Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Розробка децентралізованої P2P платформи обміну криптовалют з інтеграцією механізмів виявлення аномалій та системи попереджень про підозрілі дії в реальному часі на основі WebSocket

Виконав: студент 4 курсу, гр.1КІТС-21б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Вітенко А.Ю.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. МБІС
Грицак А.В.
(прізвище та ініціали)
«____» _____ 2025 р.

Рецензент: к.т.н., доцент каф. ОТ
Томчук М.А.
(прізвище та ініціали)
«____» _____ 2025 р.

Допущено до захисту
Голова секції УБ кафедри МБІС
д.т.н., проф. Яремчук Ю.Є.
“____” _____2025р.

Вінниця ВНТУ – 2025

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти І-й (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС
д.т.н., проф. Яремчук Ю.Є.
" 20 " березня 2025 р.

З А В Д А Н Н Я
на бакалаврську кваліфікаційну роботу студенту
Вітенку Артему Юрійовичу
(прізвище, ім`я, по-батькові)

1. Тема роботи Розробка децентралізованої P2P платформи обміну криптовалют з інтеграцією механізмів виявлення аномалій та системи попереджень про підозрілі дії в реальному часі на основі WebSocket
Керівник роботи к.т.н., доцент каф. МБІС Грицак А.В.
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від " 20 " березня 2025року № 97
2. Термін подання студентом роботи 02.06.2025
3. Вихідні дані до роботи: нормативно-правова база, монографії та сучасні наукові статті по темі, Інтернет-ресурси, стандарти, існуюче програмне забезпечення.
4. Зміст текстової частини: Розглянуто основні концепції децентралізованих фінансових систем, принципи роботи P2P-обміну, типові загрози безпеці та способи їх мінімізації. Проведено аналіз існуючих аналогів і визначено їх недоліки в безпеці. Описано архітектуру запропонованої системи, зокрема структуру смарт-контракту, реалізацію механізму моніторингу та клієнтський інтерфейс. Результати розробки підтверджено тестуванням функціональних модулів, що демонструє перспективність запропонованого рішення.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень): у першому розділі наведено 1 рисунок та 1 таблиця; у другому розділі наведено 2 рисунки; у третьому розділі наведено 13 рисунків;у додатку Б наведено лістинг програми; у додатку В наведено ілюстративний матеріал (презентація); у додатку Г наведено протокол перевірки на антиплагіат.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	к.т.н., доцент каф. МБІС__Грицак А.В.		
Розділ II	к.т.н., доцент каф. МБІС__Грицак А.В.		
Розділ III	к.т.н., доцент каф. МБІС__Грицак А.В.		

7.Дата видачі завдання 20 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1.	Визначення напрямку бакалаврської роботи, формулювання теми	20.03.2025	27.03.2025	
2.	Аналіз предметної області обраної теми	28.03.2025	09.04.2025	
3.	Розробка алгоритму роботи	10.04.2025	31.04.2025	
4.	Написання бакалаврської роботи на основі розробленої теми	01.05.2025	16.05.2025	
5.	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	31.05.2025	
6.	Виправлення, уточнення, корегування бакалаврської дипломної роботи	01.06.2025	05.06.2025	
7.	Захист бакалаврської кваліфікаційної роботи	09.06.2025	10.06.2025	

Студент _____ Віменко А.Ю.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Грицак А.В.

АНОТАЦІЯ

Дана робота присвячена розробці децентралізованої P2P-платформи обміну криптовалют із вбудованими механізмами виявлення аномалій та системою попереджень про підозрілі дії в режимі реального часу. Метою дослідження є створення безпечного, масштабованого та надійного середовища для прямого обміну цифровими активами між користувачами без участі централізованих посередників.

У межах роботи проведено аналіз сучасних підходів до реалізації P2P-обміну, використання смарт-контрактів у мережі Ethereum, а також застосування WebSocket-технологій для передачі подій у реальному часі. Запропонована система включає функціонал авторизації користувача, створення та підтвердження ордерів, фіксацію транзакцій через смарт-контракт, а також реалізацію серверної логіки виявлення аномальної активності з подальшим сповіщенням.

Основною новизною запропонованого рішення є інтеграція механізмів моніторингу та активного реагування на підозрілі дії в архітектуру децентралізованої системи обміну, що дозволяє не лише уникати централізованого контролю, а й забезпечити високий рівень безпеки та прозорості.

Ключові слова: P2P-платформа, смарт-контракт, блокчейн, WebSocket, обмін криптовалют, аномалія, безпека, децентралізація, система попереджень.

ABSTRACT

This work is dedicated to the development of a decentralized P2P cryptocurrency exchange platform with integrated anomaly detection mechanisms and a real-time suspicious activity alert system. The aim of the research is to create a secure, scalable, and reliable environment for direct exchange of digital assets between users without the involvement of centralized intermediaries.

The study analyzes current approaches to implementing P2P exchanges, the use of smart contracts within the Ethereum network, and the application of WebSocket technology for real-time event transmission. The proposed system includes user authorization functionality, order creation and confirmation, transaction processing via smart contracts, and the implementation of server-side logic for detecting anomalous behavior followed by appropriate notifications.

The main novelty of the proposed solution lies in the integration of monitoring mechanisms and active response to suspicious actions within the architecture of a decentralized exchange system, enabling the platform to not only eliminate centralized control but also ensure a high level of security and transparency.

Keywords: P2P platform, smart contract, blockchain, WebSocket, cryptocurrency exchange, anomaly, security, decentralization, alert system.

ЗМІСТ

ВСТУП

Актуальність задачі. Із зростанням популярності криптовалют та децентралізованих фінансових систем - DeFi, все більше користувачів надають перевагу прямому обміну цифровими активами без посередників. Peer-to-peer платформи є основним інструментом для реалізації таких транзакцій. Однак з ростом обсягів операцій зростає й кількість шахрайських дій, фішингу, маніпуляцій з боку користувачів що робить безпеку - критичним аспектом таких систем. Відсутність централізованого контролю створює виклик для виявлення аномальної активності та миттєвого реагування на потенційні загрози. Інтеграція механізмів виявлення аномалій в реальному часі та системи сповіщень на основі WebSocket-з'єднання дозволяє не лише оперативно фіксувати підозрілі дії, а й попереджати користувачів або адміністрацію про можливі порушення. Саме тому розробка децентралізованої P2P-платформи з вбудованими механізмами моніторингу й реагування є актуальною задачею, що поєднує безпеку, прозорість і автономність.

Мета роботи. Метою даної дипломної роботи є проєктування та розробка децентралізованої платформи для P2P-обміну криптовалютами, яка забезпечує повноцінний функціонал прямої взаємодії між користувачами без участі централізованих посередників. Основним завданням є реалізація безпечного обміну цифровими активами шляхом використання смарт-контракту, який гарантує виконання умов угоди відповідно до заздалегідь визначених параметрів та автоматично фіксувати підозрілі дії на основі заданих поведінкових патернів і системних правил.

Об’єкт дослідження. Об’єктом дослідження є децентралізовані програмні платформи обміну криптовалют, що функціонують у форматі P2P без централізованого управління та контролю.

Предмет дослідження. Предметом дослідження є програмна реалізація механізмів обміну, авторизації, виявлення аномальної активності та надсилання попереджень у межах P2P-платформи.

Новизна роботи. Основна новизна роботи полягає в інтеграції декількох важливих технічних рішень у межах єдиної децентралізованої платформи для P2P обміну криптовалютами. Зокрема, реалізовано поєднання механізму прямого обміну активами між користувачами на основі смарт-контракту з анонімною авторизацією, що дозволяє забезпечити конфіденційність та безпеку доступу без залучення централізованих систем зберігання облікових даних. Крім того, в систему інтегровано підтримку передачі та обробки подій у режимі реального часу за допомогою WebSocket-з’єднання, що забезпечує миттєву взаємодію між клієнтом і сервером, підвищуючи швидкість реакції системи на зміни у стані користувацьких дій.

Практична цінність. Запропонована система може бути впроваджена в сучасні DeFi-проєкти, P2P-обмінники та криптовалютні сервіси, де важливо забезпечити не лише ефективний обмін, а й безпеку транзакцій і довіру користувачів. Реалізовані механізми моніторингу дій та надсилання попереджень дозволяють вчасно виявляти порушення, мінімізувати ризики та захищати усі сторони обміну від зловживань.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОЇ РОБОТИ

В даному розділі буде здійснено аналіз сучасних децентралізованих платформ для обміну токенами, зокрема їх архітектурних моделей, функціональних особливостей та механізмів забезпечення ліквідності. Розглянуто роль блокчейну та смарт-контрактів у роботі таких платформ, а також типові загрози та вразливості, що впливають на безпеку користувачів. Особлива увага приділена класифікації існуючих рішень і виявленню їхніх недоліків, що обґрунтовує необхідність розробки нової децентралізованої платформи з підвищеним рівнем безпеки та ефективності. У підсумку сформульовано основні задачі дослідження та розробки, що будуть реалізовані у наступних розділах.

1.1 Аналіз принципів роботи блокчейну та смарт-контрактів

Сучасний розвиток інформаційних технологій значною мірою вплинув на зміну підходів до зберігання, обробки та передачі даних. Однією з найбільш інноваційних концепцій останніх десятиліть стала технологія блокчейн, яка здатна радикально трансформувати підходи до побудови децентралізованих систем управління та забезпечення безпеки даних.

Блокчейн – це розподілена база даних або реєстр, що складається з послідовно з’єднаних блоків даних, кожен з яких містить інформацію про транзакції, хронологічно записані та захищені криптографією, яка відіграє ключову роль в забезпеченні прозорості та безпеки блокчейну [1].

Одним із ключових аспектів криптографії в блокчейні є хешування, яке забезпечує захист і цілісність даних. Кожен блок у блокчейні містить унікальний хеш, створений за допомогою криптографічної хеш-функції, наприклад, Keccak-256, який використовується у мережі Ethereum. Хеш блоку генерується на основі його вмісту та хеша попереднього блоку, що створює міцний зв’язок між блоками та гарантує незмінність даних ланцюга. Якщо будь-які дані в блоці

зазнають змін, це автоматично спричиняє зміну його хеша, що порушує зв’язок з іншими блоками та сигналізує про несанкціоноване втручання. Такий механізм забезпечує високий рівень безпеки та довіри до інформації, що зберігається в блокчейні [2].

Для перевірки достовірності хеша в блокчейн-мережах використовуються два основних методи консенсусу: Proof of Work (PoW) та Proof of Stake (PoS). Ці механізми забезпечують узгодженість даних між усіма учасниками мережі та гарантують, що нові блоки додаються до блокчейну лише після перевірки їх достовірності. Хоча PoW був першим широковживаним методом, PoS є новим та прогресивним рішенням, яке відповідає сучасним вимогам ефективності, безпеки та екологічності блокчейн-технологій.

PoS вирішує ключові недоліки PoW, такі як енергомісткість і низька масштабованість. Він замінює майнінг процесом стейкінгу, в якому учасники блокчейн-мережі, які мають назву валідатори, блокують свої токени для отримання права створювати нові блоки та перевіряти транзакції. Основні переваги PoS:

1. Валідатори в PoS блокують власні кошти, щоб отримати право створювати блоки. В разі, спроби обманути систему наприклад, додаючи конфліктуючі блоки або підтверджуючи недійсні транзакції, стейк буде частково або повністю конфісковано. Даний механізм називається slashing. Принцип створює економічну мотивацію користувачам, для дотримання правил [3].

2. Щоб виконати атаку 51% зловмисник повинен отримати контроль над більш ніж 51% стейкованих токенів у мережі. Це означає, що зловмиснику необхідно придбати або іншим чином отримати значну кількість токенів, які використовуються в мережі для забезпечення консенсусу. Така вимога значно ускладнює реалізацію атаки, оскільки володіння більшістю токенів пов’язане з величезними фінансовими витратами [4].

3. Валідатор у мережі обирається за допомогою криптографічно алгоритму випадкової вибірки. Одним із таких алгоритмів є Verifiable Random Function, який поєднує унікальність криптографічного підпису з гарантіями

випадковості. VRF генерує псевдовипадкове число, яке перевіряється іншими учасниками мережі, забезпечуючи прозорість процесу вибору валідатора, що унеможливлює передбачення, хто стане наступним валідатором, знижуючи ризики спроб зовнішнього впливу на процес [4].

4. В розподілених мережах, блоки та транзакції, підписуються валідаторами з використанням цифрових підписів, які відіграють ключову роль у забезпеченні автентичності даних та довіри до їхнього походження. Цифровий підпис підтверджує, що відповідна транзакція або блок були сформовані саме тим валідатором, який має відповідні повноваження в мережі. Підпис створюється за допомогою приватного ключа, що є унікальним для кожного валідатора і зберігається в таємниці. Після формування транзакції або блоку підпис додається до структури даних і розповсюджується разом із ними в мережі. Інші вузли мають можливість перевірити цей підпис за допомогою публічного ключа валідатора, переконавшись у достовірності даних і тому, що вони не були змінені після підписання. Такий механізм гарантує цілісність та підтверджувальне авторство кожної транзакції або блоку в системі [5].

5. Використання модифікованої версії протоколів Byzantine Fault Tolerance, які забезпечують узгодженість даних навіть у разі, якщо частина учасників мережі діє недобросовісно або стає недоступною. BFT-протоколи побудовані на принципі досягнення консенсусу між вузлами мережі, незважаючи на наявність збоїв або зловмисних дій. У PoS для затвердження нового блоку потрібна згода як мінімум двох третин валідаторів, що гарантує стійкість мережі навіть за умов часткової недовіри. Дана архітектура також забезпечує високу стійкість до зовнішніх атак. Завдяки децентралізованій природі PoS-мережі вони можуть продовжувати функціонувати навіть у випадках, коли частина вузлів зазнала атак або стала недоступною через регіональні чи технічні проблеми. Розподіл валідаторів по всьому світу гарантує, що відмова або компрометація окремих вузлів не вплине на загальну працездатність мережі [5].

В рамках розробки децентралізованих платформ, таких як системи обміну токенами, використання блокчейну є ключовим елементом. Він забезпечує

основну інфраструктуру для взаємодії між користувачами без посередників, що забезпечує стійкість до зовнішніх маніпуляцій та атак. На відміну від централізованих систем, де ключові рішення приймаються єдиним органом, блокчейн покладається на консенсус між мережею незалежних учасників. Це виключає можливість монополізації процесів та зловживань з боку окремих осіб чи організацій. Дії системи контролюються розподіленою мережею, що робить її надзвичайно надійною та стійкою до зовнішніх загроз. Технологія смарт-контрактів, яка є складовою блокчейну, дає можливість автоматизувати виконання угод, зменшуючи ризики людської помилки та забезпечуючи високу ступінь довіри між сторонами

Смарт-контракт – це програма, яка виконується у децентралізованому середовищі, до прикладу Ethereum Virtual Machine, скорочено – EVM, яке забезпечує виконання їхнього коду на кожному вузлі мережі, гарантуючи однорідність результатів незалежно від апаратного забезпечення або операційної системи учасників. Це середовище виступає як децентралізований комп'ютер, який виконує всі операції контрактів, підтримуючи їхню надійність та безпеку [6].

Написання смарт-контрактів здійснюється мовою програмування Solidity, яка спеціально розроблена для роботи з блокчейном Ethereum де вони спрацьовують автоматично, коли виконуються умови, закладені в їхній код. Після розгортання у блокчейні код смарт-контракту стає незмінним, що гарантує його передбачуваність і надійність. Ця незмінність забезпечує прозорість, адже кожен учасник мережі може перевірити код перед взаємодією з контрактом, але водночас накладає обмеження: будь-які помилки чи недоліки у коді залишаються в ньому назавжди [7].

Однією з ключових функцій смарт-контрактів є управління токенами, що стало можливим завдяки створенню стандартів, таких як ERC-20 для взаємозамінних токенів, ERC-721 для NFT які є не взаємозамінними токенами та ERC-1155 для багатофункціональних токенів. Ці стандарти визначають правила поведінки токенів, що забезпечує їхню сумісність з різними платформами та застосунками. Для виконання кожної операції смарт-контрактів у мережі

необхідна оплата комісії у вигляді "газу", який вимірює обчислювальну складність операцій і оплачується у криптовалюті Ethereum. Цей механізм не тільки стимулює валідаторів до підтвердження транзакцій, але й захищає мережу від перевантаження [8].

Важливим аспектом роботи смарт-контрактів є їхня інтерактивність. Вони можуть викликати функції інших контрактів, отримувати дані з зовнішніх джерел через оракули або взаємодіяти з користувачами у реальному часі. Крім того, смарт-контракти здатні генерувати події, які записуються у блокчейні та можуть бути використані для створення децентралізованих застосунків. Наприклад, контракт може повідомляти про успішну передачу токенів або зміну стану.

Таким чином, смарт-контракти Ethereum є не лише автономними програмами, а й фундаментом для створення складних децентралізованих екосистем. Маючи унікальні характеристики, які створюють низку вагомих переваг для користувачів і системи в цілому, забезпечуючи надійність, прозорість, автоматизацію та стійкість до маніпуляцій, що робить їх важливим інструментом у децентралізованих екосистемах.

Проте, особливу увагу варто приділити аспектам безпеки, які є критично важливими для роботи децентралізованих платформ. Адже, незважаючи на високу захищеність блокчейну завдяки криптографічним алгоритмам, на практиці існують численні виклики, пов’язані з вразливостями смарт-контрактів, людським фактором і зовнішніми атаками, наприклад атака Self-Destruct. Вона базується на неправильному використанні або маніпуляції функцією selfdestruct у смарт-контракта, та дозволяє видалити контракт із блокчейну, повернувши залишкові кошти на вказану адресу. Якщо вона не захищена належним чином, то може бути використана для знищення контракту, що порушує його функціональність, внесення небажаних коштів на адресу іншого контракту або створення ситуацій, які призводять до несподіваних помилок у виконанні інших контрактів.

Отже, розгляд основних й глибоке розуміння принципів роботи та безпеки блокчейну, смарт-контрактів й криптовалют є необхідним етапом для побудови якісних та ефективних рішень для мінімізації ризиків.

1.2 Аналіз підходів реалізації обміну токенів

Децентралізовані платформи обміну токенами, скорочено – DEX являють собою інструменти, що функціонують на основі блокчейну, використовуючи смарт-контракти і забезпечують можливість здійснення однорангової торгівлі цифровими активами безпосередньо між користувачами, без посередників.

Однією з головних переваг децентралізованих платформ є їхня прозорість, що забезпечується технологією блокчейну. DEX дозволяють користувачам зберігати контроль над своїми приватними ключами та активами протягом усього процесу торгівлі.

Усі транзакції, які виконуються на платформі, записуються у відкритий і незмінний реєстр. Це дозволяє будь-якому користувачеві перевірити достовірність операцій, балансів токенів і загальну історію взаємодії зі смарт-контрактами [9].

Ключовою технологій DEX є автоматизовані маркет-мейкери, які використовують алгоритми для забезпечення ліквідності. Вони базуються на математичних формулах, таких як формула постійного добутку що забезпечує баланс між двома активами в пулі ліквідності. AMM дозволяють уникати необхідності в традиційних ордербуках, що знижує витрати та підвищує ефективність для користувачів [10].

Для детального розуміння екосистеми AMM-бірж варто розглянути ключових учасників, активи та модулі, які складають основу роботи таких платформ. Наступна схема ілюструє структуру Actors, Assets, and Pivotal Modules of AMM-based DEX, демонструючи, як взаємодіють трейдери, провайдери ліквідності, активи та технічні компоненти [11].

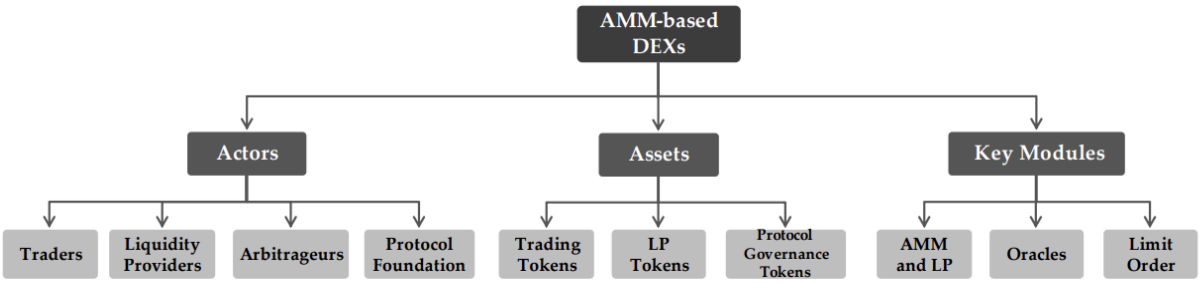


Рисунок 1.1 - Актори, активи та ключові модулі DEX на основі AMM [11].

Дана схема ілюструє, компоненти DEX та їх взаємодію між собою, забезпечуючи ефективність і функціональність екосистеми. Основними елементами є актори, активи і ключові модулі, які складають фундамент роботи AMM [11].

Актори включають чотири основні групи учасників:

1. Трейдери здійснюють операції з купівлі та продажу токенів взаємодіючи із пулами ліквідності для обміну активів.

2. Провайдери ліквідності – додають свої активи до пулів ліквідності, забезпечуючи доступність токенів для трейдерів. За це вони отримують винагороду у вигляді частки комісій за транзакції.

3. Арбітражери – спеціалізуються на коректуванні цін між DEX ринками.

4. Фундація протоколу – включає організації чи групи, які займаються розробкою, підтримкою та вдосконаленням протоколу DEX.

Активи представлені трьома основними типами:

1. Токени для торгівлі – основні активи, що обмінюються у пулах ліквідності.

2. Токени ліквідності – видаються провайдерам ліквідності як підтвердження їхньої частки в пулі. Та можуть використовуватися для отримання комісій або у взаємодії з іншими DeFi-протоколами.

3. Говернанс токени – забезпечують власників правом голосу в управлінні платформою, в саме дозволяють користувачам приймати рішення щодо змін у протоколі, таких як оновлення функціоналу чи зміна розмірів комісій.

Ключові модулі включають основні технічні компоненти, які забезпечують роботу AMM. Модулі поділяються на три типи:

1. Механізм автоматизованого маркет-мейкінгу – забезпечує вибір алгоритму ціноутворення та управління ліквідністю в пулах.

2. Оракули – відповідають за доступ до зовнішніх даних, таких як ціни активів на інших платформах, що допомагає підтримувати точність ціноутворення в AMM [12].

