button type="info"
@click="unstakeCall">Unstake</el-button>
      <el-button type="warning"
@click="withdrawRewardsCall">Withdraw</el-button>
    </div>
```

Код логіки та виконання транзакцій, що означає взяємодію зі смарт контрактом, а саме виклик певних функцій, обробка результатів, та обробка статусів транзакцій:

```
import { Contract } from '@ethersproject/contracts';
import stakingABI from '@/abis/staking.json';
import { WeiPerEther } from '@ethersproject/constants';

export default {
  computed: {
    chainId() {
      return this.$store.getters.getChainId;
    },
    signer() {
      return this.$store.getters.getSigner;
    },
    account() {
      return this.$store.getters.getAccount;
    },
    stakingContractAddress() {
      return this.$store.getters.getStakingContractAddress;
    },
    successTransactions() {
      return this.$store.getters.getSuccessTransactions;
    },
  },
  created() {
    this.getStakingData();
  },
  methods: {
    async stake() {
```

```
          const stakingContract = new
Contract(this.stakingContractAddress,
            stakingABI,
            this.signer);
          const tx = await
stakingContract.stake(WeiPerEther.mul(10000));
          await this.handleTransaction(tx);
        },
        async withdrawRewards() {
          const stakingContract = new
Contract(this.stakingContractAddress,
            stakingABI,
            this.signer);
          const tx = await stakingContract.withdrawRewards();
          await this.handleTransaction(tx);
        },
        async getStakingData() {
          const stakingContract = new
Contract(this.stakingContractAddress,
            stakingABI,
            this.signer);
          const { balance: staked } = await
stakingContract.getUserData(this.account);
          const rewards = await
stakingContract.getUserRewards(this.account);
          this.$store.commit('setUserStaked',
staked.toString());
          this.$store.commit('setUserRewards',
rewards.toString());
        },
        async handleTransaction(tx) {
          await this.$store.dispatch('setTransaction', tx.hash);
          await tx.wait(1);
          await this.$store.dispatch('completeTransaction',
tx.hash);
          await this.getStakingData();
        },
      },
    };
```

3.3 Опис коду для компоненті Telegram Bot

Програмний код для Telegram Bot виконаний за допомогою сучасної мови програмування JS та за допомогою офіційної бібліотеки для взаємодії з Telegram bot - telegraf.

Програмний код для контролю бота буде складатися з частини, яка відповідає за відповіді від імені бота на запити користувача. На приклад, бот буде завжди відповідати на команду */start* тим, що буде у відповідь відправляти користувачеві кнопку, при натисканні на яку, буде відчинятися вікно в месенджері Telegram для взаємодії з блокчейном.

Код відповіді на команду від користувача виглядає наступним чином:

```
const bot = new Telegraf('Телеграм Бот Токен');
bot.command('start', (ctx) => {
 bot.telegram.sendMessage(ctx.chat.id, "Connect", {
   reply_markup: {
     keyboard: [
       [
         {
           text: "Connect",
           web_app: {url: "Посилання на сервер"}
         }
       ]
     ]
   }
 })
})
bot.launch().then(() => {
});
```

Додатковою можливістю даного методу для взаємодії користувача з блокчейном, є швидка взаємодія у разі необхідності зі своїми даними в мережі блокчейн.

Тобто завдяки сучасним месенджерам та широкому розповсюдженню блокчейн технологій реалізовано всі сформульовані задачі проекту.

4 РЕЗУЛЬТАТИ РОБОТИ ТА ІНСТРУКЦІЯ КОРИСТУВАЧА

Користуватися даною системою можливо за допомоги Telegram додатку як для мобільних пристроїв, так і для комп’ютерного використання. Мається на увазі, що користувач вже вміє користуватися месенджером Telegram.

Під час першого користування потрібно встановити та налаштувати будь-який крипто гаманець з підтримкою протоколу WalletConnect. Для прикладу буде використаний безкоштовний крипто гаманець додаток для мобільного телефону Metamask. Необхідно його встановити з офіційних джерел, та пройти не складу авторизацію для створення або імпорту доступу до власних акаунтів в блокчейні. Подібний спосіб дозволить безпечно отримати доступ для взаємодії з блокчейном.

Після підготовки крипто, потрібно в месенджері знайти спеціально підготовленого бота для даної системи(рис. 4.1).

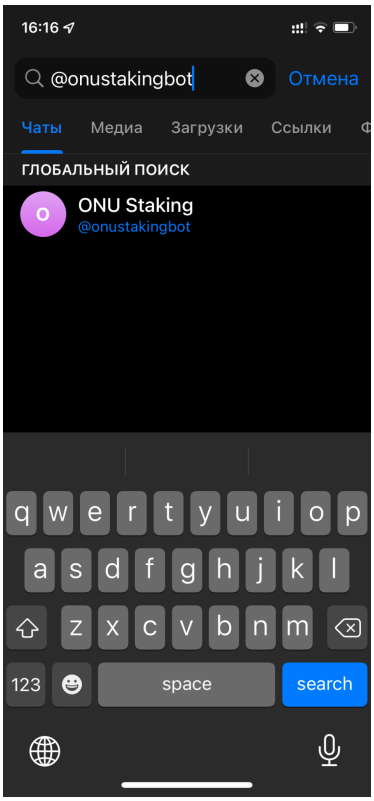

Рисунок 4.1 – Пошук бота в месенджері

Після знаходження бота, потрібно надіслати повідомлення для початку роботи з ним, для цього було обрано команду **/start**, на дане повідомлення бот відразу відповідає на повідомлення текстом та відображає кнопку для взаємодії з блокчейном **Connect**(рис. 4.2).

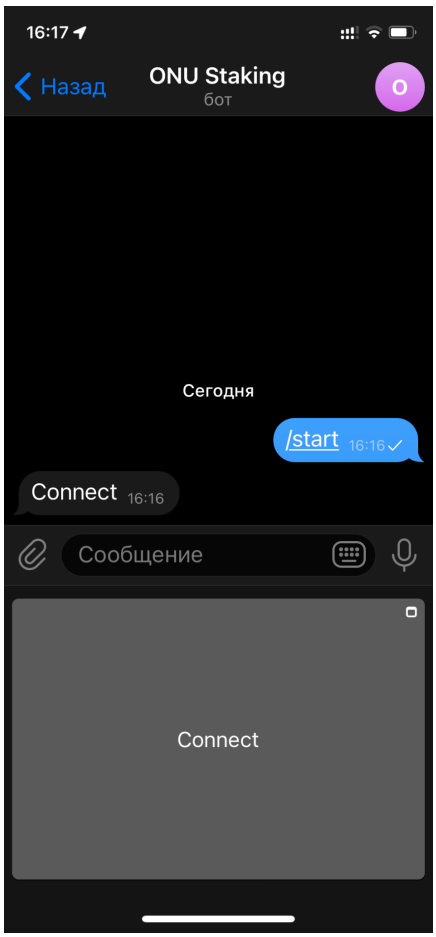