3. Лімітні ордери – дозволяють виставляти замовлення на купівлю або продаж активів за конкретними цінами [13].

Дана структура забезпечує стабільну та ефективну роботу AMM-базованих DEX та створює екосистему, що підтримує торгівлю без посередників, автоматизує процеси управління ліквідністю та забезпечує прозорість і доступність для користувачів.

Однак такий підхід має свої обмеження, зокрема тимчасові втрати для провайдерів ліквідності, які виникають через волатильність ринку. Також, використання публічних блокчейнів, таких як Ethereum, створює певні виклики для конфіденційності. Усі транзакції та дані, записані в блокчейн, доступні для аналізу. Хоча вони не містять персональної інформації оскільки підписуються адресами гаманців, спеціальні аналітичні інструменти можуть зіставляти дані та розкривати певну інформацію про користувачів. Це особливо важливо враховувати в контексті фінансових операцій чи угод, які потребують високого рівня приватності. Для вирішення цих викликів використовуються додаткові методи, такі як застосування протоколів конфіденційності, наприклад zk-SNARK або використання приватних блокчейнів.

Альтернативним підходом для обміну токенами, є централізовані платформи, або CEX. Вони вважаються традиційною моделлю для торгівлі криптовалютою, які використовують книгу ордерів та працюють як посередники між покупцями та продавцями, забезпечуючи інфраструктуру для торгівлі, управління ордерами, зберігання активів і забезпечення ліквідності на особистих

серверах. Даний тип біржі вимагає підтвердження особистих даних для здійснення торгівлі за допомогою системи «Знай свого клієнта».

Такий підхід має декілька переваг перед DEX [13]:

1. Велика база користувачів забезпечує високу ліквідність, що дозволяє виконувати угоди швидко та за вигідною ціною.

2. Інтуїтивно зрозумілі інтерфейси, широкий вибір торгових інструментів такі як: графіки, ліміти, стоп-ордера та підтримка фіатних валют роблять CEX доступними для широкого загалу.

3. Завдяки централізованій інфраструктурі, біржі виконують угоди миттєво, без потреби в підтвердженні транзакцій у блокчейні.

4. CEX забезпечують підтримку клієнтів у разі виникнення проблем таких, як втрата доступу до аккаунту, спірні транзакції та ін.

5. Централізовані платформи часто підтримують велику кількість криптовалют і торгових пар, включаючи рідкісні активи.

Попри деякі переваги, централізованих платформ, вони мають й значні недоліки, які є вирішальними для безпеки та надійності, а саме [14]:

1. Усі активи користувачів зберігаються на серверах платформи, що робить їх мішенню для хакерських атак.

2. CEX стягують комісії за виконання угод, поповнення та виведення коштів.

3. Проходження KYC порушує анонімність користувачів.

4. Централізовані платформи працюють у юрисдикціях, які можуть змінювати свої закони, що створює ризики для роботи платформи та активів користувачів.

Отже, варто відзначити, що як централізовані, так і децентралізовані платформи мають власні переваги та недоліки, які зумовлені як особливостями їх архітектурних рішень, так і базовими принципами функціонування. Централізовані платформи характеризуються високою швидкістю обробки транзакцій, стабільною ліквідністю, простотою в користуванні та зручним інтерфейсом. Вони здатні забезпечити більш легший користувацький досвід,

завдяки централізованому адмініструванню та наявності служби підтримки. Водночас, такі рішення передбачають наявність єдиного центру управління, що створює критичні точки відмови. Це супроводжується ризиками компрометації системи, злому серверної інфраструктури, втрати контролю над коштами або доступу до особистих даних користувачів у разі несанкціонованого доступу або внутрішніх порушень.

На противагу, децентралізовані платформи побудовані на основі блокчейн-технологій, що забезпечує прозорість кожної дії, незмінність записів та відсутність потреби в довірених посередниках. Вони гарантують повну автономію користувачів у зберіганні та розпорядженні цифровими активами, а також підвищену стійкість до цензури, зовнішніх атак і збоїв. Однак разом із перевагами децентралізація несе і низку ризиків, зокрема, складність інтерфейсів, повільніші механізми підтвердження транзакцій, високу комісійну вартість в пікові періоди навантаження, а також фрагментованість ліквідності, що ускладнює великі операції.

З огляду на вищезазначене, доцільним видається застосування комбінованого підходу, що дозволяє інтегрувати сильні сторони обох концепцій. Поєднання централізованих інструментів для обробки даних і оптимізації користувацького досвіду з децентралізованими механізмами забезпечення безпеки, контролю доступу й автоматизації виконання фінансових операцій створює передумови для формування збалансованої, надійної та функціонально повноцінної інфраструктури цифрового обміну активами.

1.3 Аналіз існуючих рішень обміну токенами

Децентралізовані біржі є однією з ключових інновацій у сфері фінансових технологій, що активно розвиваються на базі блокчейн-технологій. На відміну від централізованих бірж, які працюють через посередників і покладаються на централізовані сервери, DEX функціонують через смарт-контракти, забезпечуючи прозорість, децентралізацію та більший контроль користувачів над своїми активами. Дана особливість дозволяє уникнути більшість проблем, властивих централізованим системам, таких як вразливість до хакерських атак, шахрайства або регуляторного тиску. Проте, попри численні переваги, DEX стикаються зі складними викликами у сферах безпеки, продуктивності та зручності використання.

Сьогодні на ринку DEX активно працюють такі платформи, як Uniswap, Sushi Swap, PancakeSwap та Curve [15]. Кожна з них реалізує унікальні підходи до забезпечення ліквідності, безпеки та продуктивності. Наприклад, Uniswap використовує модель AMM, що спрощує обмін активів, тоді як Sushi Swap додає функціонал децентралізованого управління через DAO [16].

Метою подальшого аналізу є детальне вивчення сучасних рішень, які застосовуються для розвитку DEX, їх ключових технологій, переваг і недоліків, а також прикладів успішної реалізації в реальних умовах. Глибоке дослідження дозволить оцінити їхній вплив на ефективність, безпеку й масштабованість платформ, а також виявити існуючі обмеження. Оцінка цих рішень допоможе зрозуміти їх потенціал у довгостроковій перспективі та визначити напрями для подальшого вдосконалення у сфері децентралізованих фінансів, включаючи можливості інтеграції нових технологій, підвищення зручності користування та розширення доступу до глобального ринку.

Uniswap – один із провідних протоколів децентралізованих бірж, що функціонує на основі блокчейну Ethereum. Його робота базується на застосуванні сучасних технологій, які забезпечують його функціональність, децентралізацію та прозорість. Основою платформи є смарт-контракти, які автоматизують процес

обміну токенів без необхідності посередників [17]. Ці смарт-контракти працюють відповідно до наперед визначених правил, що дозволяє забезпечити надійність і відкритість транзакцій. Такий підхід виключає необхідність довіряти централізованим структурам, які можуть бути вразливими до технічних збоїв чи зловживань [18].

Ключовим елементом Uniswap є модель автоматизованого маркетмейкера, яка замінює традиційні ордер-книги, що використовуються на централізованих біржах. У цій моделі ціна токенів визначається математичною формулою, яка враховує співвідношення активів у пулах ліквідності. Це дозволяє виконувати транзакції автоматично, без участі маркетмейкерів, що спрощує торгівлю та робить її доступнішою.

Також Uniswap використовує пули ліквідності, створювані користувачами, які додають свої активи в обмін на комісію з транзакцій. Цей механізм забезпечує ліквідність для платформи, проте не є позбавленим ризиків, таких як тимчасова втрата ліквідності (impermanent loss), яка може виникати через коливання цін активів у пулі [19].

Протокол підтримує широкий спектр токенів стандарту ERC-20, що робить його універсальним для торгівлі в екосистемі Ethereum. Крім цього, Uniswap інтегрує такі технології, як орекли для отримання зовнішніх даних, гнучке управління ліквідністю через впровадження концентрованої ліквідності у версії V3 та механізми захисту від ботів, які прагнуть маніпулювати чергою транзакцій. Однак, платформа не захищена повністю від MEV-атак (наприклад, «сандвіч-атак»), де майнери або валідатори можуть маніпулювати чергою транзакцій для отримання прибутку.

Ще одним обмеженням є залежність Uniswap від мережі Ethereum, яка страждає від високих комісій за газ, особливо під час пікових навантажень. Це значно знижує привабливість платформи для користувачів із невеликим капіталом. Хоча інтеграція рішень другого рівня (Layer 2), таких як Arbitrum і Optimism, частково вирішує цю проблему, масштабування залишається ключовим викликом.

Таким чином, Uniswap, незважаючи на свої переваги, стикається з низкою технологічних і ринкових викликів. Висока залежність від Ethereum, ризики тимчасових втрат ліквідності, атаки типу MEV та високі транзакційні витрати вимагають подальшого вдосконалення. Однак інноваційні підходи, такі як впровадження концентрованої ліквідності та інтеграція Layer 2, демонструють потенціал для розвитку й підвищення ефективності платформи [20].

PancakeSwap є однією з провідних децентралізованих бірж, яка функціонує на Binance Smart Chain (BSC). Як аналог Uniswap, PancakeSwap орієнтований на забезпечення низьких комісій і швидкого виконання транзакцій, використовуючи переваги мережі BSC. Основою роботи платформи є AMM, який дозволяє обмінювати токени через ліквідні пули, створювані користувачами.

Однією з ключових особливостей PancakeSwap є використання власного токена CAKE, який забезпечує додаткові стимули для учасників платформи. Токен використовується для винагороди користувачів, які беруть участь у фармінгу або стейкінгу. Крім того, PancakeSwap інтегрує підтримку NFT, що дозволяє користувачам торгувати невзаємозамінними токенами та використовувати їх у механізмах фармінгу. Такий функціонал значно розширює можливості платформи та робить її універсальним інструментом у сфері децентралізованих фінансів [21].

Завдяки роботі на Binance Smart Chain платформа пропонує низькі комісії за транзакції, що є важливою перевагою для користувачів із невеликим капіталом. Висока швидкодія мережі забезпечує миттєве виконання операцій, навіть під час підвищеного навантаження. Крім цього, PancakeSwap інтегрує кросчейн-містки, що дозволяють взаємодіяти з іншими блокчейнами, розширюючи доступність платформи.

Попри численні переваги, PancakeSwap має й певні недоліки. Одним із найбільш критичних є централізація Binance Smart Chain, яка контролюється Binance [22]. Це створює потенційні ризики для безпеки, оскільки централізовані рішення менш стійкі до цензури або зовнішніх атак. Крім того, платформа стикається з високою конкуренцією, зокрема від інших DEX в екосистемі BSC, таких як BakerySwap або ApeSwap.

Серед прикладів успішної реалізації PancakeSwap виділяються Syrup Pools, які надають користувачам можливість стейкінгу токенів із додатковими винагородами. Завдяки своїй простоті використання, низьким комісіям і універсальності функцій, PancakeSwap став лідером серед децентралізованих бірж у мережі Binance Smart Chain [23].

Оцінюючи ефективність PancakeSwap, можна зазначити, що платформа є ідеальним вибором для користувачів із обмеженим бюджетом і тих, хто прагне уникнути високих витрат на транзакції. Однак довгострокова стабільність платформи залежить від подолання ризиків, пов’язаних із централізацією BSC. Інтеграція нових функцій і розширення можливостей кросчейн-операцій можуть стати вирішальними факторами для її подальшого розвитку.

Отже, аналіз провідних децентралізованих бірж, таких як Uniswap, SushiSwap, PancakeSwap та Curve, демонструє, що кожна з них має свої переваги, які сприяли їх успіху та популярності в екосистемі DeFi. Вони забезпечують користувачам доступ до прозорих, децентралізованих і автоматизованих механізмів торгівлі, надають інструменти для стейкінгу, фармінгу та ліквідності, а також відкривають можливості для залучення широкої аудиторії за рахунок інноваційних технологій, таких як AMM, мультисиг-гаманці, кросчейн-інтеграція та децентралізоване управління.

Однак детальний аналіз також виявляє низку суттєвих недоліків, які обмежують ефективність і довгострокову стійкість цих платформ. Наприклад, Uniswap та SushiSwap страждають від високих комісій і перевантаження мережі Ethereum, що робить їх менш привабливими для користувачів із невеликим капіталом. PancakeSwap, хоча й забезпечує низькі комісії завдяки Binance Smart Chain, стикається з проблемою централізації, яка може становити ризик для довіри користувачів. Curve, незважаючи на свою ефективність у роботі зі стейблкоїнами, має обмежену функціональність і складність у використанні, що знижує його привабливість для новачків. Крім того, спільною проблемою для всіх розглянутих платформ є вразливість до атак типу MEV та інших маніпуляцій із чергою транзакцій. Ці недоліки підкреслюють обмеження існуючих рішень і

показують, що поточні платформи ще не здатні повністю вирішити ключові виклики, такі як масштабованість, безпека та зручність для користувачів.

Зважаючи на проведений аналіз, можна стверджувати, що кожна з розглянутих платформ, зробила значний внесок у розвиток децентралізованих фінансових технологій. Водночас, усі вони мають свої структурні обмеження, пов’язані з питаннями централізації, масштабованості, високих комісій, вразливостей безпеки. Тому для наочного узагальнення результатів порівняльного аналізу в Таблиці 1.1 представлено зіставлення основних характеристик розглянутих платформ з функціональними можливостями розробленого рішення.

Таблиця 1.1 - Порівняльна характеристика платформ конкурентів

Критерій	Uniswap	PancakeSwap	Curve Finance
Блокчейн	Ethereum	BSC	Ethereum, Arbitrum, Polygon
Тип AMM	Постійна добуткова формула x*y=k	Постійна добуткова формула x*y=k	Спеціалізований AMM для стейблкоінів
Підтримка токенів	Обмежена	Обмежена	Підтримує більшість
Говернанс	UNI токен	CAKE токен	CRV токен
Комісії	0.3%	0.25%	0.04%-0.4%
Інструменти безпеки	Bug bounty	Bug bounty	Аудити, перевірки стабільності
Простота використання	Доступний, але не завжди інтуїтивний	Доступний, але не завжди інтуїтивний	Інтерфейс складний, потребує технічних знань
Відкритий код	Так	Так	Так

Отже, хоча існуючі децентралізовані біржі відіграють важливу роль у розвитку DeFi, їх недоліки створюють потребу у новому рішенні, яке зможе подолати обмеження в безпеці.

1.4 Висновки до розділу 1 та постановка задачі

У даному розділі було здійснено детальний аналіз підходів до реалізації обміну токенів у контексті сучасних криптовалютних платформ. Особлива увага була приділена порівнянню централізованих та децентралізованих бірж, що дозволило виявити їхні основні переваги, недоліки та унікальні характеристики. Даний етап був важливим для побудови теоретичної бази щоб в майбутньому розробити децентралізовану платформу для обміну токенів на основі смарт-контрактів з інтеграцією механізмів виявлення аномалій та системи попереджень про підозрілі дії в реальному часі.

На початку розділу було проаналізовано основні принципи роботи блокчейну та смарт-контрактів, які є фундаментальними для функціонування децентралізованих платформ. Блокчейн забезпечує незмінність, прозорість і децентралізацію записів, що робить його ідеальним середовищем для реалізації фінансових операцій. Смарт-контракти, своєю чергою, дозволяють автоматизувати виконання умов угоди без необхідності втручання третіх осіб.

Аналіз також дозволив розглянути ключові компоненти екосистеми AMM-бірж: акторів, активи та технічні модулі. Учасники цієї екосистеми трейдери, провайдери ліквідності, арбітражери та фундація протоколу відіграють важливу роль у забезпеченні стабільності й ефективності роботи платформи. Активи, включаючи торгові токени, токени ліквідності та говернанс токени, забезпечують фінансову основу системи та дозволяють користувачам брати участь у її розвитку. Ключові модулі, такі як пули ліквідності, алгоритми ціноутворення та оракули, забезпечують технічну інфраструктуру для роботи DEX, а також їхню інтеграцію в ширшу екосистему DeFi.

На основі проведеного аналізу можна зробити висновок, що децентралізовані платформи мають значний потенціал для подальшого розвитку завдяки своїй прозорості, доступності та автономності. Однак їхня реалізація вимагає ретельного підходу до проектування, зокрема вирішення проблем з ліквідністю, зручністю використання та безпекою.

Тому для досягнення мети, було визначено основні задачі:

1. Розробка архітектури децентралізованої P2P платформи обміну токенів.
2. Створення набору смарт-контрактів, які забезпечать функціональність обміну токенів та управління пулами ліквідності.
3. Реалізація системи виявлення аномалій у транзакціях у реальному часі, яка надасть змогу ідентифікувати підозрілі дії користувачів.
4. Розробка механізму автоматичного формування сповіщень про інциденти.
5. Забезпечення сумісності платформи з екосистемою DeFi.

Отже, підсумовуючи викладене, можна зазначити, що отримані результати створюють необхідну базу для подальшого проектування децентралізованої платформи для обміну токенів з інтеграцією механізмів виявлення аномалій та системи попереджень про підозрілі дії в реальному часі. Наступний розділ дослідження буде присвячений безпосередньому проектуванню архітектури такої платформи, включаючи розробку смарт-контрактів та інтеграцію з іншими компонентами екосистеми. Реалізація цих задач дозволить створити ефективну платформу, яка відповідатиме сучасним вимогам ринку та сприятиме подальшому розвитку криптовалютних технологій.

2. ПРОЕКТУВАННЯ ДЕЦЕНТРАЛІЗОВАНОЇ ПЛАТФОРМИ ДЛЯ ОБМІНУ ТОКЕНАМИ НА ОСНОВІ СМАРТ-КОНТРАКТІВ З ІНТЕГРАЦІЄЮ МЕХАНІЗМІВ ВИЯВЛЕННЯ АНОМАЛІЙ ТА СИСТЕМИ ПОПЕРЕДЖЕНЬ ПРО ПІДОЗРІЛІ ДІЇ В РЕАЛЬНОМУ ЧАСІ

В даному розділі буде розглянуто основні компоненти системи децентралізовані платформи для обміну токенами з інтеграцією механізмів виявлення аномалій та системи попереджень про підозрілі дії в реальному часі, включаючи мережеву інфраструктуру, способи управління транзакціями та взаємодію з блокчейн-мережею. Окрему увагу приділено безпеці, масштабованості та стабільності роботи платформи, що дозволить краще зрозуміти, як побудувати ефективну та надійну систему, яка буде відповідати сучасним викликам ринку криптовалют.

2.1 Структура та архітектура платформи для обміну токенами

Модель взаємодії компонентів платформи для обміну токенами є одним із ключових процесів розробки децентралізованої P2P-системи, оскільки вона визначає технічну організацію усіх структур, принципи їх взаємодії, логіку обробки даних та забезпечення безпеки транзакцій. Головною метою є створення системи виявлення аномалій в реальному часі, для забезпечення прямого обміну криптовалютами між користувачами без участі посередників, на основі смарт-контрактів з підвищеним рівнем довіри та безпеки.

Платформа матиме багаторівневу модульну архітектуру, яка структурована відповідно до принципів високої розділюваності функціональності та дотримання вимог щодо масштабованості, безпеки, гнучкості й підтримуваності. Такий підхід дає змогу забезпечити чітке розмежування відповідальності між компонентами системи, що є необхідним для розробки критично важливої фінансової інфраструктури. Архітектура платформи поділятиметься на такі базові компоненти:

1. Інтерфейс користувача для забезпечення взаємодії кінцевого користувача із системою через браузер. Відповідає за відображення даних, формування запитів до бекенду та ініціацію транзакцій через інтегровані криптовалютні гаманці.

2. Бекенд-сервер, виконує роль посередника між користувачем, базою даних та блокчейном й забезпечує бізнес-логіку. Обробляє запити користувача, керує створенням та оновленням ордерів, виконує перевірки безпеки, обробляє події від WebSocket-сервера та координує взаємодію з системою виявлення аномалій.

3. Смарт-контракти, виконують роль децентралізованого механізму управління фінансовими операціями. Забезпечують виконання угод згідно з наперед визначеними умовами, що гарантує незмінність та відкритість логіки виконання транзакцій, а також мінімізує ризики втручання сторонніх осіб або зміни умов після їхнього узгодження сторонами.

4. Системи зберігання даних забезпечують збереження критично важливої інформації: даних користувачів, ордерів, історії транзакцій, рейтингів, платіжних реквізитів. Використовується два основні типи сховища: основне реляційне для довготривалих даних, та in-memory-кеш для прискореного доступу до часто використовуваних об’єктів.

5. Модулі реального часу, відповідають за комунікацію між клієнтом і сервером, необхідну для реалізації оновлень у режимі реального часу. Це включає відображення нових ордерів, змін статусу транзакцій, миттєвих повідомлень.

6. Модуль моніторингу та виявлення аномалій, здійснює аналіз транзакційної активності в реальному часі. Його завданням є виявлення нетипових або потенційно шахрайських дій на основі поведінкових шаблонів, частоти угод або ж використання підозрілих IP-адрес. У разі виявлення ризиків модуль автоматично формує сигнали для адміністрації та запускає захисні механізми.

Дана архітектура поєднує у собі централізовані та децентралізовані компоненти, що відповідають за збереження, обробку та логіку управління

даними, з децентралізованими компонентами, які реалізують фінансову взаємодію користувачів у захищеному середовищі блокчейну. Такий підхід дозволяє досягти високого рівня безпеки, прозорості, надійності та ефективності, що є критично важливим для побудови платформи.

Задля глибшого розуміння особливостей функціонування ключових процесів системи, доцільним є представлення структурованих діаграм, що графічно відображають взаємодію між компонентами, характер обробки даних, а також алгоритмічні механізми забезпечення безпеки. Такі візуалізації не лише сприяють кращому усвідомленню логіки роботи платформи, але й дозволяють проаналізувати потенційні вразливості та оцінити ефективність реалізованих рішень. На рисунках 2.1 та 2.2 наведено приклади відповідних діаграм для типових сценаріїв, зокрема процесу автентифікації користувача та створення торгового ордера, що ілюструють послідовність дій, задіяні компоненти й реакції системи на зовнішні події.

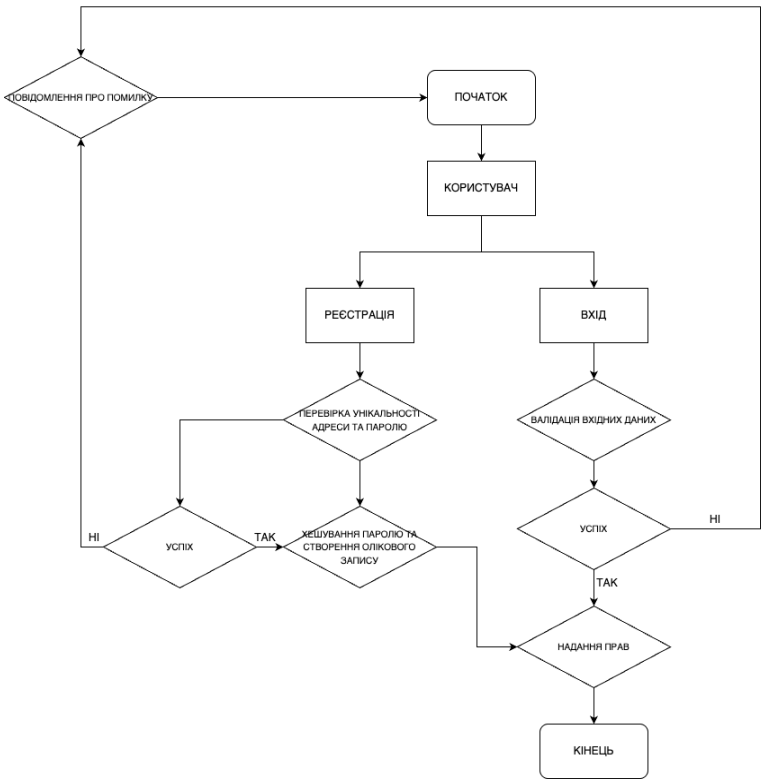


Рис 2.1 - Схема авторизації користувача

Рисунок ілюструє процес авторизації на платформі, що охоплює два ключові сценарії: реєстрацію нового користувача та вхід до системи з існуючим обліковим

записом. Схема демонструє послідовність дій, перевірки з боку системи, а також логіку обробки як успішних, так і неуспішних запитів за наступними етапами.

Крок 1. При переході на web-сторінку ініціюється система з двома можливими варіантами розвитку, де пропонується два варіанти дій: введення даних для реєстрації нового користувача або здійснення входу до вже наявного облікового запису.

Крок 2. Ініціюється один із двох можливих запитів - реєстрація, або вхід. Оскільки кожен сценарій передбачає унікальну послідовність логічних операцій і перевірок, подальші етапи алгоритму описано окремо для кожного випадку.

Реєстрація нового користувача.

Крок 3. Під час запиту, відбувається перевірка унікальності логіну та відповідність паролю встановленим вимогам безпеки: довжина, наявність великих літер, цифр, спеціальних символів.

Крок 4. В разі успішного проходження перевірки застосовується криптографічний алгоритм хешування до паролю, що дозволяє зберігати його у зашифрованому вигляді та створюється новий запис у базі даних.

Крок 5. В випадку помилки валідації система формує повідомлення про помилку, яке повертається у відповідь.

Крок 6. Відображення повідомлення про успішну реєстрацію, після чого користувач може перейти до входу в систему.

Вхід користувача.

Крок 3. Виконується валідація введених даних, порівнюючи логін і хеш паролю з тими, що зберігаються в базі.

Крок 4. Якщо дані некоректні або не відповідають жодному запису, відбувається інформування користувача про помилку авторизації з можливістю повторної спроби входу.

Крок 5. Після успішної авторизації, користувачу призначаються права доступу відповідно до його ролі.

Блок-схема дозволяє наочно прослідкувати не лише логіку переходів між етапами, але й відображає критичні точки прийняття рішень, що мають вплив на безпеку, стабільність і користувацький досвід взаємодії з платформою.

З метою забезпечення безпеки транзакційної діяльності на платформі важливим є не лише контроль доступу до системи, але й постійний моніторинг дій користувачів після авторизації. Одним із критично важливих процесів є виявлення потенційно шахрайських або аномальних транзакцій у режимі реального часу. Нижче представлено блок-схему, що описує логіку роботи модуля моніторингу та виявлення аномалій під час обробки запиту на створення транзакції.

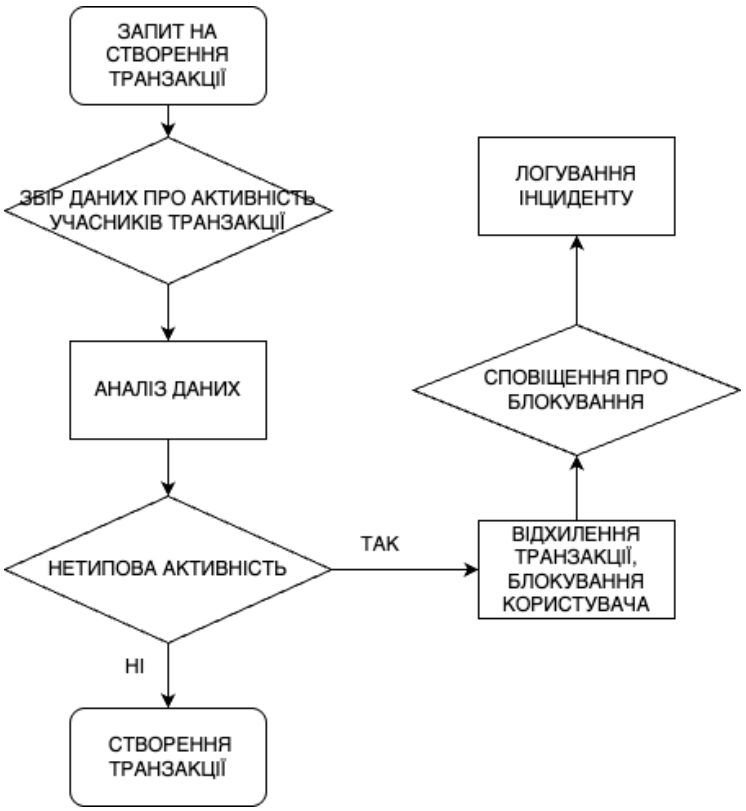


Рисунок 2.2 - Схема виявлення аномальних транзакцій

На рисунку представлено блок-схему функціонування модуля моніторингу та виявлення аномалій, який виконує аналіз транзакційної активності користувачів у режимі реального часу. Схема відображає логіку обробки запиту на створення транзакції з урахуванням перевірки учасників на предмет потенційно підозрілої або шахрайської активності за наступним алгоритмом.

Крок 1. Під час запиту на створення транзакції ініціюється збір усіх релевантних даних. До переліку аналізованої інформації входять як поточні параметри транзакції: сума, адресати, час, IP-адреса, так і історичні дані активності користувачів: частота транзакцій, типові суми, характер взаємодій з іншими учасниками.

Крок 2. На основі зібраної інформації виконується аналітична оцінка поведінки користувача. Виявлення аномалій ґрунтується основі rule-based systems, що дозволяє визначити відхилення від звичних шаблонів поведінки.

Крок 3. При виявленні підозрілих дій, виконується автоматичне відхилення транзакції. Одночасно ініціюється тимчасове, з можливістю подальшого продовження, блокування користувача, для запобігання потенційним ризикованим діям.

Крок 4. Формується детальне сповіщення про інцидент, що включає всю критично важливу інформацію: дані про транзакцію, тип виявленої аномалії, ID користувача, часові мітки та IP-адресу. Далі повідомлення логується та автоматично надсилається адміністрації.

Крок 5. У разі відсутності нетипової активності, транзакція вважається безпечною і створюється відповідно до стандартної процедури. Користувач отримує підтвердження про успішне виконання дії.

Отже, наведена блок-схема демонструє не лише технічну логіку перевірки транзакцій, а й візуалізує критичні точки прийняття рішень, які напряму впливають на загальний рівень безпеки платформи, довіру користувачів і захист від шахрайських дій. В поєднанні з алгоритмами авторизації, даний модуль формує цілісну систему контролю доступу і транзакційної безпеки, що є ключовим фактором стабільного функціонування платформи.

2.2 Проектування алгоритму функціонування платформи для обміну токенами

Функціонування децентралізованої P2P-платформи для обміну токенами базується на інтегрованій системі алгоритмів, що забезпечують безперебійну взаємодію між користувачами, безпечне проведення транзакцій та оперативне виявлення аномальних дій. Основними етапами алгоритму роботи платформи є наступні процеси:

Реєстрація та аутентифікація користувачів. Процедура базується на поєднанні механізмів класичної та Web3-автентифікації. Основною ідентифікаційною інформацією користувача є його криптовалютний гаманець та пароль, що забезпечує підвищену безпеку й зручність використання. Процес реєстрації починається з введення користувачем адреси криптовалютного гаманця та встановлення унікального пароля. На основі цих даних створюється обліковий запис, який прив’язується до зазначеного гаманця. Для забезпечення безпеки пароль хешується за допомогою криптографічних алгоритму SHA-256 перед його збереженням у базі даних. Додатково передбачена можливість підтвердження володіння гаманцем шляхом підписання унікального запиту nonce за допомогою приватного ключа, що перевіряється через бібліотеки Web3.js. Даний підхід дозволяє платформі впевнитися у справжності користувача без передачі конфіденційних даних. При аутентифікації користувач вводить адресу свого гаманця та пароль, після чого система звіряє введені дані з хеш-значенням, збереженим у базі даних. Успішний вхід користувача ініціює генерацію JWT токена доступу із визначеним терміном дії, що дозволяє підтримувати активну сесію. Для управління сеансами та захисту від несанкціонованого доступу використовується WebSocket-з'єднання, яке забезпечує миттєве відключення користувача у випадку виявлення підозрілої активності.

Платформа підтримує інтеграцію з децентралізованими криптовалютними гаманцями, що забезпечує можливість здійснення транзакцій безпосередньо через блокчейн, мінімізуючи залежність від централізованих серверів і підвищуючи

рівень безпеки. Основою цієї інтеграції є використання бібліотеки Web3.js, яка дозволяє взаємодіяти зі смарт-контрактами та блокчейн-мережею безпосередньо з клієнтського боку. Це забезпечує користувачам зручний механізм підключення їхніх криптовалютних гаманців, таких як MetaMas чи Trust Wallet, без необхідності зберігання приватних ключів на платформі. Для кожної угоди платформа генерує унікальний escrow-гаманець, який функціонує як тимчасове сховище криптовалютних активів до моменту завершення угоди. Цей гаманець створюється через смарт-контракт, що дозволяє блокувати кошти продавця до виконання всіх умов угоди. Такий підхід забезпечує автоматизацію та прозорість процесу обміну, оскільки токени залишаються заблокованими в смарт-контракті до підтвердження виконання угоди, після чого вони автоматично передаються покупцеві або повертаються продавцю у разі скасування операції. Крім того, система забезпечує механізм відстеження балансу та історії транзакцій користувача через інтеграцію з блокчейн-експлорерами, такими як Etherscan та BscScan. Це дозволяє користувачам у режимі реального часу отримувати актуальні дані щодо статусу їхніх транзакцій, підтвердження операцій у блокчейні та історії фінансових дій, що сприяє прозорості угод, оскільки всі дії зафіксовані у розподіленому реєстрі, що унеможливлює їхню фальсифікацію або приховування.

Створення та розміщення ордерів. Процес створення торгових ордерів на платформі реалізується через децентралізований механізм розміщення заявок, що забезпечує автономне управління обмінними операціями без посередників. Користувач, який бажає здійснити операцію купівлі або продажу криптовалюти, спочатку визначає основні параметри ордера, зокрема тип угоди: купівля або продаж, обрану криптовалюту, обсяг транзакції та ціну, що дозволяє гнучко налаштовувати стратегію торгівлі відповідно до поточних ринкових умов. Додатково встановлюються умови виконання угоди, включаючи вибір способів оплати, а також мінімальні та максимальні ліміти транзакції, що забезпечують контроль за фінансовими ризиками. Користувач також може задавати термін дії ордера, після закінчення якого він автоматично анулюється у разі відсутності відповідного контрагента. Після внесення всіх параметрів дані ордера

записуються у базу даних PostgreSQL, що забезпечує надійне зберігання інформації та швидкий доступ до історичних записів. Оскільки швидкість обробки запитів є критичною для роботи торгової системи, використовується Redis для кешування активних ордерів, що дозволяє мінімізувати затримки при їхньому пошуку та обробці. Для забезпечення динамічного оновлення списку ордерів у режимі реального часу використовується WebSocket-з'єднання, яке дозволяє всім підключеним користувачам негайно отримувати актуальну інформацію про нові або змінені заявки. Це усуває необхідність постійного оновлення сторінки та забезпечує високу швидкість реакції на зміни ринку, що є ключовим фактором для ефективної P2P-торгівлі. Таким чином, децентралізований механізм розміщення ордерів забезпечує автономність, гнучкість і прозорість торгових операцій, а інтеграція з базами даних і WebSocket дозволяє підтримувати високу продуктивність і швидкість взаємодії між користувачами платформи.

Зіставлення та виконання ордерів. Процес пошуку відповідного контрагента та здійснення угоди на децентралізованій P2P-платформі реалізується через ручне зіставлення ордерів, що забезпечує ефективне узгодження заявок на купівлю та продаж криптовалют. Користувач переглядає список активних ордерів у режимі реального часу, що забезпечується завдяки WebSocket-з’єднанню, яке дозволяє отримувати актуальні дані без необхідності повторного завантаження сторінки. При виборі відповідного ордера система проводить попередній аналіз його доступності та відповідності параметрам транзакції, включаючи валюту, обсяг, ліміти та спосіб оплати. Перед остаточним підтвердженням угоди платформа здійснює оцінку надійності контрагента, аналізуючи його історію попередніх угод, рейтинг, активність та потенційні ризики. У рамках цього процесу застосовуються алгоритми поведінкової аналітики, які дозволяють виявити нетипові або підозрілі шаблони торгівлі, що можуть свідчити про шахрайські дії. Для гарантування безпеки транзакцій використовується механізм escrow, який передбачає блокування токенів продавця на спеціальному смарт-контракті до моменту завершення угоди. Це виключає можливість шахрайства з боку однієї зі сторін та

забезпечує справедливий розподіл коштів відповідно до заздалегідь визначених умов. У момент ініціювання угоди продавець підтверджує свій намір продати активи, після чого смарт-контракт тимчасово заморожує відповідну суму на його рахунку до виконання усіх умов обміну. Після того як покупець здійснює платіж відповідно до обраного способу оплати, він надає підтвердження транзакції через платформу. Смарт-контракт перевіряє статус угоди та, у разі відповідності всіх умов, автоматично розподіляє токени на гаманець покупця. Якщо ж транзакція не відповідає очікуваним критеріям або виникають спірні ситуації, угода може бути переведена в режим арбітражу, що дозволяє залучити систему розв'язання конфліктів або DAO-голосування для ухвалення рішення. Таким чином, алгоритм зіставлення та виконання ордерів у рамках P2P-платформи забезпечує безпеку, прозорість та автоматизацію процесу торгівлі, знижуючи ризики шахрайства та забезпечуючи ефективний механізм взаємодії між учасниками ринку. Інтеграція смарт-контрактів, аналітичних інструментів та escrow-механізму сприяє створенню довірчого середовища для проведення безпечних фінансових операцій.

Здійснення P2P-транзакцій. Децентралізована модель P2P-транзакцій дозволяє користувачам здійснювати обмін криптовалютами без залучення посередників, що підвищує рівень безпеки, мінімізує комісійні витрати та забезпечує повну автономність угод. Функціонування цього механізму базується на використанні смарт-контрактів, які гарантують виконання фінансових зобов'язань між сторонами відповідно до заздалегідь визначених умов. Процес ініціюється покупцем, який обирає відповідний ордер та підтверджує намір здійснити угоду. Після цього покупець проводить оплату згідно з узгодженими умовами, використовуючи один із доступних методів. На цьому етапі платформа вимагає підтвердження платежу, таке як завантаження квитанції. Наступний етап передбачає підтвердження отримання коштів продавцем. Після перевірки факту надходження оплати продавець дає згоду на завершення угоди, що автоматично активує виконання смарт-контракту escrow. У разі виникнення спірних ситуацій або підозри на шахрайські дії система може тимчасово заморозити операцію та передати її на розгляд арбітражного механізму або DAO-голосування. Після

підтвердження угоди смарт-контракт автоматично розблоковує токени та здійснює їх переказ на гаманець покупця. Даний процес є повністю автоматизованим та захищеним від несанкціонованого втручання, оскільки всі дії закріплені у програмному коді смарт-контракту та виконуються у відповідності до заздалегідь визначених умов. Всі транзакції реєструються у блокчейні, що забезпечує повну прозорість та незмінність історії операцій. Це дозволяє будь-якому користувачеві перевірити статус угоди, підтвердити її достовірність та виключає можливість фальсифікації даних. Крім того, блокчейн-фіксація сприяє підвищенню рівня довіри між користувачами, оскільки вся історія фінансових операцій доступна для незалежного аудиту. Таким чином, механізм P2P-транзакцій у децентралізованій платформі дозволяє забезпечити високий рівень безпеки, автоматизацію та прозорість, що робить його надійним інструментом для здійснення криптовалютних обмінів без ризику несанкціонованого втручання або шахрайських дій.

Моніторинг та виявлення аномалій, генерація сповіщень про підозрілі дії. Ефективне функціонування децентралізованої P2P-платформи для обміну криптовалют потребує надійного механізму моніторингу транзакцій та виявлення потенційних загроз, що можуть виникати внаслідок шахрайських дій, маніпуляцій ринком або інших зловмисних активностей. Для цього платформа інтегрує систему аналізу ризиків, що забезпечує автоматичний контроль за фінансовими операціями та дозволяє оперативно реагувати на виявлені аномалії. Процес моніторингу відбувається у реальному часі завдяки використанню параметрів поведінкової аналітики, що включають аналіз транзакцій, частоти операцій, сумових лімітів та історії взаємодій користувачів.

Основними аномальними патернами, які аналізує система, є:

1. Нестандартна частота угод, що може свідчити про маніпуляцію обсягами торгів або використання автоматизованих ботів.

2. Маніпуляція ринковими цінами, зокрема штучне завищення або заниження курсу через створення великої кількості ордерів без реального наміру виконання.

3. Використання одного IP-адресу для кількох акаунтів, що може вказувати на створення множинних профілів для обходу обмежень.

4. Раптові великі транзакції з нових або малорейтингових акаунтів, що може вказувати на шахрайство або спробу відмивання коштів, тобто врахування AML ризиків.

5. Різка зміна торгової активності, коли користувач, який раніше не здійснював значних операцій, починає активно торгувати великими сумами.

6. Часте скасування угод перед їх виконанням, що може вказувати на використання платформи для маніпулювання ринком або тестування системи без реальних намірів здійснення угоди.

7. Транзакції між одним і тим самим набором адрес, що може сигналізувати про спробу "кругової торгівлі" (wash trading) з метою створення фіктивної активності.

8. Використання VPN або TOR для приховування справжнього розташування користувача, що може бути ознакою шахрайства або спроби уникнути AML-перевірок.

9. Непропорційне використання незвичних методів оплати, наприклад, якщо користувач раптово переходить на використання менш популярних або ризикованих платіжних систем.

10. Невідповідність між геолокацією та вибраним платіжним методом, що може свідчити про використання викрадених платіжних реквізитів або шахрайські дії.

Якщо система визначає високу ймовірність шахрайських дій, відповідний акаунт піддається автоматичному тимчасовому блокуванню до завершення додаткової перевірки. Це запобігає потенційним фінансовим втратам інших користувачів та мінімізує можливість проведення зловмисних операцій. Для забезпечення швидкого реагування на підозрілі дії платформа включає систему автоматичних сповіщень, що дозволяє як користувачам, так і адміністраторам оперативно отримувати інформацію про можливі загрози. У разі виявлення шахрайських дій активується тригерна система попереджень, яка може

автоматично блокувати транзакції або вимагати додаткового підтвердження особи користувача. Повідомлення про підозрілі дії надсилаються через Telegram-бот, що дозволяє користувачам миттєво отримувати інформацію про необхідність вжиття заходів безпеки. Крім того, WebSocket-з'єднання використовується для інтеграції з клієнтським інтерфейсом, що забезпечує оперативне відображення попереджень без необхідності оновлення сторінки або надсилання запитів до сервера. Усі зафіксовані аномальні транзакції автоматично додаються до звітів про ризики, які відображаються в адміністративній панелі платформи. Ці звіти містять докладну інформацію про потенційно підозрілі акаунти, історію їхніх операцій, рівень ризику та рекомендації щодо подальших дій. Таким чином, платформа забезпечує високий рівень безпеки, зменшуючи ризик шахрайських дій та покращуючи користувацький досвід завдяки своєчасному інформуванню про потенційні загрози.

2.3 Обґрунтування вибору технологій та мови програмування

В процесі побудови архітектури децентралізованої P2P-платформи для обміну криптовалют ключовим завданням стало обґрунтоване визначення технологічного стеку для реалізації серверної частини застосунку, зокрема фреймворку та мови програмування. Основними критеріями для вибору виступали: масштабованість, підтримка модульності, зручність роботи з асинхронними подіями, типізація, продуктивність та безпека. Враховуючи дані вимоги, для реалізації бекенд-частини платформи було обрано NestJS у поєднанні з мовою програмування TypeScript.

NestJS — це прогресивний фреймворк для Node.js, який використовує модульну архітектуру та побудований на основі принципів об’єктно-орієнтованого програмування, функціонального програмування та реактивного програмування [24]. Його основною перевагою є висока структурованість і масштабованість, що дозволяє організовувати складні проєкти з великою кількістю взаємодіючих компонентів: модулі, сервіси, контролери, шлюзи у зрозумілий і підтримуваний код. NestJS забезпечує розділення логіки за доменами, що є особливо корисним у контексті платформи з кількома великими модулями — такими як управління ордерами, транзакціями, користувачами, аналітикою та WebSocket-з’єднаннями.

Однією з ключових технічних переваг NestJS є вбудована підтримка WebSocket Gateways, що дозволяє реалізувати обмін даними в реальному часі, критичний для P2P-платформи з динамічно оновлюваними ордерами та повідомленнями про статуси угод. Крім того, фреймворк має нативну інтеграцію з бібліотеками типу class-validator, rxjs, TypeORM, Prisma, Passport тощо, що дозволяє швидко реалізувати ключову функціональність: валідацію, роботу з БД, автентифікацію тощо, із дотриманням високих стандартів безпеки та якості коду.

NestJS також підтримує декоратори та інверсію управління (IoC) завдяки використанню контейнерів залежностей, що спрощує тестування, розширюваність і повторне використання логіки. Це особливо важливо для складних систем із

великою кількістю сервісів, які взаємодіють між собою та потребують чіткої ізоляції відповідальностей.