Рисунок 4.2 – Початок взаємодії з блокчейном через месенджер

Після всіх дій вища, користувач може вже почати взаємодію за блокчейном, а саме натиснути на кнопку **Connect** та в додатку одразу відкривається кастомний UI для взаємодії(рис. 4.3).

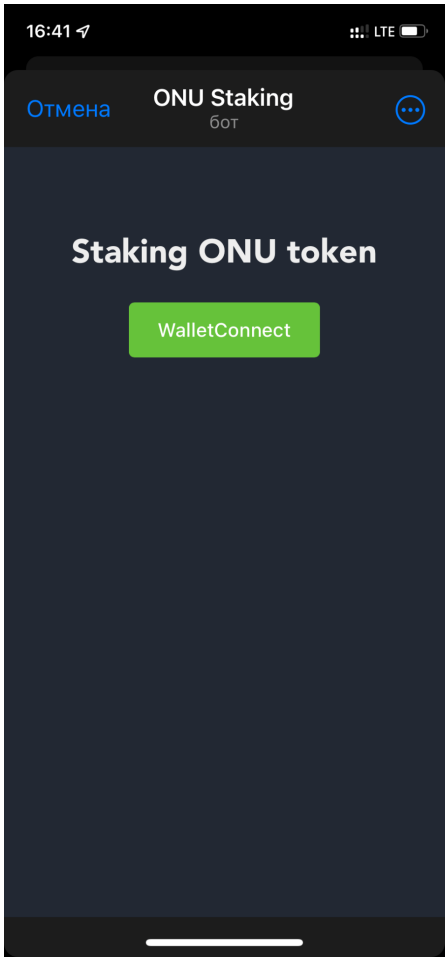

Рисунок 4.3 – Перший екран кастомного UI в месенджері

Тепер для всіх дій в блокчейні, користувачеві потрібно перший раз під’єднати свій крипто-гаманець. Для цього потрібно натиснути на кнопку WalletConnect та вибрати зі списку свій гаманець, на прикладі обрано гаманець MetaMask та автоматично відчиняється додаток на смартфоні з запитом на підтвердження дій заради безпеки(рис. 4.4).

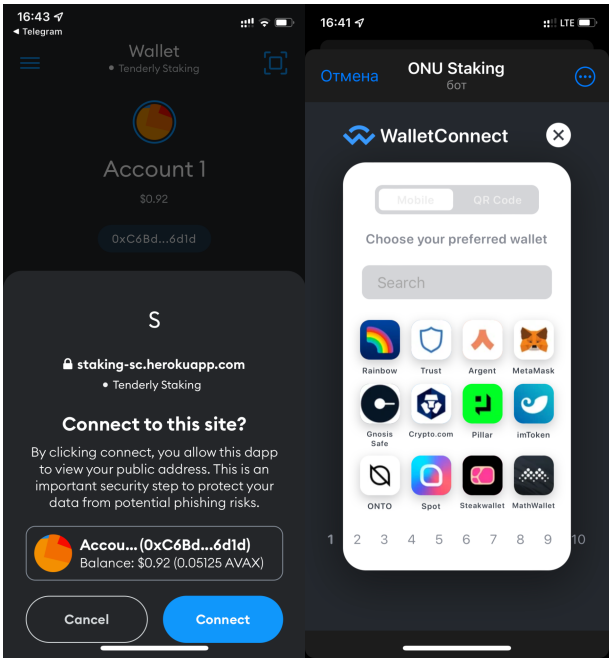

Рисунок 4.4 – Підключення крипто-гаманця до системи

Після успішного виконання вище описаних дій, користувача повертає в месенджер, де вже знаходиться адаптований інтерфейс з інформацією про аккаунт користувача в блокчейні, а також 3 кнопки для взаємодії з ним(рис. 4.5).

Опис елементів інтерфейсу:

1. APR - це річний дохід в відсотках за депозит ERC20 Токена ONU
2. Balance - це поточний рахунок користувача, інакше кількість токенів ONU в блокчейні.
3. Staked - це кількість ERC20 Токенів ONU, які користувач перерахував до SC для подальшого накопичення відсотків.
4. Rewards - це кількість ONU токенів, які користувач заробив.
5. Stake(10k) - кнопка для перерахування 10000 ERC20 Токенів ONU на рахунок SC
6. Unstake - кнопка для зняття усіх своїх коштів зі SC
7. Withdraw - кнопка для зняття усієї накопиченої нагороди

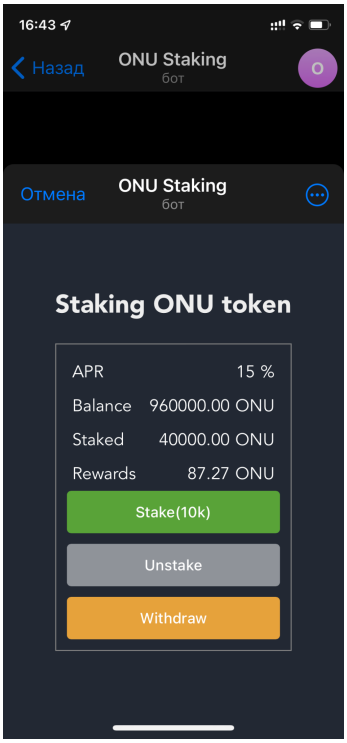

Рисунок 4.5 – Основний інтерфейс системи