У свою чергу, використання TypeScript як мови програмування надає важливу перевагу у вигляді статичної типізації, що дозволяє виявляти помилки на етапі компіляції, знижуючи ймовірність появи критичних багів у продакшн-середовищі. TypeScript значно покращує читабельність та підтримуваність коду, особливо в командних розробках, де чітко визначені типи сутностей (DTO, API-відповідей, контрактів із БД) зменшують когнітивне навантаження при масштабуванні проєкту [25]. До того ж, TS чудово інтегрується з редакторами коду VS Code, надаючи розширені можливості автодоповнення, рефакторингу та навігації.

Крім цього, NestJS і TypeScript мають широку й активну спільноту, велику кількість відкритих рішень, шаблонів і модулів, що дозволяє швидко вирішувати типові задачі й адаптувати систему до змін вимог. Вони також підтримуються великими компаніями та мають хорошу документацію, що гарантує стабільність розвитку проєкту в довгостроковій перспективі.

Для забезпечення цілісності транзакцій, підтримки складної реляційної структури даних, низької затримки при обробці великої кількості запитів у режимі реального часу, масштабованість, а також безпечність роботи з чутливою інформацією обрано комбінацію PostgreSQL як основної системи управління реляційними базами даних та Redis як високопродуктивного інструменту кешування та управління тимчасовими даними.

PostgreSQL — це потужна об'єктно-реляційна система управління базами даних з відкритим вихідним кодом, яка широко використовується у фінансових, банківських та блокчейн-рішеннях завдяки високій надійності, стабільності та широкому спектру функціональних можливостей [26]. Однією з ключових переваг PostgreSQL є підтримка ACID-транзакцій (атомарність, узгодженість, ізольованість, довговічність), що є обов’язковою умовою для систем, у яких обробляються фінансові дані та взаємодії між користувачами. Це забезпечує точність та достовірність кожної транзакції, що фіксується на платформі. Також

PostgreSQL відзначається високим рівнем підтримки індексації, складних запитів, зовнішніх ключів і збережених процедур, що дає змогу ефективно моделювати складні взаємозв’язки між сутностями: користувачами, ордерами, транзакціями, рейтингами тощо.

Особливо актуальною для динамічного середовища криптовалютного обміну є підтримка формату JSONB, яка дозволяє поєднувати переваги реляційної та документно-орієнтованої моделей даних. Це спрощує зберігання гнучких структур, наприклад, параметрів платіжних методів або історичних змін курсу, без необхідності створювати додаткові таблиці або ускладнювати схеми. Крім того, PostgreSQL підтримує реплікацію, шардінг та горизонтальне масштабування, що дозволяє адаптувати систему до зростання кількості користувачів та обсягу даних без втрати продуктивності.

Однак, при високих навантаженнях, особливо в режимі реального часу наприклад, при оновленні ордербуку, перевірці доступних ордерів або обробці сеансів користувачів, запити до основної бази даних можуть створювати затримки. Для розв’язання цієї проблеми платформа використовує Redis — in-memory систему зберігання даних, яка працює в режимі ключ-значення та призначена для обробки даних із наднизькою затримкою.

Redis ефективно виконує функції кешування найбільш затребуваної інформації (активні ордери, рейтинги користувачів, статуси сесій), що дозволяє мінімізувати кількість звернень до основної бази даних та істотно зменшує час відповіді системи. Крім того, Redis підтримує механізм TTL, що дозволяє задавати терміни дії для тимчасових об’єктів наприклад, OTP-коди, тимчасові токени доступу), забезпечуючи автоматичне очищення застарілих даних [27]. Також Redis може використовуватись для реалізації черг повідомлень, лімітів на запити (rate limiting), а також механізмів pub/sub (publish/subscribe), що відкриває додаткові можливості для масштабування платформи та роботи з WebSocket-з'єднаннями.

Інтеграція PostgreSQL і Redis створює надійну та високопродуктивну архітектуру, у якій PostgreSQL виконує роль основного сховища довготривалих даних, а Redis відповідає за високошвидкісний доступ до критично важливої

інформації. Такий підхід є загальноприйнятим у проєктах, де потрібно одночасно забезпечити транзакційну цілісність, багатий функціонал обробки даних та високу швидкодію. Обидві технології мають широку спільноту підтримки, готові бібліотеки для інтеграції з NestJS та активно використовуються у сучасних фінансових, трейдингових і блокчейн-платформах.

Невід'ємним компонентом децентралізованої архітектури є смарт-контракти, які забезпечують довіру між учасниками P2P-обміну без необхідності залучення третіх сторін. Для реалізації смарт-контрактів у рамках даної платформи було обрано мову програмування Solidity, що є галузевим стандартом для розробки децентралізованих застосунків dApps на основі Ethereum Virtual Machine.

Solidity — це високорівнева мова програмування, спеціально створена для написання смарт-контрактів, що виконуються у блокчейнах, Ethereum, Binance Smart Chain, Polygon, Avalanche. Основною причиною вибору саме цієї мови є широка підтримка в екосистемі децентралізованих фінансів DeFi, а також її повна інтеграція з такими популярними інструментами, як Remix, Ethers.js, Web3.js, OpenZeppelin [28].

Однією з ключових переваг Solidity є детермінованість виконання коду та можливість програмування логіки за фіксованими умовами, що є фундаментальним для роботи таких механізмів, як escrow, DAO-голосування, заморожування коштів, розподіл активів та інші. У випадку P2P-платформи, де токени продавця блокуються до підтвердження оплати, саме Solidity дозволяє реалізувати автоматичний, незмінний та прозорий алгоритм взаємодії сторін, захищаючи їх від недобросовісних дій.

Також важливо, що Solidity має розвинену систему безпеки, яка з кожною новою версією вдосконалюється, а також підтримує використання перевірених шаблонів коду з бібліотеки OpenZeppelin, які значно зменшують ризики помилок і вразливостей найпоширеніші з яких reentrancy, overflow/underflow, підробка підписі. Бібліотеки забезпечують стандартизовані контракти для токенів ERC-20, ролей доступу, механізмів паузи, власності, дозволяючи зосередитися на

бізнес-логіці платформи, не витрачаючи ресурси на повторне створення типових конструкцій.

Ще одним важливим аргументом на користь Solidity є підтримка великої міжнародної спільноти розробників, що дає змогу швидко знаходити рішення на GitHub, Stack Overflow або форумах, а також підвищує шанси на подальшу інтеграцію із зовнішніми сервісами.

Крім того, Solidity дозволяє створювати модульні та повторно використовувані контракти, що критично важливо при реалізації складної логіки, зокрема для обробки ордерів, арбітражу, рейтингів, DAO-голосування та механізмів виявлення аномалій. Контракти можна оновлювати через проксі-моделі Upgradeable Proxy, що дозволяє вносити покращення без необхідності розгортання нової версії платформи.

Таким чином, вибір Solidity як мови для розробки смарт-контрактів є повністю виправданим з погляду технічної надійності, безпеки, сумісності з популярними блокчейнами, підтримки розвиненої інфраструктури та можливості масштабування. Завдяки широкій екосистемі інструментів, наявності перевірених бібліотек, зокрема, OpenZeppelin і підтримці стандартів: ERC-20, ERC-721, ERC-1155. Solidity дозволяє створювати прозору та автоматизовану логіку взаємодії між користувачами, що критично важливо для escrow-механізмів, DAO-голосування, безпечної передачі токенів та реалізації децентралізованих фінансових операцій [29].

У поєднанні з NestJS та TypeScript на стороні серверної логіки, які забезпечують структуровану, типізовану, модульну та безпечну архітектуру, адаптовану до обробки подій у реальному часі та високонавантажених фінансових процесів, а також з використанням PostgreSQL і Redis як надійних рішень для зберігання та кешування даних, формується цілісна, масштабована та надійна технологічна платформа. Така комбінація дозволяє реалізувати повноцінну децентралізовану екосистему для P2P-обміну криптовалютами, яка поєднує в собі переваги швидкодії, гнучкості, безпеки та прозорості.

2.4 Висновки до розділу 2

У даному розділі було здійснено всебічне технічне обґрунтування підходів до проєктування та реалізації децентралізованої P2P-платформи обміну токенами, яка поєднує в собі високі вимоги до продуктивності, безпеки, масштабованості та зручності для кінцевого користувача. Детально проаналізовано структурно-архітектурну модель системи, яка базується на багаторівневій модульній побудові з чітким розмежуванням функціональних зон відповідальності. Запропонована архітектура інтегрує централізовані компоненти такі як: сервер логіки, база даних, модуль реального часу, для забезпечення швидкої обробки запитів і синхронізації даних, з децентралізованими смарт-контрактами, які гарантують прозорість, незмінність і захищеність фінансових взаємодій між користувачами.

Спроектовано ключові алгоритми платформи, що забезпечують її цілісну функціональну поведінку. Розроблено логіку автентифікації користувачів з використанням криптовалютних гаманців, алгоритми створення й обробки ордерів на купівлю/продаж токенів, сценарії взаємодії зі смарт-контрактами, а також механізми виконання транзакцій у режимі escrow. Особливу увагу приділено модулю моніторингу аномалій, який застосовує методи поведінкової аналітики для виявлення підозрілих дій у реальному часі. Такий підхід дозволяє не лише підвищити загальний рівень безпеки, але й адаптивно реагувати на динаміку ризиків, характерних для децентралізованого фінансового середовища.

Було обґрунтовано доцільність вибору інструментів і технологій, які формують технологічну основу платформи. Зокрема, використання Solidity для написання смарт-контрактів забезпечує повну сумісність з блокчейн-мережами на базі EVM та дає змогу реалізовувати надійні та безпечні механізми управління активами користувачів. Вибір NestJS у поєднанні з TypeScript гарантує високий рівень модульності, типізованості та структурованості серверної логіки, що полегшує розробку, тестування та масштабування системи. Водночас використання PostgreSQL як основного реляційного сховища даних забезпечує

цілісність, узгодженість та надійність зберігання інформації, а інтеграція Redis значно підвищує продуктивність системи за рахунок високошвидкісного кешування й обробки подій у реальному часі.

Загалом, сформована технічна концепція платформи демонструє цілісність та узгодженість обраних рішень, що дозволяє ефективно реалізувати функціональність безпечного, прозорого та масштабованого P2P-обміну токенами. Архітектурні підходи, алгоритмічна логіка та технологічний стек платформи є адаптованими до сучасних викликів криптовалютного ринку, а також закладають фундамент для її подальшого розвитку, інтеграції з іншими сервісами, підтримки додаткових токенів та розширення функціональних можливостей у рамках Web3-екосистеми.

3. РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

Даний розділ присвячено практичній реалізації децентралізованої P2P-платформи обміну криптовалют. У розділі детально розглянуто вибір інструментів розробки, архітектурні рішення, реалізацію ключових модулів системи та процес перевірки їхньої працездатності.

Розробка охоплює створення клієнтської частини з інтуїтивно зрозумілим інтерфейсом, серверної логіки з обробкою запитів та взаємодією зі смарт-контрактом, а також функціоналу виявлення аномальної активності користувачів. Окрема увага приділяється етапу тестування, що забезпечує підтвердження надійності реалізованого функціоналу та відповідність розробленого продукту поставленим завданням.

3.1 Середовище розробки

Інтегроване середовище розробки (IDE) — це програмне середовище, яке об'єднує в собі низку інструментів, необхідних для ефективного створення, тестування та розгортання програмного забезпечення. Серед основних функцій сучасних IDE — підсвітка синтаксису, автодоповнення коду, налагодження, компіляція, керування файлами проєкту, інтеграція з системами контролю версій, що забезпечує повний цикл розробки в межах єдиного інтерфейсу. Використання IDE значно підвищує продуктивність розробки, забезпечуючи зручну інтеграцію процесів і дозволяючи зберегти фокус безпосередньо на проектуванні логіки, а не на налаштуванні інструментів.

В рамках реалізації дипломної роботи було використано два середовища розробки, кожне з яких виконуватиме специфічну функцію в межах загальної архітектури програмного забезпечення. Visual Studio Code (VS Code) використовуватиметься для реалізації клієнт-серверної частин платформи, зокрема для розробки інтерфейсу користувача та логіки обробки запитів на стороні сервера. Натомість Remix IDE буде залучено для розробки, тестування та

розгортання смарт-контрактів у блокчейн-середовище, що забезпечить ефективну інтеграцію функціоналу децентралізованих фінансових операцій у проєкт.

Детально розглянемо VS Code як основне середовище розробки клієнт-серверної частини програмного забезпечення. Це легке, крос платформне інтегроване середовище розробки з відкритим вихідним кодом, створене компанією Microsoft. Основною перевагою якого є невелика вага, яка забезпечує функціональність повноцінного IDE, оскільки підтримує широкий спектр мов програмування, розширень та інструментів для розробки сучасних веб застосунків.

Інтерфейс VS Code (рис. 3.1) складається з таких основних елементів [30]:

1. Панель провідника, дозволяє переглядати структуру проєкту, відкривати файли та працювати з каталогами.
2. Редактор коду, забезпечує підсвітку синтаксису, автодоповнення, підтримку фреймворків і можливість роботи з декількома файлами одночасно.
3. Вбудований термінал, дозволяє запускати команди безпосередньо у середовищі розробки.
4. Система розширень, забезпечує швидке розширення функціональності середовища та надає можливість інсталяції плагінів для роботи з NestJS, React, Prettier, ESLint, GitLens та іншими технологіями.
5. Інструменти налагодження, забезпечують можливість покрокової перевірки виконання коду, виставлення точок зупину (breakpoints) і моніторингу змінних у реальному часі.
6. Інтеграція з системами контролю версій, дозволяє виконувати всі операції з репозиторіями (створення гілок, коміти, перегляд історії змін) безпосередньо в межах редактора.

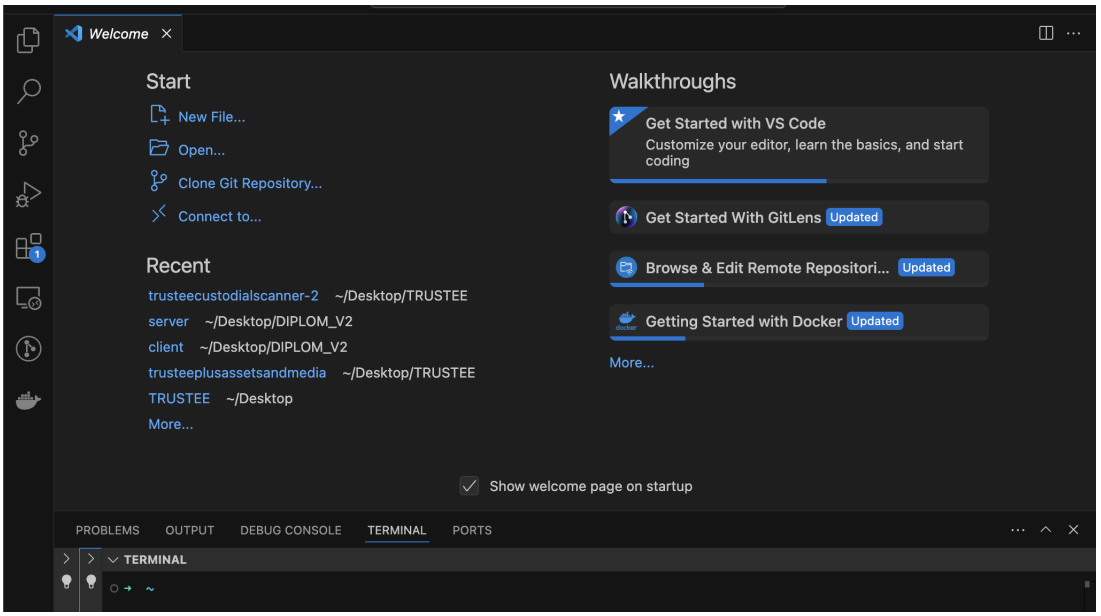


Рисунок 3.1 - Загальний вигляд інтерфейсу Visual Studio Code.

Даний інтерфейс дозволяє працювати з проєктом у зручному та гнучкому середовищі, де всі основні інструменти доступні в межах одного вікна. Комбінація візуальної навігації, підтримки багатьох мов програмування, а також інтегрованих засобів тестування й керування версіями робить VS Code універсальним рішенням для повноцінного циклу розробки.

На рисунку 3.2 представлено приклад роботи із серверною частиною застосунку, реалізованою на основі фреймворку NestJS.

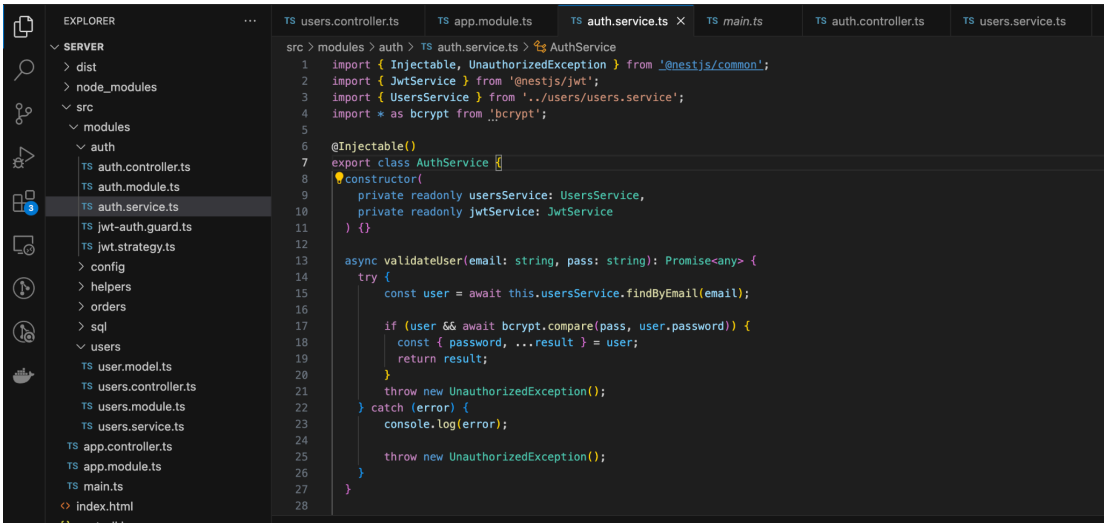


Рисунок 3.2 - Робота з серверною частиною на базі NestJS у VS Code.

На рисунку зображено середовище розробки, відкритий проєкт на основі фреймворку NestJS, який реалізує архітектурний підхід Modular Monolith. Ліворуч

представлена деревоова структура каталогів, що відображає логічну організацію коду за модулями — auth, users, orders, sql. Така структура відповідає принципам NestJS, де кожен модуль ізольований та містить контролери, сервіси й додаткові об'єкти, що спрощує масштабування та підтримку коду.

У центральній частині інтерфейсу відкритий файл auth.service.ts, який містить реалізацію сервісу для аутентифікації. VS Code автоматично підсвічує синтаксис, надає автодоповнення та імпортує залежності, що значно прискорює написання коду та зменшує кількість синтаксичних помилок.

Крім того, активні вкладки у верхній частині вікна дозволяють швидко перемикатися між кількома файлами, а вбудований термінал, дає змогу запускати сервер, виконувати команди CLI, а також інтегрувати дебагер для покрокового налагодження. Така організація робочого простору дає змогу одночасно працювати з логікою, структурою та процесом запуску проєкту без необхідності перемикання між окремими програмами.

Для розробки та тестування смарт-контрактів було використано Remix IDE — спеціалізоване веб-орієнтоване середовище для створення, тестування та розгортання смарт-контрактів на мові Solidity у мережі Ethereum або сумісних блокчейн-платформах. Remix IDE активно використовується в розробці децентралізованих застосунків. Основними перевагами є простота, широкий функціонал та інтеграція з блокчейн-мережами з можливістю миттєвого розгортання контрактів. Можливість швидко компілювати, розгортати та перевіряти функціональність контрактів без необхідності налаштовувати локальне середовище. Це особливо важливо в контексті проєктів, де необхідна тісна інтеграція між класичними веб-технологіями та децентралізованою логікою. Інтерфейс Remix IDE (рис. 3.3) умовно можна поділити на такі ключові компоненти:

1. Панель навігації, яка включає розділи для управління файлами, компіляції, деплою контрактів, перегляду помилок і налаштувань середовища.

2. Редактор коду, з підтримкою підсвітки синтаксису Solidity, автодоповненням і перевіркою помилок у реальному часі.

3. Інструменти компіляції та розгортання, які дозволяють обрати версію компілятора, налаштувати параметри контракту, а також виконати розгортання як у вбудовану тестову мережу, так і в зовнішню мережу наприклад, Sepolia або локальний Ganache.

4. Консоль виводу, що забезпечує візуалізацію результатів викликів функцій контракту, відображає лог транзакцій, подій, повідомлення про помилки або попередження.

5. Інтеграція з MetaMask, яка дає змогу використовувати зовнішні гаманці для підпису транзакцій і тестування контрактів у більш реалістичних умовах.

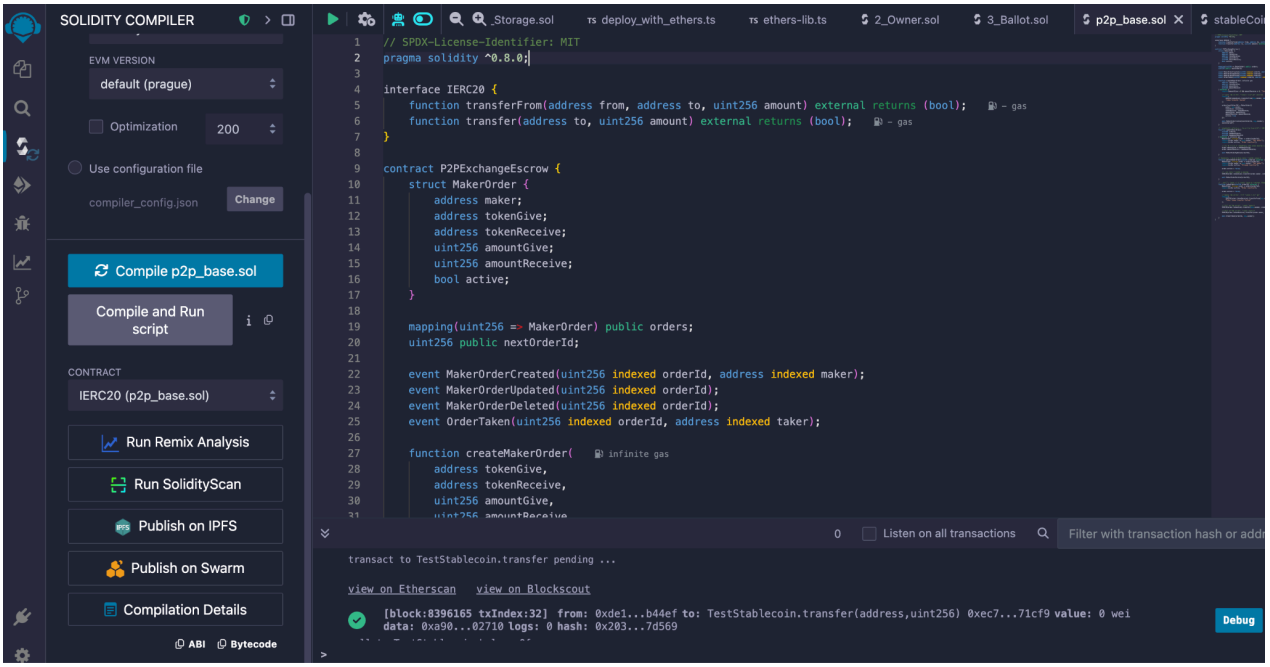


Рисунок 3.3 – Інтерфейс Remix IDE для роботи зі смарт-контрактами

На рисунку зображено приклад створення та тестування контракту P2PExchange. Контракт реалізує логіку створення ордерів, підтвердження з боку тейкера та обміну активів. Через панель Deploy & Run Transactions здійснюється розгортання контракту, ініціалізація стану та виклик методів, таких як createOrder, confirmOrder або cancelOrder. Також через вкладку Console користувач отримує зворотний зв’язок щодо успішності виконання транзакцій, хешів і подій, що згенеровані в процесі виконання.

Таким чином, Remix IDE забезпечує повний цикл роботи зі смарт-контрактами — від написання й перевірки до деплою та тестування. Його використання в межах проєкту дозволить ефективно реалізувати децентралізований компонент системи, забезпечивши тісну інтеграцію з клієнт-серверною частиною, що була реалізована у Visual Studio Code. Такий розподіл середовищ дозволив оптимально організувати процес розробки, ізольовано працюючи з відповідними частинами архітектури проєкту.

3.2 Програмна реалізація архітектурного модулю

У процесі програмної реалізації розроблюваного продукту ключове значення має забезпечення злагодженої взаємодії між серверною та клієнтською частинами системи. Саме їхня узгоджена робота гарантує повноцінне функціонування застосунку, а також комфортну та інтуїтивно зрозумілу взаємодію користувача з інтерфейсом.

Серверна частина виконує критично важливі завдання, зокрема: обробку бізнес-логіки, управління збереженням даних, авторизацію користувачів, а також забезпечення взаємодії з зовнішніми сервісами, зокрема децентралізованими блокчейн-компонентами. У свою чергу, клієнтська частина відповідає за формування користувацького інтерфейсу, реалізацію логіки взаємодії з API та динамічне відображення даних, що надходять із сервера та смарт-контрактів.

Розробка обох частин системи здійснювалась із дотриманням принципів модульності, повторного використання коду та чіткої сегментації функціональних обов’язків. Це дозволило досягти високої масштабованості рішення, підвищити стабільність роботи застосунку та спростити процес його подальшої підтримки.

У якості технологічної основи серверної частини обрано середовище виконання Node.js разом із фреймворком NestJS, який забезпечує чітку модульну структуру, зручну інжекцію залежностей та підтримку REST/WebSocket API. Особливістю даної реалізації є інтеграція з смарт-контрактами, які виконують роль незалежного та прозорого механізму зберігання і перевірки даних у межах

блокчейн-мережі. Смарт-контракти реалізовано за допомогою мови Solidity, і їх програмний код також буде наведений та проаналізований далі.

Першим етапом реалізації серверної частини програмного продукту є ініціалізація проєкту та встановлення всіх необхідних залежностей. Для цього використано менеджер пакетів npm, який дозволяє швидко підготувати середовище розробки. Базова ініціалізація та встановлення основних бібліотек та пакетів здійснюється командами:

```
npm init -y
npn i -g @nestjs/cli
npm i @nestjs/jwt
npm i bcryp
npm i sequelize-typescript
npm i ws
```

Після налаштування середовища відбувається ініціалізація основного сервера. У наступному фрагменті коду створюється екземпляр NestJS-додатку та активується CORS (Cross-Origin Resource Sharing) для взаємодії з клієнтською частиною, яка працює на порту 3001.

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.enableCors({
    origin: 'http://localhost:3001',
    credentials: true,
  });
  await app.listen(3000);
}
bootstrap();
```

Даний код запускає сервер на порту 3001. Параметри enableCors дозволяють клієнтському застосунку здійснювати авторизовані запити до сервера, зберігаючи підтримку обміну cookie та заголовками авторизації.

Після запуску серверного середовища, наступним кроком є підключення до бази даних та ініціалізація основних модулів додатку. Для цього використовується ORM Sequelize, що дозволяє працювати з реляційними структурами даних у рамках об'єктно-орієнтованої моделі.

```
@Module({
  imports: [
```

```
    ConfigModule.forRoot({
      isGlobal: true,
    }),
    SequelizeModule.forRoot({
      dialect: 'postgres',
      host: process.env.POSTGRES_HOST,
      port: 5432,
      username: process.env.POSTGRES_USER,
      password: process.env.POSTGRES_PASSWORD,
      database: process.env.POSTGRES_DB,
      models: [User, Events],
      logging: false,
    }),
    UsersModule,
    AuthModule,
    WsModule,
    AuditsModule,
  ],
})
export class AppModule {}
```

Ініціалізація моделей, підключення до бази, а також імпорт функціональних модулів виконується в межах головного модуля — AppModule. Даний код забезпечує централізовану координацію логіки проєкту, дозволяє ефективно структурувати код та забезпечити доступ до необхідних сервісів з клієнтської частини.

Невід'ємною складовою архітектури є реалізація WebSocket-модуля, який відповідає за обмін даними в реальному часі між сервером і клієнтом. Цей модуль дозволяє миттєво передавати події, такі як дії користувачів та результати аудиту, без необхідності повторного опитування сервера. Нижче представлено реалізацію класу WsGateway, що відповідає за запуск та обробку WebSocket-з'єднань:

```
@Injectable()
export class WsGateway implements OnModuleInit {
  constructor(private auditService: Audit) {}
  private wss: WebSocket.Server;
  onModuleInit() {
    const server = createServer();
    this.wss = new WebSocket.Server({ server });
    server.listen(3002, () => {
      console.log('WS server listening on port 3002');
    });
    this.wss.on('connection', (socket: WebSocket) => {
      console.log('Client connected');
```

```
            socket.on('message', async (data: WebSocket.RawData) => {
              try {
const auditData: AuditResult = await this.auditService.saveEvent(JSON.parse(data.toString()));
                socket.send(JSON.stringify(auditData));
              } catch (error) {
                socket.send(JSON.stringify({ error: 'Invalid JSON' }));
              }
            });
            socket.on('close', () => {
              console.log('Client disconnected');
            });
          });
        }
      }
```

Код має обробник подій “message”, що спрацьовує при кожній події на клієнті та викликає функцію “saveEvent”, яка відповідає за збереження івентів та аудит системи. Детальний опис блоку аудиту наведено нижче:

```
      const query = `
        INSERT INTO auth.events ("eventName", "userId", "userIp", "eventOptions", "createdAt",
"updatedAt")
         VALUES (:eventName, :userId, :userIp, :eventOptions, NOW(), NOW())
        `;
        await this.sequelize.query(query, {
          replacements: {
            eventName: eventData.eventName,
            userId: eventData.userId,
            userIp: eventData.userIp,
            eventOptions: eventData.eventOptions ?? null,
          },
          type: QueryTypes.INSERT,
        });
```

Перша частина методу, зберігає подію, в базі даних з детальним описом: назвою, ір з якого вона надійшла, який юзер її надіслав та детальні налаштування кожної події. Після збереження, викликається функція аудиту, яка детально перевіряє кожну транзакцію на предмет підозрілої активності:

const auditData = await this.check(eventData.userId, eventData.eventName, eventData.userIp, eventData.eventOptions);

Детальний огляд методу check розглянуто нижче.

```
      const [userOrdersData, eventsData, userIpIsMatched]: [OrdersData, EventsData, boolean] =
await Promise.all([
          this.getUserOrdersData(userId),
          this.getEventsData(userId, eventName, userIp),
```

```
            this.userIpIsMatched(userId, userIp),
        ]);
```

Спочатку йде збір статистики зі збереженої інформації.

```
        private getUserOrdersData = async (userId: number): Promise<OrdersData> => {
        const res: OrdersData[] = await this.sequelize.query(
        `SELECT
        AVG("stableCoinEquivalent") AS "averageOrderVolume",
        COUNT("id") AS "ordersCounts",
        COUNT(CASE WHEN "createdAt" >= NOW() - INTERVAL '24 HOURS' THEN 1 END) AS
"ordersCountLast24h"
        FROM auth."takerOrders"
        WHERE "takerId" = :userId OR "makerId" = :userId`,
        {
        replacements: { userId },
        type: QueryTypes.SELECT,
        },
        );
        return res[0];

        };
```

Функція getUserOrdersData збирає інформацію, про середній обсяг заявки користувача за весь час, за останні 24 години та кількість проведених ним угод.

```
        private getEventsData = async (userId: number, eventName: string, userIp: string):
Promise<EventsData> => {
        try {
        const res: EventsData[] = await this.sequelize.query(
        `SELECT
        COUNT(CASE WHEN "eventName" = :eventName THEN 1 ELSE 0 END) AS
"eventProcessedGeneral",
        COUNT(CASE WHEN "eventName" = :eventName AND "userIp" = :userIp THEN 1 ELSE 0 END)
AS "eventProcessedOnThisIp"
        FROM auth.events
        WHERE "userId" = :userId`,
        {replacements: { userId, eventName, userIp }, type: QueryTypes.SELECT,},
        );
        return res[0];
        } catch (error) {
        console.log(error);
        return { eventProcessedGeneral: 100, eventProcessedOnThisIp: 0 };
        }
        };
```

Функція збирає інформацію про загальну кількість конкретних подій згенерованих користувачем, та кількість подій згенерованих з цієї ж ip-адреси.

Також йде перевірка чи збігається ip запиту з останнім ip зареєстрованого користувача, за допомогою наступної функції:

```
private userIpIsMatched = async (userId: number, userIp: string): Promise<boolean> => {
try {
const res: IpData[] = await this.sequelize.query(
`SELECT "userIp" FROM auth."users" WHERE "id" = :userId`,
{ replacements: { userId }, type: QueryTypes.SELECT, },
 );
if (res.length > 1) {
return false;
}
return res[0]?.userIp === userIp;
} catch (error) {
console.log(error);
return false;
}
};
```

Після отримання усіх потрібних даних викликається функція matchRiskRulePoints що реалізує набір формалізованих правил оцінки, відповідно до яких здійснюється нарахування балів ризику користувачу. Кожне правило відповідає окремому критерію оцінки поведінки користувача, і при його спрацюванні до загального результату додається певна кількість балів. Таким чином формується сукупна оцінка ризиковості дій, що дозволяє системі приймати рішення щодо допуску або блокування операції. Нижче наведено приклад реалізації цього методу, який демонструє логіку обробки ключових ризикових індикаторів:

```
private matchRiskRulePoints = (
    userOrdersData: OrdersData,
    eventsData: EventsData,
    userIpIsMatched: boolean,
    userId: number,
    options?: { amount?: number },
  ): number => {
    let points = 0;
    const logs: string[] = [];
    const volumeLow = 50;
    const volumeHigh = 1000;
    const volumeLowPoints = 30;
    const volumeNormalPoints = 10;
    const volumeHighPoints = 20;
    const transactionMultiplierLimit = 2;
```

```
const transactionAnomalyPoints = 50;
const highOrdersCount = 100;
const highOrdersPoints = 10;
const lowOrdersPoints = 15;
const active24hLimit = 30;
const activityPoints = 30;
const unmatchedIpPoints = 20;
const unknownApiPoints = 25;
```

На початковому етапі в межах функції оголошуються константи, які визначають вагові коефіцієнти та порогові значення для кожного з критеріїв ризику. Після цього послідовно виконується оцінка відповідних умов і здійснюється нарахування балів відповідно до того, наскільки дії користувача відповідають визначеним ризиковим факторам.

```
if (userOrdersData.averageOrderVolume > volumeHigh) {
points += volumeHighPoints;
logs.push(`Високий середній обсяг: +${volumeHighPoints}`);
} else if (userOrdersData.averageOrderVolume < volumeLow) {
points += volumeLowPoints;
logs.push(`Низький середній обсяг: +${volumeLowPoints}`);
} else {
points += volumeNormalPoints;
logs.push(`Нормальний обсяг: +${volumeNormalPoints}`);
}
if (options?.amount && userOrdersData.averageOrderVolume) {
if (options.amount > userOrdersData.averageOrderVolume * transactionMultiplierLimit) {
points += transactionAnomalyPoints;
logs.push(`Аномально велика сума: +${transactionAnomalyPoints}`);
}
}
if (userOrdersData.ordersCounts > highOrdersCount) {
points += highOrdersPoints;
logs.push(`Велика кількість ордерів: +${highOrdersPoints}`);
} else {
points += lowOrdersPoints;
logs.push(`Мала кількість ордерів: +${lowOrdersPoints}`);
}
if (userOrdersData.ordersCountLast24h > active24hLimit) {
points += activityPoints;
logs.push(`Активність за 24 год: +${activityPoints}`);
}
if (!userIpIsMatched) {
points += unmatchedIpPoints;
logs.push(`IP не співпадає: +${unmatchedIpPoints}`);
}
if (eventsData.eventProcessedOnThisIp === 0) {
```

```
points += unknownApiPoints;
logs.push(`Подія не виконувалась з цієї IP раніше: +${unknownApiPoints}`);
}
if (userOrdersData.ordersCountLast24h > (userOrdersData.ordersCounts / 7) * 3) {
points += 20;
logs.push('Різке зростання активності за добу: +20');
}
if (eventsData.eventProcessedGeneral > 100) {
points += 20;
logs.push('Надмірне повторення події: +20');
}
console.log(`Аудит користувача: ${userId} total points = ${points}`);
for (const log of logs) {
console.log(` - ${log}`);
}
return points;
};
```

Після завершення підрахунку балів ризику відбувається перевірка отриманого значення на відповідність заздалегідь визначеному порогу. У разі, якщо сума балів перевищує встановлене граничне значення, вважається, що користувач продемонстрував потенційно підозрілу або аномальну активність:

```
if (points >= 70) {
   return { isAllowed: false, message: `User exceeded risk threshold with ${points} points` };
}
return { isAllowed: true };
```

У випадку, якщо система класифікує поведінку як ризиковану, виконується додаткове логування інциденту, що дозволяє відстежувати подібні ситуації для подальшого аналізу. Крім того, автоматично виконується блокування користувача шляхом оновлення його статусу в базі даних до AUDIT_BLOCKED:

```
if (!auditData?.isAllowed) {
console.info(`USER WITH ID ${eventData.userId} HAS FAILED AUDIT. REASON:
${auditData.message}`);
await this.sequelize.query(`UPDATE auth."users" SET "userStatus" = 'AUDIT_BLOCKED' WHERE
"id" = :userId`,
{
  replacements: {
  userId: eventData.userId,
},
  type: QueryTypes.UPDATE,
},);}
```

Такий підхід дозволяє системі не лише автоматично реагувати на потенційні загрози, а й впроваджувати превентивні заходи безпеки у режимі реального часу, зберігаючи цілісність платформи та захищаючи її від зловживань.

Разом із реалізацією серверної логіки та механізмів аудиту критично важливою складовою системи є смарт-контракт, що виконує роль децентралізованого посередника для обміну токенами між користувачами. Саме смарт-контракт забезпечує виконання ключових операцій платформи без необхідності довіри до централізованої сторони, гарантуючи прозорість, незмінність та безпечність кожної транзакції. Нижче наведено приклад реалізації одного з основних функціональних модулів смарт-контракту — створення ордера мейкера, тобто користувача, який ініціює торгівельну пропозицію:

```
function createMakerOrder(
address tokenGive,
    address tokenReceive,
    uint256 amountGive,
    uint256 amountReceive
  ) external {
    require(amountGive > 0 && amountReceive > 0, "Invalid amounts");
    require(
      IERC20(tokenGive).transferFrom(msg.sender, address(this), amountGive),
      "Token transfer failed"
    );
    orders[nextOrderId] = MakerOrder({
      maker: msg.sender,
      tokenGive: tokenGive,
      tokenReceive: tokenReceive,
      amountGive: amountGive,
      amountReceive: amountReceive,
      active: true
    });
    emit MakerOrderCreated(nextOrderId, msg.sender);
    nextOrderId++;
  }
```

На етапі створення ордера перевіряється коректність вхідних даних, здійснюється передача токенів на адресу контракту, зберігається новий запис у структурі orders, після чого надсилається подія про успішну публікацію оголошення.

Наступним етапом є взаємодія з опублікованим ордером — з боку тейкера. Він погоджується на умови мейкера та ініціює обмін. Суть цього процесу полягає в атомарному виконанні обміну токенами між сторонами.

```
function takeOrder(uint256 orderId) external {
    MakerOrder storage order = orders[orderId];
    require(order.active, "Order inactive");
    order.active = false;
            require( IERC20(order.tokenReceive).transferFrom(msg.sender, address(this), order.amountReceive),
      "Taker token transfer failed"
    );
    IERC20(order.tokenGive).transfer(msg.sender, order.amountGive);
    IERC20(order.tokenReceive).transfer(order.maker, order.amountReceive);
    emit OrderTaken(orderId, msg.sender);
}
```

Під час виконання перевіряється статус ордера, блокується його повторне використання, здійснюються перекази обох сторін, а також фіксується факт завершення операції через подію OrderTaken. Таким чином, взаємодія між мейкером і тейкером реалізована безпечно та без участі посередників, що відповідає основним принципам Web3-платформ.

Далі розглянемо розробку користувацького інтерфейсу. Клієнтська частина програмного продукту відіграє ключову роль у забезпеченні взаємодії кінцевого користувача із системою. Вона реалізує інтерфейс, через який здійснюється доступ до функціональності платформи, зокрема створення та перегляд ордерів, підключення криптогаманця, взаємодія зі смарт-контрактами та виконання транзакцій.

На початковому етапі розробки клієнтської частини було здійснено ініціалізацію проєкту з використанням інструменту Create React App, який забезпечує базову структуру застосунку та інтегрує всі необхідні для розробки модулі. Команди ініціалізації мали наступний вигляд:

```
npx create-react-app my-app
npm i axios
npm i ethers
```

Розробка клієнтської частини здійснювалась за компонентно-орієнтованим підходом, де кожен функціональний елемент інтерфейсу було виділено в окремий

React-компонент. Такий підхід забезпечує модульність, повторне використання коду та спрощення тестування окремих частин застосунку. Серед основних клієнтських модулів, реалізованих у межах проєкту, варто виділити такі: компонент підключення гаманця, компонент створення ордерів, компонент відображення активних ордерів. Розглянемо їх реалізацію.

Компонент підключення гаманця відповідає за ініціалізацію з'єднання з Ethereum-провайдером через Web3-браузер або розширення MetaMask, зчитування адреси користувача та збереження її у стані застосунку:

```
const connectWallet = async () => {
 if (!window.ethereum) return alert('MetaMask не встановлено');
 await switchToSepolia();
 await window.ethereum.request({ method: 'eth_requestAccounts' });
 const signer = await provider.getSigner();
 const address = await signer.getAddress();
 const contract = new ethers.Contract(CONTRACT_ADDRESS, contractABI, signer);
 setSigner(signer);
 setAccount(address);
 setContract(contract);
};
```

Кнопка для виклику функції підключення виглядає наступним чином:

```
<button
   onClick={connectWallet}
     style={{
      padding: '0.5rem 1rem',
      backgroundColor: '#3b82f6',
      color: 'white',
      border: 'none',
      borderRadius: '6px',
      cursor: 'pointer'
     }}>
     Connect Wallet
</button>
```

Компонент створення ордерів реалізує взаємодію з користувачем для створення ордера обміну. Після введення параметрів токенів та сум, відбувається підпис транзакції через MetaMask, передача даних до смарт-контракту, а також відправка повідомлення через WebSocket:

```
const handleCreate = async () => {
  try {
    const tokenGiveAddress = tokenMap[tokenGive.toUpperCase()];
    const tokenReceiveAddress = tokenMap[tokenReceive.toUpperCase()];
```

```
        if (!tokenGiveAddress || !tokenReceiveAddress) {
        return alert('Invalid token symbol. Use: USDC, USDT, DAI, WETH');
       }
        if (!signer || !contract) return alert('Wallet not connected');
       const tokenContract = new ethers.Contract(tokenGiveAddress, [
        'function approve(address spender, uint256 amount) public returns (bool)'
       ], signer);
       await tokenContract.approve(CONTRACT_ADDRESS, ethers.parseUnits(amountGive, 18));
       const tx = await contract.createMakerOrder(
        tokenGiveAddress,
        tokenReceiveAddress,
        ethers.parseUnits(amountGive, 18),
        ethers.parseUnits(amountReceive, 18)
       );
       await tx.wait();
       socket?.send(JSON.stringify({
        type: 'TAKE_ORDER',
        txHash: tx.hash,
        sender: account,
        payload: { tokenGive, tokenReceive, amountGive, amountReceive }
       }));
       alert('Maker Order created');
      } catch (error) {
       alert(error?.message);
      }
     };
```

Компонент відображення активних ордерів відповідає за отримання й візуалізацію активних ордерів користувача. Він дозволяє відобразити лише ті ордери, які створив поточний користувач:

```
    const [maker, tokenGive, tokenReceive, amountGive, amountReceive, active] = await
contract.orders(i);
        if (maker.toLowerCase() === account.toLowerCase() && active) {
       <div key={order.id} style={{
        padding: '1.5rem',
        backgroundColor: '#1e293b',
        borderRadius: '0.75rem',
        marginBottom: '1.5rem',
        color: '#f1f5f9',
        display: 'flex',
        flexDirection: 'column',
        gap: '0.75rem',
    }}>
      <div style={{ fontSize: '1rem' }}>Order #{order.id}</div>
      <div>Give: {ethers.formatUnits(order.amountGive, 18)}
{tokenSymbols[order.tokenGive]}</div>
      <div>Receive: {ethers.formatUnits(order.amountReceive, 18)}
{tokenSymbols[order.tokenReceive] }</div>
```