Розглянемо функцію Stake, вона потрібна для перерахування 10000 ONU токенів на рахунок SC для подальшого накопичення нагороди. А саме в результаті транзакції 10000 ONU токенів будуть перераховані на рахунок SC з рахунку користувача. Після натискання на кнопку, відчиняється крипто гаманець зі запитом на підтвердження транзакції, після підтвердження, на UI видно що значення в графі Staked зросло на 10000, а Balance зменшилося на 10000 (рис. 4.6).

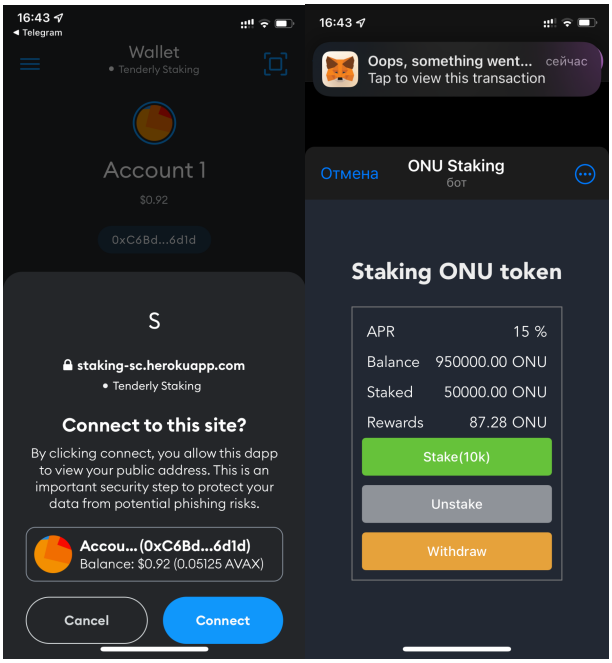

Рисунок 4.6 – Виконання функціоналу Stake

Розглянемо функцію Unstake, вона потрібна для зняття усіх коштів, які користувач раніше перерахував до SC. А саме в результаті транзакції усі кошти користувача на SC повернуться на його крипто гаманець. Після натискання на кнопку, відчиняється крипто гаманець зі запитом на підтвердження транзакції, після підтвердження, на UI видно що значення в графі Staked дорівнює 0, а Balance збільшилося на 50000 (рис. 4.7).

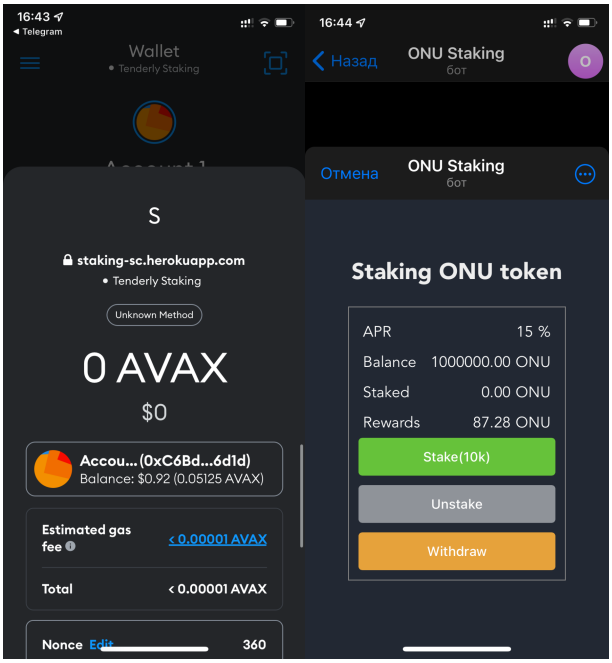

Рисунок 4.7 – Виконання функціоналу Unstake

Розглянемо функцію Withdraw, вона потрібна для зняття накопиченої нагороди. А саме в результаті транзакції кошти, які заробив користувач, за зберігання ERC20 Токенів ONU на SC . Після натискання на кнопку, відчиняється крипто гаманець зі запитом на підтвердження транзакції, після підтвердження, на UI видно що значення в графі Rewards дорівнює 0, а Balance збільшилося на розмір нагороди (рис. 4.8).

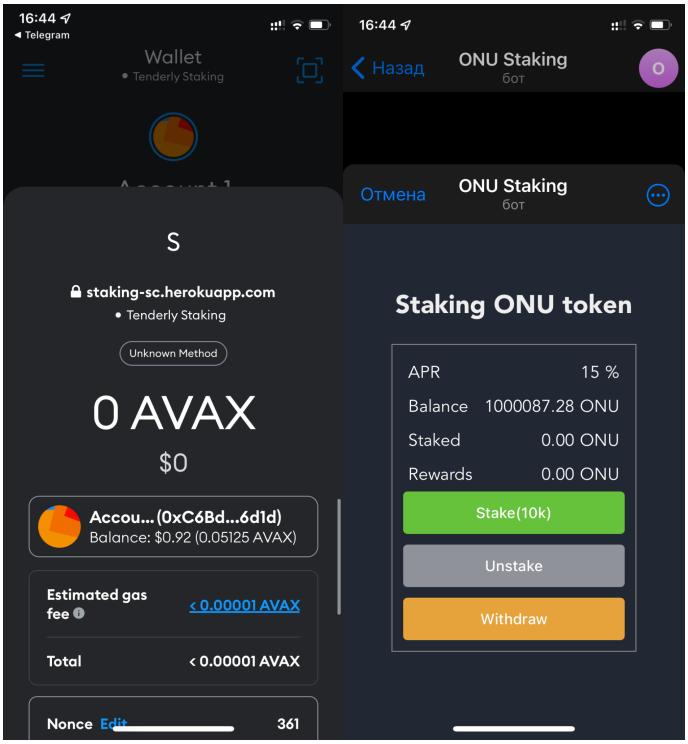

Рисунок 4.8 – Виконання функціоналу Withdraw

ВИСНОВОК

В ході виконання кваліфікаційної роботи було досліджено предметну область, сформульовано перелік вимог до системи. Окремо було досліджено сучасні месенджери, на основі чого було обрано методи для створення прототипу системи. В якості блокчейна для прототипування було обрано блокчейн на основі EVM та підтримкою SC Avalanche C-Chain, у якості сучасного месенджера з можливістю взаємодії з блокчейном було обрано Telegram. Для забезпечення функціонування системи було розроблено керуючий код, з застосуванням середовища програмування WebStorm.