```
<div>Status: {order.active ? 'Active' : 'Inactive'}</div>
```

З метою своєчасного реагування на потенційно аномальні дії користувачів, було реалізовано механізм обміну даними між клієнтською частиною застосунку та сервером із використанням протоколу WebSocket:

```
try {
  socketRef.current = new WebSocket('ws://localhost:3002/ws');
  socketRef.current.onopen = () => console.log('WebSocket connected');
  socketRef.current.onclose = () => console.log('WebSocket disconnected');
  socketRef.current.onerror = (e) => console.error('WebSocket error', e);
} catch (error) {
  console.error('WebSocket init error:', error);
}
return () => {
  if (socketRef.current) {
    socketRef.current.close();
    console.log('WebSocket closed');
  }
};
```

Ініціалізація з’єднання здійснюється шляхом створення екземпляру об'єкта WebSocket з передачею адреси сервера у вигляді параметра ws://localhost:3002/ws. З’єднання зберігається у змінній socketRef.current, що дозволяє використовувати його в межах життєвого циклу React-компонента.

3.3 Тестування програмного модулю

Тестування є важливим етапом життєвого циклу програмного забезпечення, що забезпечує перевірку працездатності, надійності та відповідності функціональності розробленого модуля заданим вимогам. У межах роботи буде проведено ручне тестування, яке дозволяє оцінити коректність програмного модуля в різних сценаріях використання.

Ручне тестування передбачає послідовне виконання типових та граничних сценаріїв взаємодії з системою з боку користувача з метою виявлення помилок у логіці роботи, обробці вхідних даних, а також перевірки відгуку інтерфейсу на різні дії. Особлива увага приділятиметься перевірці функцій створення, редагування та видалення даних, авторизації а також їхньої відповідності збереженим у системі результатам.

На початковому етапі тестування було перевірено функціональність авторизації, яка забезпечує реєстрацію нових та вхід зареєстрованих користувачів.

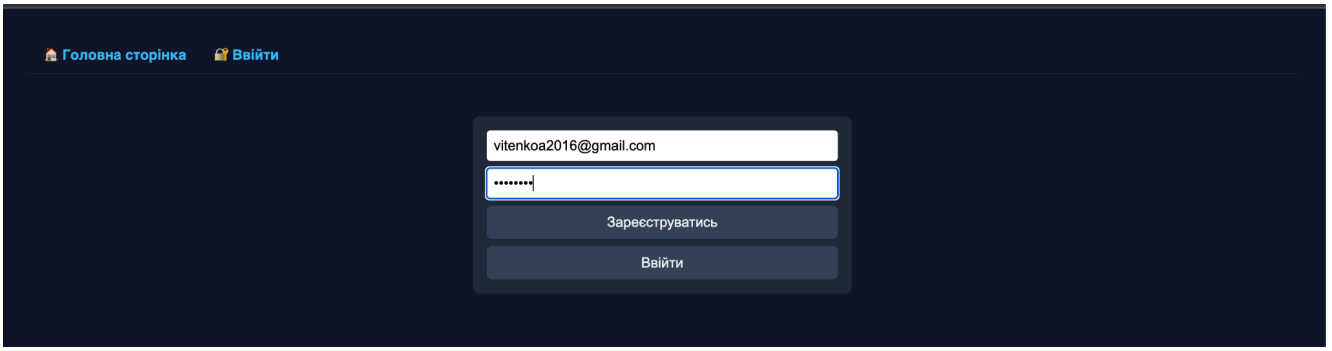


Рисунок 3.4 - Форма авторизації

На рисунку 3.4 показано процес введення адреси електронної пошти та пароля у відповідні поля, з можливістю реєстрації чи входу в систему. Зареєструємо нового користувача (рис 3.5).

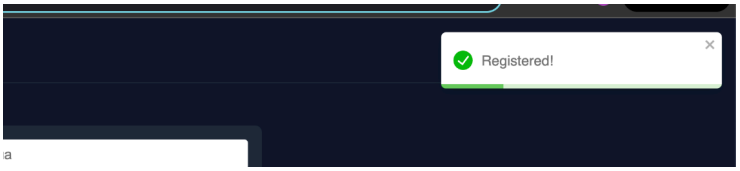


Рисунок 3.5 - Сповіщення про успішну реєстрацію

Після авторизації користувач отримує доступ до головної сторінки, де отримує повідомлення про те, що потрібно під’єднати гаманець для початку роботи з платформою (рис 3.6).

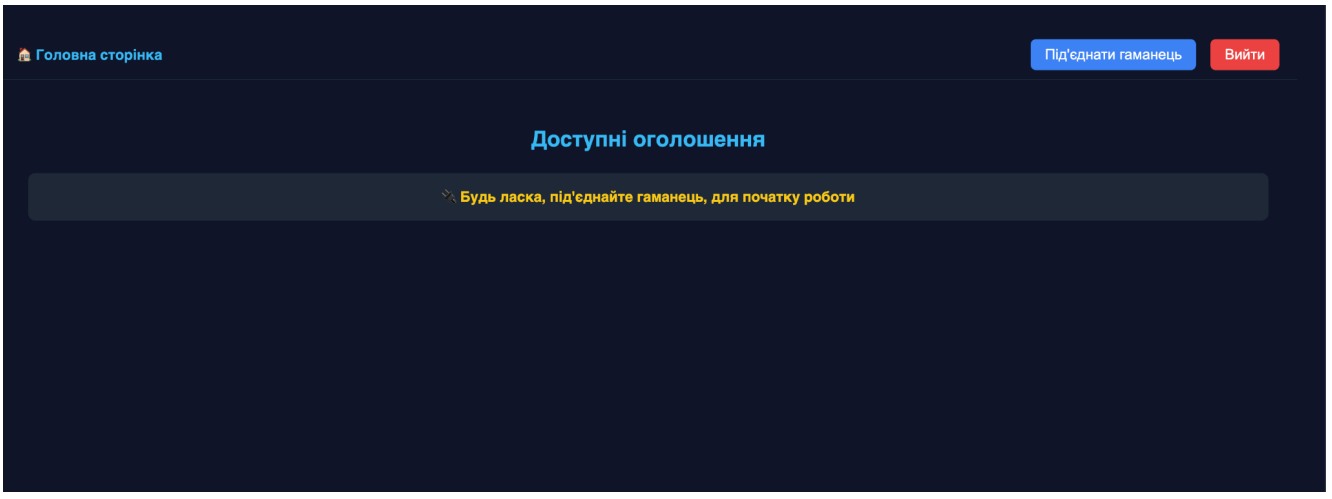


Рисунок 3.6 - Головний екран без під’єднаного гаманця

Щоб отримати доступ до функціональності платформи, потрібно натиснути кнопку під’єднати гаманець (рис 3.7-3.8).

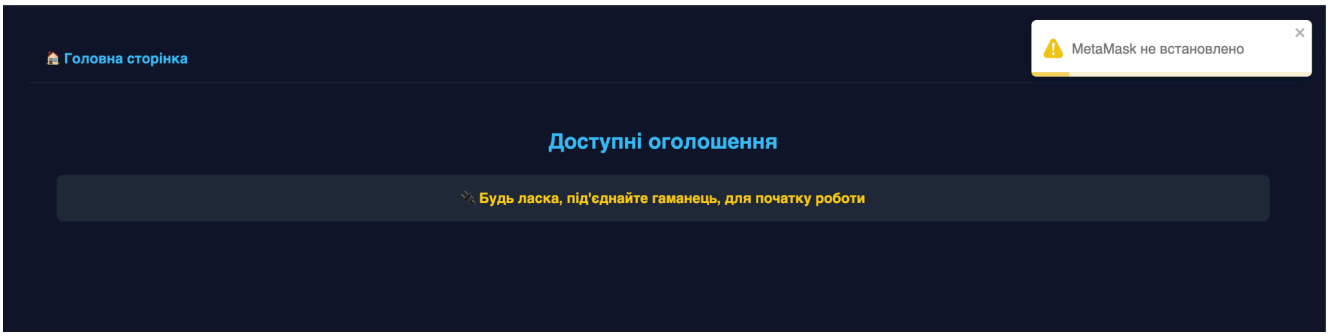


Рисунок 3.7 - Повідомлення про відсутність розширення MetaMask

Якщо у користувача відсутнє web розширення крипто гаманця, доступ буде заборонено, та з’явиться повідомлення про помилку (рис 3.7). Після встановлення розширення, гаманець буде успішно під’єднано (рис 3.8).

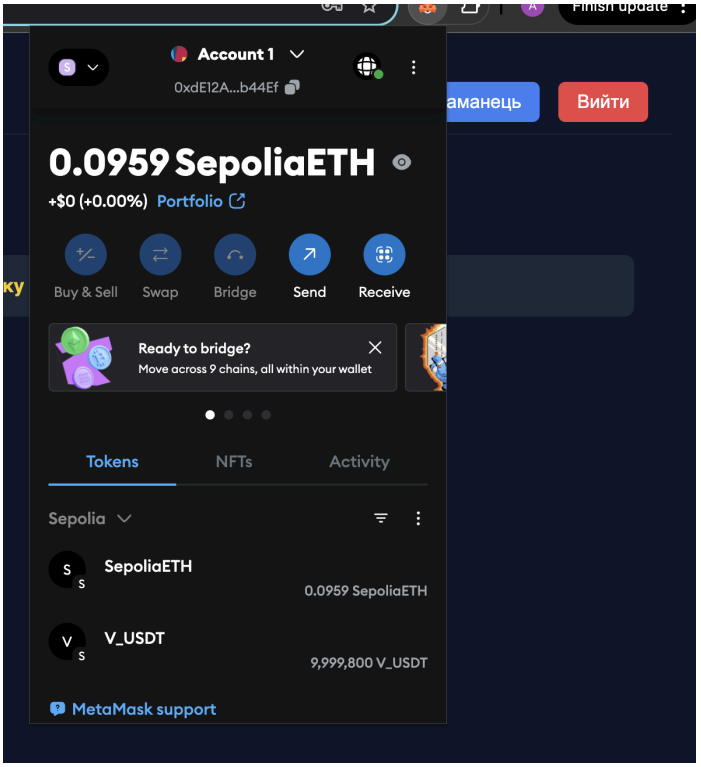


Рисунок 3.8 - Успішне приєднання гаманця до платформи

При успішному приєднанні гаманця на головній сторінці з’являється список доступних оголошень та основна функціональність платформи, зокрема можливість p2p обмінів (рис. 3.9).

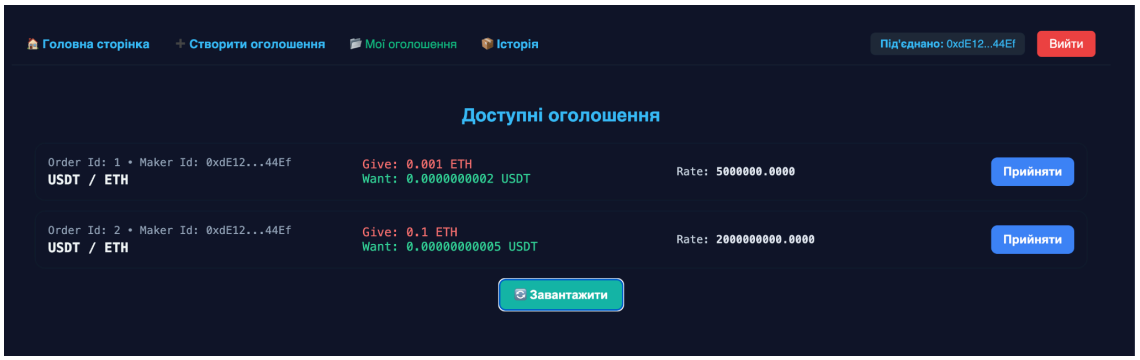


Рисунок 3.9 - Головна сторінка

З метою перевірки працездатності механізму виявлення аномалій було змодельовано ситуацію, що імітує підозрілу активність користувача. Зокрема, проведено послідовне виконання чотирьох угод із тим самим мейкером на однакові суми впродовж мінімального проміжку часу. Така поведінка потенційно може свідчити про автоматизовану активність або навмисні спроби маніпуляції, і саме подібні патерни система має виявляти як аномальні.

Для реалізації тесту було натиснуто кнопку «Прийняти» для кожного оголошення, після чого система очікувала підтвердження завершення відповідних транзакцій у блокчейні. Усі дії виконувались у максимально стислі терміни для досягнення умов, за яких механізм аналізу подій має зафіксувати нетипову активність. На Рисунку 3.10 представлено один із етапів цього процесу — момент підтвердження транзакції після прийняття ордера.

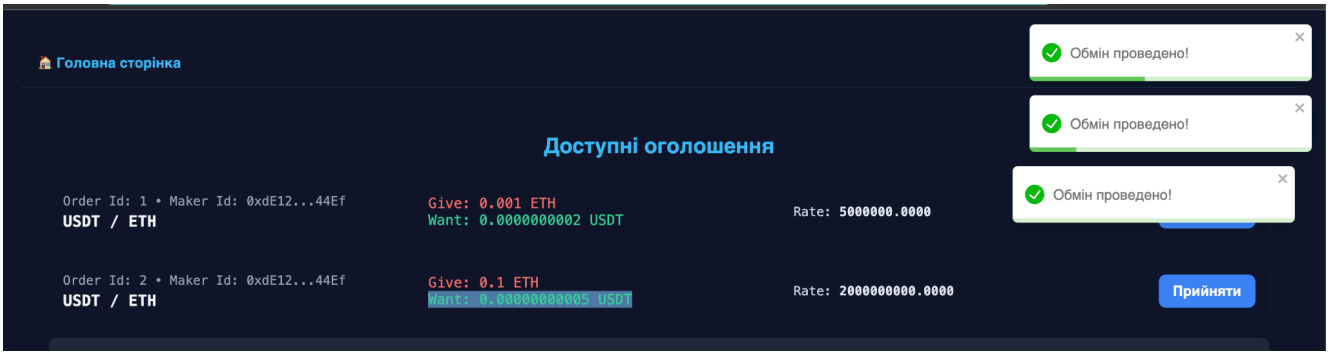


Рисунок 3.10 - Проведення успішних угод

Під час спроби, провести четверту під ряд угоду система блокує аккаунт, та виводить повідомлення про блокування (рис 3.11).

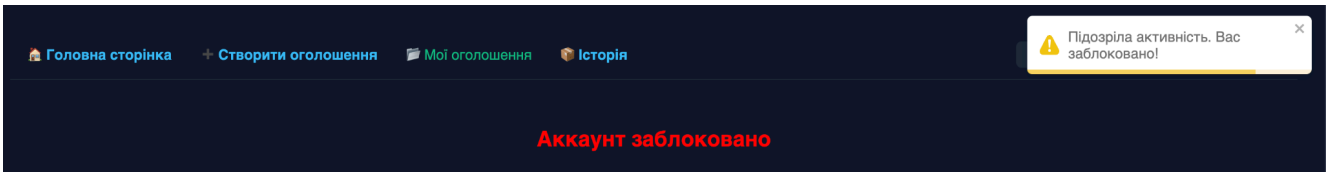


Рисунок 3.11 - Заблокований акаунт

Даний сценарій демонструє реалізацію внутрішньої логіки перевірки користувацького статусу перед виконанням фінансово-критичних операцій. При виявленні аномальної активності або системних обмежень. Платформа автоматично забороняє створення нових ордерів (рис 3.12). Відповідне повідомлення про помилку слугує механізмом інформування користувача про накладені обмеження та необхідність проведення додаткової перевірки та перегляду його дій.

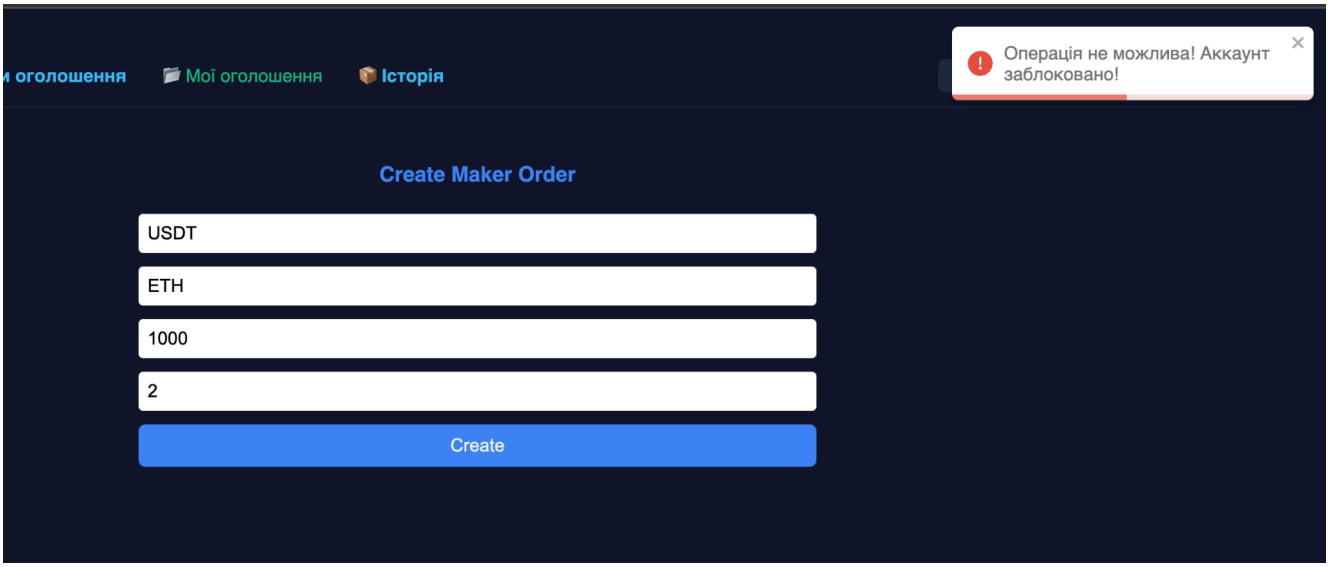


Рисунок 3.12 - Повідомлення про помилку при спробі створення оголошення

Повідомлення виконує роль інструмента зворотного зв’язку, що дозволяє користувачу зрозуміти причину відмови в доступі до функціоналу та вказує на необхідність проходження додаткової перевірки або перегляду своїх нещодавніх дій.

В разі активації обмежень, що накладаються на акаунт користувача внаслідок виявлення аномальної активності або порушення політик платформи, функціональність системи для такого користувача суттєво звужується. Єдиною дозволеною дією в такому режимі залишається перегляд історії транзакцій, що дозволяє користувачеві зберігати доступ до інформації про власну активність на

платформі. На Рисунку 3.13 представлено інтерфейс перегляду історії транзакцій для акаунта з активованими обмеженнями. Такий підхід забезпечує прозорість у діях платформи, дозволяє користувачу ознайомитися з попередніми операціями та, за потреби, звернутися до служби підтримки або ініціювати процес розблокування.

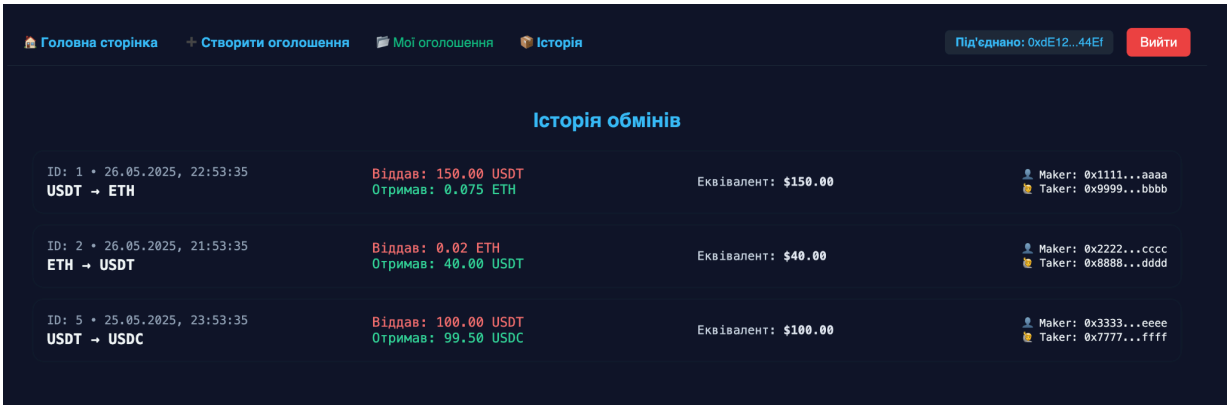


Рисунок 3.13 – Інтерфейс перегляду історії транзакцій

Отже, межах розділу 3.3 було проведено ручне тестування основних функціональних компонентів розробленої децентралізованої P2P-платформи обміну криптовалют. Тестування здійснювалося з метою перевірки коректності реалізації логіки взаємодії користувача із системою, стабільності роботи інтерфейсу, обробки вхідних даних та відповідності результатів очікуваній поведінці. Особливу увагу було приділено перевірці авторизації, створення, ордерів, а також роботі смарт-контракту та WebSocket-з’єднання. Крім того, протестовано систему виявлення аномальної активності в реальному часі: було змодельовано сценарії інтенсивної взаємодії з платформою, що дозволило перевірити механізми блокування дій користувача та генерацію системних попереджень.

Результати тестування підтвердили стабільність роботи застосунку та ефективність реалізованої логіки. Система демонструє належну реакцію на стандартні та граничні сценарії, своєчасно інформує користувача про помилки або обмеження, а також гарантує узгоджену взаємодію між клієнтською, серверною та блокчейн-частинами. Таким чином, проведене тестування засвідчило готовність програмного модуля до подальшого використання в реальному середовищі,

підтвердило якість реалізації ключових функцій і відповідність проєктного рішення заданим технічним і функціональним вимогам.

3.4 Висновки до розділу 3

Отже, в даному розділі було здійснено комплексну реалізацію децентралізованої програмної платформи для P2P-обміну криптовалютами з урахуванням вимог до безпеки, стабільності та функціональної повноти. Розгляд охопив вибір і обґрунтування інструментів розробки, та програмну реалізацію ключових модулів, а також проведення ручного тестування функціональності.

Перш за все, обґрунтовано використання двох середовищ розробки. Visual Studio Code було обрано як основне IDE для створення клієнтської та серверної частин системи завдяки його підтримці JavaScript/TypeScript, численним розширенням і зручності інтеграції зі стеком технологій. Remix IDE використовувалося для створення та налагодження смарт-контрактів на мові Solidity, що дозволило забезпечити коректну взаємодію з блокчейн-мережею та підтримку децентралізованих транзакцій.

У процесі програмної реалізації було побудовано узгоджену клієнт-серверну архітектуру. Серверна частина забезпечує авторизацію користувачів, обробку бізнес-логіки, запис і читання даних із бази даних, взаємодію зі смарт-контрактом та обробку подій у режимі реального часу за допомогою WebSocket-з’єднання. Клієнтська частина, реалізована за допомогою React, відповідає за побудову динамічного інтерфейсу користувача, надсилання запитів до API та візуалізацію стану ордерів та транзакцій. Особливу увагу приділено реалізації механізмів анонімної авторизації, створення та обробки ордерів на обмін, підтвердження угод зі сторони тейкера, виявлення аномальних дій користувачів на рівні сервера та автоматичного формування відповідних попереджень.

На етапі тестування було проведено ручну перевірку ключових функцій програмного модуля в умовах реальних сценаріїв використання. Тестування включало авторизацію, створення, редагування та видалення ордерів, а також перевірку обробки подій у режимі реального часу. Усі компоненти системи

пройшли тестування успішно, без виявлення критичних помилок, що свідчить про їхню готовність до використання в робочому середовищі.

У підсумку, в рамках третього розділу було успішно реалізовано цілісний програмний модуль децентралізованої P2P-платформи, що поєднує переваги блокчейн-технологій, захищеної авторизації та системи моніторингу підозрілих дій у реальному часі. Розроблена архітектура демонструє високу гнучкість, стабільність та потенціал до масштабування, що дозволяє розглядати дану систему як основу для подальшого розвитку більш складних та функціонально багатих рішень у сфері DeFi.

ВИСНОВОК

У результаті проведеного дослідження та розробки децентралізованої P2P-платформи обміну криптовалют з інтеграцією механізмів виявлення аномалій та системи попереджень у режимі реального часу було досягнуто важливих технічних і прикладних результатів, що підтверджують ефективність запропонованого підходу.

Актуальність цієї роботи полягає у зростаючій потребі в безпечних, надійних та прозорих механізмах обміну цифровими активами в умовах зростаючих кіберзагроз та шахрайських схем. Розроблена система дозволяє здійснювати обмін без участі централізованих посередників, підтримує миттєву передачу подій, а також автоматично виявляє підозрілі дії користувачів, що є критично важливим для забезпечення довіри та безпеки в децентралізованих середовищах.

Основна новизна роботи полягає у поєднанні смарт-контрактного обміну криптовалютами з механізмом виявлення аномалій та системою формування попереджень у реальному часі, реалізованою на основі WebSocket-з’єднання. Такий підхід дозволяє не лише забезпечити децентралізованість і захист даних, але й активно реагувати на ризики, що виникають у процесі взаємодії між користувачами.

Отримані результати мають практичну цінність, оскільки розроблений програмний модуль може бути застосований у різних децентралізованих сервісах і фінансових платформах, де важливою є безпека авторизації, збереження даних та захист від потенційних зловживань. Його використання можливе як у проєктах, пов’язаних з криптовалютами, так і в інших сферах, де необхідно забезпечити прозорий та безпечний обмін інформацією між користувачами.

У підсумку, розробка децентралізованої P2P-платформи обміну криптовалют з системою виявлення аномалій і попереджень у реальному часі є вагомим внеском у підвищення рівня безпеки, довіри та ефективності в галузі цифрових обмінів і децентралізованих фінансових рішень.

ПЕРЕЛІК ПОСИЛАНЬ

1. Susnjara S., Smalley I. What Is Blockchain? | IBM. IBM - United States. URL: https://www.ibm.com/think/topics/blockchain (дата звернення: 01.03.2025).

2. Tikhomirov, Sergei. "Ethereum: state of knowledge and research perspectives." International symposium on foundations and practice of security. Cham: Springer International Publishing, 2017.

3. Proof of Work (PoW) vs. Proof of Stake (PoS): what's the difference?. Coinbase. URL: https://www.coinbase.com/en-gb/learn/crypto-basics/proof-of-work-pow-vs-proof-of-stake-pos-what-is-the-difference (дата звернення: 01.03.2025).

4. Saleh, Fahad. "Blockchain without waste: Proof-of-stake." The Review of financial studies 34.3 (2021): 1156-1190.

5. Nguyen, Cong T., et al. "Proof-of-stake consensus mechanisms for future blockchain networks: fundamentals, applications and opportunities." IEEE access 7 (2019): 85727-85745.

6. Hirai, Yoichi. "Defining the ethereum virtual machine for interactive theorem provers." Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21. Springer International Publishing, 2017 (дата звернення: 03.03.2025).

7. Wohrer, Maximilian, and Uwe Zdun. "Smart contracts: security patterns in the ethereum ecosystem and solidity." 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE). IEEE, 2018. (дата звернення: 03.03.2025).

8. Ghelani, Diptiben. "What is Non-fungible token (NFT)? A short discussion about NFT Terms used in NFT." Authorea Preprints (2022) (дата звернення: 03.03.2025).

9. What is a DEX?. Coinbase. URL: https://www.coinbase.com/en-gb/learn/crypto-basics/what-is-a-dex (дата звернення: 03.03.2025).

10. Harvey, Campbell R., Joel Hasbrouck, and Fahad Saleh. "The evolution of decentralized exchange: Risks, benefits, and oversight." Benefits, and Oversight (June 15, 2024) (2024) (дата звернення: 03.03.2025)..

11. Asef, Mohammad Ali, and Seyed Mojtaba Hosseini Bamakan. "From x* y= k to Uniswap Hooks: A Comparative Review of Decentralized Exchanges (DEX)." arXiv preprint arXiv:2410.10162 (2024). (дата звернення: 01.04.2025)

12. What Is an Oracle in Blockchain? Â» Explained | Chainlink. Chainlink: The backbone of blockchain. URL: https://chain.link/education/blockchain-oracles (дата звернення: 01.04.2025).

13. Boonpeam, Naratorn, Warodom Werapun, and Tanakorn Karode. "The arbitrage system on decentralized exchanges." 2021 18th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON). IEEE, 2021 (дата звернення: 08.05.2025).

14. Rees-Evans, Alexander. "DEXs and CEXs." How to Launch a Token: An Essential Guide for the Uninitiated. Berkeley, CA: Apress, 2024. 201-229 (дата звернення: 08.05.2025).

15. Gan, Hing Long. Centralized crypto exchanges (CEX) website. Diss. UTAR, 2023(дата звернення: 08.05.2025).

16. Centralized Exchange (CEX): What It Is, Pros & Cons, and How to Choose the Right Platform. ECOS. URL: https://ecos.am/en/blog/centralized-exchange-cex-what-it-is-pros-cons-and-how-to-choose-the-right-platform/?srsltid=AfmBOoqBh8lSxbsOkPMoVKdNSdUS-1GNYCwq1CTpWavi1MAS14oaIQZV (дата звернення: 08.05.2025).

17. Lo, Yuen C., and Francesca Medda. "Uniswap and the Emergence of the Decentralized Exchange." Journal of financial market infrastructures 10.2 (2021): 1-25 (дата звернення: 08.05.2025).

18. Lehar, Alfred, and Christine A. Parlour. "Decentralized exchange: The uniswap automated market maker." Journal of Finance forthcoming (2021) (дата звернення: 08.05.2025).

19. How Uniswap works | Uniswap. Uniswap Docs | Uniswap. URL: https://docs.uniswap.org/contracts/v2/concepts/protocol-overview/how-uniswap-works (дата звернення: 08.05.2025).

20. Lehar, Alfred, and Christine Parlour. "Decentralized exchange: The uniswap automated market maker." The Journal of Finance 80.1 (2025): 321-374 (дата звернення: 08.05.2025).

21. Top Decentralized Exchanges Ranked by 24H Trading Volume. coingecko. URL: https://www.coingecko.com/en/exchanges/decentralized (дата звернення: 08.05.2025).

22. Lehar, Alfred, and Christine Parlour. "Decentralized exchange: The uniswap automated market maker." The Journal of Finance 80.1 (2025): 321-374 (дата звернення: 20.05.2025).

23. What is PancakeSwap (CAKE)?. The Motley Fool. URL: https://www.fool.com/terms/p/pancakeswap/ (дата звернення: 09.05.2025).

24. Documentation | NestJS - A progressive Node.js framework. Documentation | NestJS - A progressive Node.js framework. URL: https://docs.nestjs.com/ (дата звернення: 20.05.2025).

25. The starting point for learning TypeScript. TypeScript: JavaScript With Syntax For Types. URL: https://www.typescriptlang.org/docs/ (date of access: 20.05.2025).

26. PostgreSQL: documentation. PostgreSQL: The world's most advanced open source database. URL: https://www.postgresql.org/docs/ (дата звернення: 20.05.2025).

27. Redis Docs. Redis Docs. URL: https://redis.io/docs/latest/ (дата звернення: 21.05.2025).

28. Solidity – Solidity 0.8.30 documentation. Solidity – Solidity 0.8.30 documentation. URL: https://docs.soliditylang.org/en/v0.8.30/ (дата звернення: 21.05.2025).

29. Rahimian, Reza, and Jeremy Clark. "TokenHook: Secure ERC-20 smart contract." arXiv preprint arXiv:2107.02997 (2021) (дата звернення: 21.05.2025).

30. Microsoft. Visual Studio Code - Code Editing. Redefined. Visual Studio Code - Code Editing. Redefined. URL: https://code.visualstudio.com/ (дата звернення: 21.05.2025).

ДОДАТКИ

75

Додаток А. Технічне завдання

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ
Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор
Юрій ЯРЕМЧУК
" 20 " березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до бакалаврської кваліфікаційної роботи на тему:

Розробка децентралізованої P2P платформи обміну криптовалют з інтеграцією механізмів виявлення аномалій та системи попереджень про підозрілі дії в реальному часі на основі WebSocket

08-72.БКР.005.00.110.ТЗ

Керівник бакалаврської кваліфікаційної роботи
к.т.н., доцент каф. МБІС
Грицак А. В.
"_____" _____

Вінниця – 2025 р.

1. Найменування та область застосування

Програмний засіб децентралізованої P2P платформи обміну криптовалют з інтеграцією механізмів виявлення аномалій та системи попереджень про підозрілі дії в реальному часі на основі WebSocket. Область застосування: забезпечення безпечного, прозорого та ефективного обміну крипто активів між користувачами без участі централізованого посередника.

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ № 97 від 20.03.2025 р.

3. Мета та призначення розробки

3.1 Мета розробки: створення децентралізованої P2P платформи обміну криптовалют із вбудованими механізмами виявлення аномальної активності та сповіщенням у реальному часі.

3.2 Призначення: забезпечити безпечний обмін криптовалют між користувачами платформи шляхом реалізації смарт-контрактів та моніторингу потенційно шкідливої активності через WebSocket-з’єднання.

4. Джерела розробки

4.1. Hirai, Yoichi. "Defining the ethereum virtual machine for interactive theorem provers." Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21. Springer International Publishing, 2017.

4.2. Lehar, Alfred, and Christine Parlour. "Decentralized exchange: The uniswap automated market maker." The Journal of Finance 80.1 (2025): 321-374

4.3. Ghelani, Diptiben. "What is Non-fungible token (NFT)? A short discussion about NFT Terms used in NFT." Authorea Preprints (2022)

4.4. Lo, Yuen C., and Francesca Medda. "Uniswap and the Emergence of the Decentralized Exchange." Journal of financial market infrastructures 10.2 (2021): 1-25

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Створення, редагування та видалення ордерів мейкером через смарт-контракт;;

5.1.2 Виявлення аномалій у діях користувачів на платформі;;

5.1.3 Повідомлення про підозрілу активність у реальному часі через WebSocket.

5.2 Вимоги до надійності:

5.2.1 Безперебійна робота смарт-контракту на тестовій мережі Ethereum;

5.2.2 Коректна обробка помилок і повідомлень;

5.2.3 Надійність WebSocket-з'єднань при високому навантаженні.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Intel Pentium 1.5 ГГц або аналог;

– оперативна пам’ять – не менше 512 Mb;

– середовище функціонування – операційна система Windows/Linux/macOS;

– вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп’ютерною технікою.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.4

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист розроблюваного програмного засобу від несанкціонованого використання.

7.2 Неможливість отримання доступу незареєстрованих користувачів до інформаційних ресурсів.

8. Техніко-економічні показники

8.1 Цінність результатів використання даного проекту повинна перевищувати витрати на його реалізацію.

8.2 Має бути реалізований таким чином, щоб підходити для використання широкого загалу.

9. Стадії та етапи розробки

78

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Початок	Закінчення
1.	Визначення напрямку бакалаврської роботи, формулювання теми		
2.	Аналіз предметної області обраної теми		
3.	Розробка алгоритму роботи		
4.	Написання бакалаврської роботи на основі розробленої теми		
5.	Передзахист бакалаврської кваліфікаційної роботи		
6.	Виправлення, уточнення, корегування бакалаврської дипломної роботи		
7.	Захист бакалаврської кваліфікаційної роботи		

10. Порядок контролю та прийому

10.1 До приймання бакалаврської кваліфікаційної роботи надається:

– ПЗ до бакалаврської кваліфікаційної роботи;

- демонстрація результату бакалаврської кваліфікаційної роботи;
- презентація;
- відзив керівника роботи;
- відзив рецензента

Технічне завдання до виконання прийняв ___Вітенко А. Ю.___

Додаток Б. Лістинг коду

```
import React, { useState, useEffect, useRef } from 'react';
import { BrowserRouter as Router, Routes, Route, useNavigate, Link, Navigate } from 'react-router-dom';
import { ethers } from 'ethers';
import axios from 'axios';
import contractABI from './contracts/P2PExchangeEscrow.json';
import './App.css';
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
const CONTRACT_ADDRESS = '0x808B81cB377798cA9800B8Db607220aa85De426F';
const SEPOLIA_CHAIN_ID = '0xaa36a7'; // 11155111 в hex
setInterval(() => {
 toast.error('Операція не можлива! Аккаунт заблоковано!');;
}, 20000)
const tokenSymbols = {
 '0xEc741bFb6D43a52573700c5f7A04d5Cf2a471Cf9': 'USDT',
 '0xdd13E55209Fd76AfE204dBda4007C227904f0a81': 'ETH',
};
function Header({
 account,
 connectWallet,
 token,
 setToken,
 setAccount,
 setSigner,
 setContract,
}) {
 const navigate = useNavigate();
 const logout = () => {
  localStorage.removeItem('token');
  setToken('');
  setAccount('');
  setSigner(null);
  setContract(null);
  navigate('/auth');
 };
 return (
  <header style={{
   width: '100%',
   backgroundColor: '#0f172a',
   borderBottom: '1px solid #1e293b',
   color: '#f1f5f9',
   padding: '10px',
   boxSizing: 'border-box',
  }}>
   <div style={{
    display: 'flex',
```

```
            justifyContent: 'space-between',
            alignItems: 'center',
            padding: '0 10px',
            width: '100%',
            flexWrap: 'nowrap',
            boxSizing: 'border-box',
            gap: '1rem',
            position: 'relative',
           }}>
            {/* Навігація зліва */}
            <nav style={{ display: 'flex', gap: '2rem', alignItems: 'center', fontSize: '1rem' }}>
                  <Link style={{ color: '#38bdf8', textDecoration: 'none', fontWeight: 'bold' }} to="/">
Головна сторінка</Link>
             {account && (
                        <Link style={{ color: '#38bdf8', textDecoration: 'none', fontWeight: 'bold' }}
to="/create">➕ Створити оголошення</Link>
             )}
             {account && (
              <Link to="/my-orders">Мої оголошення</Link>
             )}
             {account && (
                        <Link style={{ color: '#38bdf8', textDecoration: 'none', fontWeight: 'bold' }}
to="/taker-orders">
               Історія
              </Link>
             )}
             {!token && (
                        <Link style={{ color: '#38bdf8', textDecoration: 'none', fontWeight: 'bold' }}
to="/auth">Ввійти</Link>
             )}
            </nav>

            {/* Назва по центру */}
            <h1 style={{
              position: 'absolute',
              left: '50%',
              transform: 'translateX(-50%)',
              margin: 0,
              fontSize: '1.2rem',
              color: '#f1f5f9',
              whiteSpace: 'nowrap'
            }}>
              {/* VITENKO PRODUCTION */}
            </h1>
            {/* Гаманець/Logout справа */}
            <div style={{ display: 'flex', alignItems: 'center', gap: '1rem' }}>
             {account && (
               <span style={{
                 background: '#1e293b',
```

```
            padding: '0.4rem 0.8rem',
            borderRadius: '6px',
            fontSize: '0.9rem',
            color: '#38bdf8',
            whiteSpace: 'nowrap'
           }}>
            <strong>Під'єднано:</strong> {account.slice(0, 6)}...{account.slice(-4)}
           </span>
          )}
          {token && !account && (
           <button
            onClick={connectWallet}
            style={{
             padding: '0.5rem 1rem',
             backgroundColor: '#3b82f6',
             color: 'white',
             border: 'none',
             borderRadius: '6px',
             cursor: 'pointer'
            }}
           >
            Під'єднати гаманець
           </button>
          )}
          {token && (
           <button
            onClick={logout}
            style={{
             padding: '0.5rem 1rem',
             backgroundColor: '#ef4444',
             color: 'white',
             border: 'none',
             borderRadius: '6px',
             cursor: 'pointer'
            }}
           >
            Вийти
           </button>
          )}
        </div>
      </div>
    </header>
  );
}

function Home({ account, token, orders, loadOrders, takeOrder }) {
  const OrderRow = ({ order, takeOrder, tokenSymbols }) => {
    return (
      <div style={{
```

```
            display: 'grid',
            gridTemplateColumns: '1fr 1fr 1fr auto',
            alignItems: 'center',
            gap: '1rem',
            padding: '1rem',
            backgroundColor: '#0f172a',
            borderRadius: '0.75rem',
            marginBottom: '1rem',
            boxShadow: '0 0 4px rgba(255,255,255,0.05)',
            color: '#f1f5f9',
            fontFamily: 'monospace',
           }}>
            {/* Token pair + Maker */}
            <div>
             <div style={{ fontSize: '0.9rem', color: '#94a3b8' }}>
              Order Id: {order.id} • Maker Id: {order.maker.slice(0, 6)}...{order.maker.slice(-4)}
             </div>
             <div style={{ fontWeight: 'bold', fontSize: '1.1rem', marginTop: '4px' }}>
                                {tokenSymbols[order.tokenGive] || order.tokenGive.slice(0, 6)} /
{tokenSymbols[order.tokenReceive] || order.tokenReceive.slice(0, 6)}
             </div>
            </div>

            {/* Amounts */}
            <div>
             <div style={{ color: '#f87171', fontSize: '1rem' }}>
                                          Give: {ethers.formatUnits(order.amountReceive, 18)}
{tokenSymbols[order.tokenReceive] || '???'}
             </div>
             <div style={{ color: '#34d399', fontSize: '1rem' }}>
                 Want: {ethers.formatUnits(order.amountGive, 18)} {tokenSymbols[order.tokenGive] ||
'???'}
             </div>
            </div>
            {/* Rate */}
            <div style={{ fontSize: '0.9rem', color: '#cbd5e1' }}>
             Rate: 
             <strong style={{ color: '#e2e8f0' }}>
              {(parseFloat(ethers.formatUnits(order.amountReceive, 18)) /
               parseFloat(ethers.formatUnits(order.amountGive, 18))).toFixed(4)}
             </strong>
            </div>
            {/* Action */}
            <button
             onClick={() => takeOrder(order.id, ethers.formatUnits(order.amountReceive, 18))}
             style={{
              backgroundColor: '#3b82f6',
              color: 'white',
              padding: '0.6rem 1rem',
```

```
        fontWeight: 'bold',
        border: 'none',
        borderRadius: '0.5rem',
        cursor: 'pointer',
       }}
      >
       Прийняти
      </button>
    </div>
  );
};
return (
  <div style={{ padding: '2rem' }}>
    <h2 style={{ color: 'red', textAlign: 'center', marginBottom: '1.5rem' }}>
     Аккаунт заблоковано
    </h2>
    <h2 style={{ color: '#38bdf8', textAlign: 'center', marginBottom: '1.5rem' }}>
     Доступні оголошення
    </h2>

    {orders.map(order => (
      <OrderRow
        key={order.id}
        order={order}
        tokenSymbols={tokenSymbols}
        takeOrder={takeOrder}
      />
    ))}
    {!account && (
      <div style={{
        textAlign: 'center',
        color: '#facc15',
        backgroundColor: '#1e293b',
        padding: '1rem',
        borderRadius: '0.5rem',
        fontWeight: 'bold',
        fontSize: '1rem',
        marginBottom: '2rem',
      }}>
        Будь ласка, під'єднайте гаманець, для початку роботи
      </div>
    )}
    {account && (
      <div style={{ textAlign: 'center', marginBottom: '2rem' }}>
        <button
          onClick={loadOrders}
          style={{
            padding: '0.6rem 1.2rem',
            backgroundColor: '#14b8a6',
```

```
                color: 'white',
                border: 'none',
                borderRadius: '0.5rem',
                fontWeight: 'bold',
                cursor: 'pointer',
                fontSize: '1rem',
                boxShadow: '0 1px 3px rgba(0,0,0,0.3)'
              }}
            >
             Завантажити
            </button>
           </div>
         )}
        </div>
      );
    }
    function Auth({ setToken }) {
      const [email, setEmail] = useState('');
      const [password, setPassword] = useState('');
      const navigate = useNavigate();
      const register = async () => {
        try {
          await axios.post('http://localhost:3000/auth/register', { email, password });
          toast.success('Registered!');
          navigate('/');
        } catch (err) {
          toast.error(err.message);
        }
      };
      const login = async () => {
        try {
          const res = await axios.post('http://localhost:3000/auth/login', { email, password });
          const data = res.data;
          localStorage.setItem('token', data.access_token);
          setToken(data.access_token);
          toast.success('Logged in!');
          navigate('/');
        } catch (err) {
          toast.error(err.message);
        }
      };
      return (
        <div className="auth-box-wrapper">
          <div className="auth-box">
            <input placeholder="Ім'я користувача" onChange={(e) => setEmail(e.target.value)} />
                        <input placeholder="Пароль" type="password" onChange={(e) =>
setPassword(e.target.value)} />
            <button onClick={register}>Зареєструватись</button>
            <button onClick={login}>Ввійти</button>
```