Створений прототип системи дозволяє бачити поточний баланс, кількість перерахованих коштів(інвестицій) та кількість зароблених коштів користувача, інвестувати ERC20 Токени до SC, виводити кошти, які були раніше проінвестовані користувачем, та виводити кошти, які заробив користувач, перерахувавши на SC ERC20 Токени.

Особливістю створення прототипу є те, що він демонструє повний шлях взаємодії користувача з розподіленим реєстром на основі блокчейн, та дозволяє на основі даного методу розробляти взаємодію з різними сучасними проектами на основі блокчейн для приваблення та спрощення взаємодії користувача з блокчейном.

Для більш зручного використання та розширення функціональних можливостей бажано створити програмне забезпечення з зручним інтерфейсом користувача, для забезпечення більшої зручності та привабливості користування системою, а також є можливість для розширення можливостей використовуємого бота для взаємодії з будь-якою кількістю подібних рішень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ethereum документація для розробників: [Електронний ресурс]. – Режим доступу: https://ethereum.org/en/developers/docs/.
2. Telegram Bot API документація для взаємодії з telegram: [Електронний ресурс]. – Режим доступу: https://core.telegram.org/bots/api/.
3. Ethers документація для взаємодії з блокчейном: [Електронний ресурс]. – Режим доступу: https://docs.ethers.io/v5/.
4. VueJs документація для роботи з реактивним UI: [Електронний ресурс]. – Режим доступу: https://vuejs.org/
5. Solidity документація для написання SC: [Електронний ресурс]. – Режим доступу: https://docs.soliditylang.org/en/v0.8.14/

ДОДАТОК А

Програмний код UI частини

Основний файл додатка для відображення компонент та обробки даних:

```
<template>
       <div id="app">
         <h1>Staking <b>ONU</b> token</h1>
         <div v-if="!connectStatus">
           <ConnectButton/>
         </div>

         <UserInfo v-else/>
       </div>
     </template>

     <script>
     import UserInfo from '@/components/UserInfo.vue';
     import ConnectButton from '@/components/ConnectButton.vue';
     import { mapGetters } from 'vuex';
     import 'element-theme-dark';

     export default {
       name: 'App',
       data() {
         return {
           stakingContract: undefined,
         };
       },
       components: {
         UserInfo,
         ConnectButton,
       },
       computed: {
         ...mapGetters([
           'connectStatus',
         ]),
       },
     };
     </script>

     <style>
     * {
       margin: 0;
       padding: 0;
```

```
    box-sizing: border-box;
  }
  #app {
    font-family: Avenir, Helvetica, Arial, sans-serif;
    display: flex;
    flex-direction: column;
    justify-content: flex-start;
    align-items: center;
    margin-top: 40px;
  }
  h1 {
    margin: 20px;
    font-size: 1.5em;
  }
  </style>
```

ConnectButton компонент:

```
<template>
    <el-button type="success"
@click="connectWallet">WalletConnect</el-button>
  </template>

  <script>
  import { mapActions, mapGetters } from 'vuex';

  export default {
    name: 'ConnectButton',
    computed: mapGetters([
      'connectStatus',
      'getProvider',
    ]),
    methods: mapActions([
      'connectWallet',
      'connectProvider',
    ]),
  };
  </script>

  <!-- Add "scoped" attribute to limit CSS to this component
only -->
  <style scoped>
  h3 {
    margin: 40px 0 0;
  }
```

```
      ul {
        list-style-type: none;
        padding: 0;
      }
      li {
        display: inline-block;
        margin: 0 10px;
      }
      a {
        color: #42b983;
      }
      </style>
      <template>
        <div class="wrapper">
          <div class="info">
            <div class="item">
              <span class="title">APR</span>
              <div class="valueBlock">
                <span class="value">15</span>
                <span class="symbol">%</span>
              </div>
            </div>
            <div class="item">
              <span class="title">Balance</span>
              <div class="valueBlock">
                <span class="value">{{balance}}</span>
                <span class="symbol">{{ getTokenInfo.symbol
}}</span>
              </div>
            </div>
            <div class="item">
              <span class="title">Staked</span>
              <div class="valueBlock">
                <span class="value">{{ staked }}</span>
                <span class="symbol">ONU</span>
              </div>
            </div>
            <div class="item">
              <span class="title">Rewards</span>
              <div class="valueBlock">
                <span class="value">{{ rewards }}</span>
                <span class="symbol">ONU</span>
              </div>
            </div>
          </div>
```

```
        <div class="actions">
          <el-button type="success"
@click="stakeCall">Stake(10k)</el-button>
          <el-button type="info"
@click="unstakeCall">Unstake</el-button>
          <el-button type="warning"
@click="withdrawRewardsCall">Withdraw</el-button>
        </div>
      </div>
    </template>

    <script>
```

UserInfo компонент:

```
    import staking from '@/mixins/staking';
    import token from '@/mixins/token';
    import { formatUnits } from '@ethersproject/units';
    import { mapGetters } from 'vuex';

    const amountMinForDisplay = 0.01;
    const fixPoint = 2;
    const formatAmounts = (value, decimals) => {
      const rawValue = parseFloat(formatUnits(value, decimals));
      if (rawValue < amountMinForDisplay && rawValue !== 0) {
        return `<${amountMinForDisplay}`;
      }
      return rawValue.toFixed(fixPoint);
    };
    export default {
      name: 'UserInfo',
      mixins: [staking, token],
      data() {
        return {
          balance: '0',
          staked: '0',
          rewards: '0',
        };
      },
      computed: mapGetters([
        'getUserUpdated',
        'getUserInfo',
        'getTokenInfo',
      ]),
      watch: {
        getUserUpdated(newValue) {
          if (newValue) {
```

```
            this.handleUserData();
          }
        },
      },
      methods: {
        async unstakeCall() {
          await this.unstake();
        },
        async stakeCall() {
          // await this.approve();
          await this.stake();
        },
        async withdrawRewardsCall() {
          await this.withdrawRewards();
        },
        async handleUserData() {
          this.balance = formatAmounts(this.getUserInfo.balance,
this.getTokenInfo.decimals);
          this.staked = formatAmounts(this.getUserInfo.staked,
this.getTokenInfo.decimals);
          this.rewards = formatAmounts(this.getUserInfo.rewards,
this.getTokenInfo.decimals);
        },
      },
    };
    </script>

    <!-- Add "scoped" attribute to limit CSS to this component
only -->
    <style scoped>
    .wrapper {
      max-width: 450px;
      display: flex;
      justify-content: space-between;
      border: 1px solid grey;
      padding: 10px;
    }

    .info {
      max-width: 400px;
      min-width: 200px;
      display: flex;
      flex-direction: column;
      justify-content: center;
      padding-right: 20px;
```

```
}

.info > .item {
  display: flex;
  color: white;
  justify-content: space-between;
  padding: 5px;
}

.info > .item .title {
  margin-right: 15px;
}

.info > .item .valueBlock {
  display: flex;
  justify-content: flex-start;
}

.info > .item .value {
  margin-right: 5px;
}

.actions {
  max-width: 150px;
  display: flex;
  flex-direction: column;
}

.actions .el-button + .el-button {
  margin-left: 0;
}

.actions .el-button {
  margin-bottom: 10px;
}
.actions .el-button:last-child {
  margin-bottom: 0;
}

@media screen and (max-width: 500px) {
  .wrapper {
    flex-direction: column;
    align-items: center;
  }
  .info {
```

```
      padding: 0;
    }
    .actions {
      max-width: none;
      width: 100%;
    }
  }
  </style>
```

Бізнес логіка для роботи зі смарт контрактом стєйкінга:

```
  import { Contract } from '@ethersproject/contracts';
  import stakingABI from '@/abis/staking.json';
  import { WeiPerEther } from '@ethersproject/constants';

  export default {
    computed: {
      chainId() {
        return this.$store.getters.getChainId;
      },
      signer() {
        return this.$store.getters.getSigner;
      },
      account() {
        return this.$store.getters.getAccount;
      },
      userStaked() {
        return this.$store.getters.getUserInfo.staked;
      },
      stakingContractAddress() {
        return this.$store.getters.getStakingContractAddress;
      },
      successTransactions() {
        return this.$store.getters.getSuccessTransactions;
      },
    },
    created() {
      this.getStakingData();
    },
    methods: {
      async stake() {
        const stakingContract = new
Contract(this.stakingContractAddress,
          stakingABI,
          this.signer);
```

```
          const tx = await
stakingContract.stake(WeiPerEther.mul(10000));
          await this.handleTransaction(tx);
        },
        async unstake() {
          const stakingContract = new
Contract(this.stakingContractAddress,
            stakingABI,
            this.signer);
          const tx = await
stakingContract.unstake(this.userStaked);
          await this.handleTransaction(tx);
        },
        async withdrawRewards() {
          const stakingContract = new
Contract(this.stakingContractAddress,
            stakingABI,
            this.signer);
          const tx = await stakingContract.withdrawRewards();
          console.log(tx);
          await this.handleTransaction(tx);
        },
        async getStakingData() {
          const stakingContract = new
Contract(this.stakingContractAddress,
            stakingABI,
            this.signer);
          const { balance: staked } = await
stakingContract.getUserData(this.account);
          const rewards = await
stakingContract.getUserRewards(this.account);
          this.$store.commit('setUserStaked',
staked.toString());
          this.$store.commit('setUserRewards',
rewards.toString());
        },
        async handleTransaction(tx) {
          await this.$store.dispatch('setTransaction', tx.hash);
          await tx.wait(1);
          await this.$store.dispatch('completeTransaction',
tx.hash);
          await this.getStakingData();
        },
      },
    };
```

Бізнес логіка для роботи зі смарт контрактом ERC20 Токена:

```
import { Contract } from '@ethersproject/contracts';
import { MaxUint256 } from '@ethersproject/constants';
import erc20 from '@/abis/erc20.json';

export default {
  computed: {
    chainId() {
      return this.$store.getters.getChainId;
    },
    signer() {
      return this.$store.getters.getSigner;
    },
    account() {
      return this.$store.getters.getAccount;
    },
    stakeTokenAddress() {
      return this.$store.getters.getStakeTokenAddress;
    },
    stakingContractAddress() {
      return this.$store.getters.getStakingContractAddress;
    },
  },
  created() {
    this.getTokenData();
  },
  methods: {
    async approve() {
      const tokenContract = new
Contract(this.stakeTokenAddress,
        erc20,
        this.signer);
      const tx = await
tokenContract.approve(this.stakingContractAddress, MaxUint256);
      await tx.wait();
    },
    async getTokenData() {
      const tokenContract = new
Contract(this.stakeTokenAddress,
        erc20,
        this.signer);
```

```
          const allowance = await
tokenContract.allowance(this.account,
this.stakingContractAddress);
          const balance = await
tokenContract.balanceOf(this.account);
          const name = await tokenContract.name();
          const symbol = await tokenContract.symbol();
          const decimals = await tokenContract.decimals();
          this.$store.commit('setUserInfo', { allowance:
allowance.toString(), balance: balance.toString() });
          this.$store.commit('setTokenInfo', { name, symbol,
decimals: decimals.toString() });
        },
      },
    };
```

Логіка для підключення крипто гаманця до додатку:

```
    import WalletConnectProvider from '@walletconnect/web3-
provider';
    import { Web3Provider } from '@ethersproject/providers';

    // initial state
    const stateData = () => ({
      mode: process.env.NODE_ENV || 'development',
      connected: false,
      chainId: 0,
      provider: null,
      signer: null,
      account: null,
    });

    // getters
    const getters = {
      connectStatus: (state) => state.connected,
      getProvider: (state) => state.provider,
      getSigner: (state) => state.signer,
      getAccount: (state) => state.account,
    };

    // actions
    const actions = {
      async connectWallet({
        commit,
      }) {
```

```
          const walletConnectProvider = new
WalletConnectProvider({
            rpc: {
              43114: 'https://api.avax.network/ext/bc/C/rpc',
            },
          });

          const { connector } = walletConnectProvider;
          const accounts = await walletConnectProvider.enable();
          const provider = new
Web3Provider(walletConnectProvider);
          const signer = provider.getSigner();
          commit('setChainId', connector.chainId);
          commit('setSigner', Object.freeze(signer));
          commit('setAccount', accounts[0]);
          commit('setProvider',
Object.freeze(walletConnectProvider));
          commit('setConnect', true);
        },
      };
      // mutations
      const mutations = {
        setConnect(state, status) {
          state.connected = status;
        },
        setProvider(state, payload) {
          state.provider = payload;
        },
        setChainId(state, payload) {
          state.chainId = payload;
        },
        setSigner(state, payload) {
          state.signer = payload;
        },
        setAccount(state, payload) {
          state.account = payload;
        },
      };
      export default {
        state: stateData,
        getters,
        actions,
        mutations,
      };
```