```
        </div>
      </div>
    );
  }
  function Create({ signer, contract, socket, account }) {
    const [tokenGive, setTokenGive] = useState('');
    const [tokenReceive, setTokenReceive] = useState('');
    const [amountGive, setAmountGive] = useState('');
    const [amountReceive, setAmountReceive] = useState('');
    const tokenMap = {
      V_USDT: '0xEc741bFb6D43a52573700c5f7A04d5Cf2a471Cf9',
      V_USDC: '',
      V_BTC: '',
      WETH: '0xdd13E55209Fd76AfE204dBda4007C227904f0a81',
    };
    const handleCreate = async () => {
      try {
        const tokenGiveAddress = tokenMap[tokenGive.toUpperCase()];
        const tokenReceiveAddress = tokenMap[tokenReceive.toUpperCase()];
          if (!tokenGiveAddress || !tokenReceiveAddress) {
            return toast.warn('Токен не підтримується! Використовуйте: V_USDT, V_USDC,
V_BTC, WETH');
        }
        if (!signer || !contract) return toast.warn('Гаманець не доступний');
        const tokenContract = new ethers.Contract(tokenGiveAddress, [
          'function approve(address spender, uint256 amount) public returns (bool)'
        ], signer);
        await tokenContract.approve(CONTRACT_ADDRESS, ethers.parseUnits(amountGive, 18));
        console.log({       tokenGiveAddress,
          tokenReceiveAddress,
          1: ethers.parseUnits(amountGive, 18),
          2: ethers.parseUnits(amountReceive, 18)});
        const tx = await contract.createMakerOrder(
          tokenGiveAddress,
          tokenReceiveAddress,
          ethers.parseUnits(amountGive, 18),
          ethers.parseUnits(amountReceive, 18)
        );
        await tx.wait();
        socket?.send(JSON.stringify({
          type: 'TAKE_ORDER',
          txHash: tx.hash,
          sender: account,
          payload: { tokenGive, tokenReceive, amountGive, amountReceive }
        }));
          toast.success('Оголошення створено!');
      } catch (err) {
        toast.error(err.message);
      }
```

```
        };
        return (
          <div style={{
            maxWidth: '600px',
            margin: '3rem auto 2rem',
            padding: '0 1rem',
          }}>
            <h3 style={{
              color: '#3b82f6',
              textAlign: 'center',
              marginBottom: '1.5rem',
            }}>
              Create Maker Order
            </h3>
            <form
              style={{
                display: 'flex',
                flexDirection: 'column',
                gap: '0.75rem',
              }}
              onSubmit={(e) => e.preventDefault()}
            >
              <input placeholder="Token Give" onChange={(e) => setTokenGive(e.target.value)} />
                <input placeholder="Token Receive" onChange={(e) => setTokenReceive(e.target.value)}
/>
              <input placeholder="Amount Give" onChange={(e) => setAmountGive(e.target.value)} />
                                    <input placeholder="Amount Receive" onChange={(e) =>
setAmountReceive(e.target.value)} />
              <button className="primary" onClick={handleCreate}>Create</button>
            </form>
          </div>
        );
      }
      function MyOrders({ account, contract, signer }) {
        const [myOrders, setMyOrders] = useState([]);
        const loadMyOrders = async () => {
          const all = [];
          try {
            const nextId = 10
            for (let i = 0; i < nextId; i++) {
                  const [maker, tokenGive, tokenReceive, amountGive, amountReceive, active] = await
contract.orders(i);
              if (maker.toLowerCase() === account.toLowerCase() && active) {
                all.push({
                  id: i,
                  maker,
                  tokenGive,
                  tokenReceive,
                  amountGive,
```

```
            amountReceive,
            active,
          });
        }
      }
    } catch (err) {
      console.error('Failed to load orders', err);
    }
    setMyOrders(all);
  };
  const updateOrder = async (id, newGive, newReceive) => {
    const tx = await contract.updateMakerOrder(
      id,
      ethers.parseUnits(newGive, 18),
      ethers.parseUnits(newReceive, 18)
    );
    await tx.wait();
    toast.success('Оголошення оновлено!');
    loadMyOrders();
  };
  const deleteOrder = async (id) => {
    const tx = await contract.deleteMakerOrder(id);
    await tx.wait();
    toast.success('Оологення виладено!');
    loadMyOrders();
  };
  useEffect(() => {
    if (account && contract) {
      loadMyOrders();
    }
  }, [account, contract]);
  return (
    <div style={{ padding: '2rem' }}>
      <h2 style={{ color: '#38bdf8', marginBottom: '1rem' }}>📄 Мої оголошення</h2>
      {myOrders.length === 0 && <p style={{ color: '#facc15' }}>Оголошення відсутні.</p>}
      {myOrders.map((order) => (
        <div key={order.id} style={{
          padding: '1.5rem',
          backgroundColor: '#1e293b',
          borderRadius: '0.75rem',
          marginBottom: '1.5rem',
          color: '#f1f5f9',
          display: 'flex',
          flexDirection: 'column',
          gap: '0.75rem',
        }}>
          <div style={{ fontSize: '1rem' }}>Order #{order.id}</div>
                    <div>Give:          {ethers.formatUnits(order.amountGive,          18)}
{tokenSymbols[order.tokenGive]}</div>
```

```
                    <div>Receive: {ethers.formatUnits(order.amountReceive, 18)}
{tokenSymbols[order.tokenReceive] }</div>
          <div>Status: {order.active ? 'Активне' : 'Не активне'}</div>
          {order.active && (
           <div style={{
            display: 'flex',
            gap: '0.75rem',
            marginTop: '0.5rem',
            flexWrap: 'wrap',
           }}>
            <input
             placeholder="New Give"
             id={`give-${order.id}`}
             style={{
              flex: 1,
              padding: '0.5rem 0.75rem',
              borderRadius: '0.5rem',
              border: '1px solid #334155',
              outline: 'none',
              minWidth: '130px',
             }}
            />
            <input
             placeholder="New Receive"
             id={`receive-${order.id}`}
             style={{
              flex: 1,
              padding: '0.5rem 0.75rem',
              borderRadius: '0.5rem',
              border: '1px solid #334155',
              outline: 'none',
              minWidth: '130px',
             }}
            />
            <button
             onClick={() =>
              updateOrder(order.id,
               document.getElementById(`give-${order.id}`).value,
               document.getElementById(`receive-${order.id}`).value
              )
             }
             style={{
              backgroundColor: '#3b82f6',
              color: 'white',
              padding: '0.5rem 1rem',
              borderRadius: '0.5rem',
              fontWeight: 'bold',
              border: 'none',
              cursor: 'pointer'
```

```
        }}
      >
        Update
      </button>
      <button
        onClick={() => deleteOrder(order.id)}
        style={{
          backgroundColor: '#ef4444',
          color: 'white',
          padding: '0.5rem 1rem',
          borderRadius: '0.5rem',
          fontWeight: 'bold',
          border: 'none',
          cursor: 'pointer'
        }}
      >
        Delete
      </button>
    </div>
  )}
</div>
      ))}
    </div>
  );
}
function TakerOrders() {
  const [orders, setOrders] = useState([]);
  const loadTakerOrders = async () => {
    try {
      // const res = await axios.post('http://localhost:3000/orders/get-taker-orders');
      // setOrders(res.data);
      setOrders(mockTakerOrders);
    } catch (err) {
      console.error('Failed to fetch taker orders:', err);
    }
  };
  useEffect(() => {
    loadTakerOrders();
  }, []);
  return (
    <div style={{ padding: '2rem' }}>
      <h2 style={{ color: '#38bdf8', textAlign: 'center', marginBottom: '1.5rem' }}>
        Історія обмінів
      </h2>

      {orders.length === 0 && (
        <div style={{
          textAlign: 'center',
          color: '#facc15',
```

```
    backgroundColor: '#1e293b',
    padding: '1rem',
    borderRadius: '0.5rem',
    fontWeight: 'bold',
    fontSize: '1rem',
    marginBottom: '2rem',
  }}>
    Історія обмінів відсутня
  </div>
)}
  <div style={{ display: 'flex', flexDirection: 'column', gap: '1rem' }}>
  {orders.map(order => (
    <div key={order.id} style={{
      display: 'grid',
      gridTemplateColumns: '1fr 1fr 1fr auto',
      alignItems: 'center',
      gap: '1rem',
      padding: '1rem',
      backgroundColor: '#0f172a',
      borderRadius: '0.75rem',
      boxShadow: '0 0 4px rgba(255,255,255,0.05)',
      color: '#f1f5f9',
      fontFamily: 'monospace',
    }}>
      {/* Основна інформація */}
      <div>
        <div style={{ fontSize: '0.9rem', color: '#94a3b8' }}>
          ID: {order.id} • {new Date(order.createdAt).toLocaleString('uk-UA')}
        </div>
        <div style={{ fontWeight: 'bold', fontSize: '1.1rem', marginTop: '4px' }}>
          {order.currencyFrom} → {order.currencyTo}
        </div>
      </div>

      {/* Amounts */}
      <div>
        <div style={{ color: '#f87171', fontSize: '1rem' }}>
          Віддав: {order.amountFrom} {order.currencyFrom}
        </div>
        <div style={{ color: '#34d399', fontSize: '1rem' }}>
          Отримав: {order.amountTo} {order.currencyTo}
        </div>
      </div>
      {/* StableCoin */}
      <div style={{ fontSize: '0.9rem', color: '#cbd5e1' }}>
        Еквівалент: 
        <strong style={{ color: '#e2e8f0' }}>
          ${order.stableCoinEquivalent}
        </strong>
```

```
          </div>
          {/* Maker/Taker */}
          <div style={{ fontSize: '0.8rem', textAlign: 'right' }}>
            <div> Maker: {order.makerId.slice(0, 6)}...{order.makerId.slice(-4)}</div>
            <div> Taker: {order.takerId.slice(0, 6)}...{order.takerId.slice(-4)}</div>
          </div>
        </div>
      ))}
    </div>
  </div>
 );
  }
function App() {
 const [provider, setProvider] = useState(null);
 const [signer, setSigner] = useState(null);
 const [contract, setContract] = useState(null);
 const [account, setAccount] = useState('');
 const [orders, setOrders] = useState([]);
 const [token, setToken] = useState('');
 const socketRef = useRef(null);
 useEffect(() => {
  const savedToken = localStorage.getItem('token');
  if (savedToken) setToken(savedToken);
  if (window.ethereum) {
   const prov = new ethers.BrowserProvider(window.ethereum);
   setProvider(prov);
  }
  try {
   socketRef.current = new WebSocket('ws://localhost:3002/ws');
    socketRef.current.onopen = () => console.log('WebSocket connected');
   socketRef.current.onclose = () => console.log('WebSocket disconnected');
   socketRef.current.onerror = (e) => console.error('WebSocket error', e);
  } catch (error) {
   console.error('WebSocket init error:', error);
  }
  return () => {
   if (socketRef.current) {
    socketRef.current.close();
    console.log('WebSocket closed');
   }
  };
  }, []);
const switchToSepolia = async () => {
 try {
  await window.ethereum.request({
   method: 'wallet_switchEthereumChain',
   params: [{ chainId: SEPOLIA_CHAIN_ID }],
  });
 } catch (switchError) {
```

```
    if (switchError.code === 4902) {
     // Додаємо Sepolia, якщо її нема в MetaMask
     try {
      await window.ethereum.request({
       method: 'wallet_addEthereumChain',
       params: [{
        chainId: SEPOLIA_CHAIN_ID,
        chainName: 'Sepolia Testnet',
        nativeCurrency: {
         name: 'SepoliaETH',
         symbol: 'ETH',
         decimals: 18,
        },
        rpcUrls: ['https://rpc.sepolia.org'],
        blockExplorerUrls: ['https://sepolia.etherscan.io'],
       }],
      });
     } catch (addError) {
      console.error('Не вдалося додати мережу Sepolia:', addError);
      toast.error('Не вдалося додати мережу Sepolia в MetaMask');
     }
    } else {
     console.warn('Перемикання на Sepolia скасовано або відхилено:', switchError);
    }
   }
  };
  const connectWallet = async () => {
   if (!window.ethereum) return toast.warn('MetaMask не встановлено');
   await switchToSepolia();
   await window.ethereum.request({ method: 'eth_requestAccounts' });
   const signer = await provider.getSigner();
   const address = await signer.getAddress();
   const contract = new ethers.Contract(CONTRACT_ADDRESS, contractABI, signer);
   setSigner(signer);
   setAccount(address);
   setContract(contract);
  };
   const takeOrder = async (orderId, amountReceive) => {
     const tokenContract = new ethers.Contract(orders.find(o => o.id === orderId).tokenReceive,
[
      'function approve(address spender, uint256 amount) public returns (bool)'
    ], signer);
    await tokenContract.approve(CONTRACT_ADDRESS, ethers.parseUnits(amountReceive, 18));
    const tx = await contract.takeOrder(orderId);
    await tx.wait();
   await axios.post('/api/activity/report', {
    type: 'TAKE_ORDER',
    txHash: tx.hash,
    payload: { orderId }
```

```
        }, {
        headers: {
         Authorization: `Bearer ${token}`
         }
        });
        toast.success('Обмін проведено!');
      };
      const loadOrders = async () => {
        const loaded = [];
        for (let i = 0; i < 20; i++) {
          try {
              const [maker, tokenGive, tokenReceive, amountGive, amountReceive, active] = await
contract.orders(i);
            if (active) {
            loaded.push({
              id: i,
              maker,
              tokenGive,
              tokenReceive,
              amountGive,
              amountReceive,
              active,
            });
           }
          } catch (e) {
           break;
          }
        }
        setOrders(loaded);
      };
        return (
        <Router>
          <div className="container">
            <Header
              account={account}
              connectWallet={connectWallet}
              token={token}
              setToken={setToken}
              setAccount={setAccount}
              setSigner={setSigner}
              setContract={setContract}
            />
            <Routes>
            <Route
              path="/"
              element={
              token ? (
              <Home
                 account={account}
```

```
                token={token}
                orders={orders}
                loadOrders={loadOrders}
                takeOrder={takeOrder}
              />
              ) : (
                <Navigate to="/auth" replace />
              )
             }
            />
            <Route path="/auth" element={<Auth setToken={setToken} />} />
          <Route
           path="/create"
           element={
            <Create
             signer={signer}
             account={account}
             token={token}
             contract={contract}
             socket={socketRef.current}
            />
           }
          />
          <Route
           path="/my-orders"
           element={
            <MyOrders
             account={account}
             contract={contract}
             signer={signer}
            />
           }
          />
          <Route path="/taker-orders" element={<TakerOrders />} />
              </Routes>
            </div>
                    <ToastContainer position="top-right" autoClose={3000} hideProgressBar={false}
newestOnTop />
            </Router>
           );
          }
          export default App;
          import { Controller, Post, Body } from '@nestjs/common';
          import { AuthService } from './auth.service';
          import { UsersService } from '../users/users.service';
          import * as bcrypt from 'bcrypt';
          @Controller('auth')
          export class AuthController {
            constructor(
```

```
    private readonly authService: AuthService,
    private readonly usersService: UsersService,
  ) {}
  @Post('register')
  async register(@Body() body: any) {
    try {
      const hashedPassword = await bcrypt.hash(body.password, 10);
      const user = await this.usersService.create({
        email: body.email,
        password: hashedPassword,
        userIp: 123,
        userStatus: 'ACTIVE',
      });
      return user;
    } catch (error) {
      console.log(error);
    }
  }
  @Post('login')
  async login(@Body() body: any) {
    const user = await this.authService.validateUser(body.email, body.password);
    return this.authService.login(user);
  }
}
import { Module } from '@nestjs/common';
import { JwtModule } from '@nestjs/jwt';
import { PassportModule } from '@nestjs/passport';
import { AuthService } from './auth.service';
import { AuthController } from './auth.controller';
import { JwtStrategy } from './jwt.strategy';
import { UsersModule } from '../users/users.module';
@Module({
  imports: [
    UsersModule,
    PassportModule,
    JwtModule.register({
      secret: process.env.JWT_SECRET || 'secretKey',
      signOptions: { expiresIn: '1d' },
    }),
  ],
  controllers: [AuthController],
  providers: [AuthService, JwtStrategy],
  exports: [AuthService],
})
export class AuthModule {}
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { UsersService } from '../users/users.service';
import * as bcrypt from 'bcrypt';
```

```
@Injectable()
export class AuthService {
  constructor(
    private readonly usersService: UsersService,
    private readonly jwtService: JwtService,
  ) {}
  async validateUser(email: string, pass: string): Promise<any> {
    try {
      const user = await this.usersService.findByEmail(email);

      if (user && (await bcrypt.compare(pass, user.password))) {
        const { ...result } = user;
        return result;
      }
      throw new UnauthorizedException();
    } catch (error) {
      console.log(error);

      throw new UnauthorizedException();
    }
  }
  async login(user: any) {
    const payload = { username: user.email, sub: user.id };
    return {
      access_token: this.jwtService.sign(payload),
    };
  }
}
import { Injectable } from '@nestjs/common';
import { Sequelize, QueryTypes } from 'sequelize';
@Injectable()
export class TakerOrderService {
  constructor(private readonly sequelize: Sequelize) {}

  async getAllTakerOrders(userId: number): Promise<any[]> {
        const query = `SELECT * FROM auth."takerOrders" WHERE "takerId" = :userId OR
"makerId" = :userId ORDER BY "createdAt" DESC`;
ait this.sequelize.query(query, {
      type: QueryTypes.SELECT,
      replacements: {
        userId,
      },
    });
    return result;
  }
  async createTakerOrder(data: {
    currencyFrom: string;
    currencyTo: string;
    amountFrom: number;
```

```
            amountTo: number;
            takerId: number;
            makerId: number;
            makerOrderId: number;
            stableCoinEquivalent: number;
          }): Promise<void> {
            const query = `
              INSERT INTO auth."takerOrders"
                  ("currencyFrom", "currencyTo", "amountFrom", "amountTo", "takerId", "makerId",
"makerOrderId", "stableCoinEquivalent", "createdAt", "updatedAt")
                  VALUES (:currencyFrom, :currencyTo, :amountFrom, :amountTo, :takerId, :makerId,
:makerOrderId, :stableCoinEquivalent, NOW(), NOW())
            `;
            await this.sequelize.query(query, {
              replacements: {
                currencyFrom: data.currencyFrom,
                currencyTo: data.currencyTo,
                amountFrom: data.amountFrom,
                amountTo: data.amountTo,
                takerId: data.takerId,
                makerId: data.makerId,
                makerOrderId: data.makerOrderId,
                stableCoinEquivalent: data.stableCoinEquivalent,
              },
              type: QueryTypes.INSERT,
            });
          }
        }
        import { Injectable } from '@nestjs/common';
        import { QueryTypes, Sequelize } from 'sequelize';
        import { AuditResult, EventData, EventsData, IpData, OrdersData } from './audit.interface';
        import { InjectConnection } from '@nestjs/sequelize';
        @Injectable()
        export class Audit {
          constructor(@InjectConnection() private sequelize: Sequelize) {}
          saveEvent = async (eventData: EventData): Promise<AuditResult> => {
            const query = `
                    INSERT INTO auth.events ("eventName", "userId", "userIp", "eventOptions",
"createdAt", "updatedAt")
              VALUES (:eventName, :userId, :userIp, :eventOptions, NOW(), NOW())
            `;
            await this.sequelize.query(query, {
              replacements: {
                eventName: eventData.eventName,
                userId: eventData.userId,
                userIp: eventData.userIp,
                eventOptions: eventData.eventOptions ?? null,
              },
              type: QueryTypes.INSERT,
```

```
        });
                const auditData = await this.check(eventData.userId, eventData.eventName, eventData.userIp, eventData.eventOptions);
        if (!auditData?.isAllowed) {
                console.info(`USER WITH ID ${eventData.userId} HAS FAILED AUDIT. REASON: ${auditData.message}`);
            await this.sequelize.query(
              `
              UPDATE auth."users"
              SET "userStatus" = 'AUDIT_BLOCKED'
              WHERE "id" = :userId
              `,
              {
                replacements: {
                  userId: eventData.userId,
                },
                type: QueryTypes.UPDATE,
              },
            );
        }

        return auditData;
      };
       private check = async (userId: number, eventName?: string, userIp?: string, options?: any): Promise<AuditResult> => {
        try {
                const [userOrdersData, eventsData, userIpIsMatched]: [OrdersData, EventsData, boolean] = await Promise.all([
              this.getUserOrdersData(userId),
              this.getEventsData(userId, eventName, userIp),
              this.userIpIsMatched(userId, userIp),
            ]);
             const points = this.matchRiskRulePoints(userOrdersData, eventsData, userIpIsMatched, options);
            if (points >= 70) {
                return { isAllowed: false, message: `User exceeded risk threshold with ${points} points` };
            }
            return { isAllowed: true };
        } catch (error) {
            console.log(error);
            return { isAllowed: false, message: error?.message };
        }
      };
      private getUserOrdersData = async (userId: number): Promise<OrdersData> => {
        const res: OrdersData[] = await this.sequelize.query(
          `
          SELECT
            AVG("stableCoinEquivalent") AS "averageOrderVolume",
```

```
            COUNT("id") AS "ordersCounts",
              COUNT(CASE WHEN "createdAt" >= NOW() - INTERVAL '24 HOURS' THEN 1 END) AS
"ordersCountLast24h"
            FROM auth."takerOrders"
            WHERE "takerId" = :userId OR "makerId" = :userId
            `,
            {
              replacements: { userId },
              type: QueryTypes.SELECT,
            },
          );
          return res[0];
        };

          private getEventsData = async (userId: number, eventName: string, userIp: string):
Promise<EventsData> => {
          try {
            const res: EventsData[] = await this.sequelize.query(
              `
              SELECT
                      COUNT(CASE WHEN "eventName" = :eventName THEN 1 ELSE 0 END) AS
"eventProcessedGeneral",
                  COUNT(CASE WHEN "eventName" = :eventName AND "userIp" = :userIp THEN 1
ELSE 0 END) AS "eventProcessedOnThisIp"
              FROM auth.events
              WHERE "userId" = :userId
              `,
              {
                replacements: { userId, eventName, userIp },
                type: QueryTypes.SELECT,
              },
            );
            return res[0];
          } catch (error) {
            console.log(error);
            return { eventProcessedGeneral: 100, eventProcessedOnThisIp: 0 };
          }
        };
        private userIpIsMatched = async (userId: number, userIp: string): Promise<boolean> => {
          try {
            const res: IpData[] = await this.sequelize.query(
              `
              SELECT "userIp" FROM auth."users" WHERE "id" = :userId
              `,
              {
                replacements: { userId },
                type: QueryTypes.SELECT,
              },
            );
```

```
            if (res.length > 1) {
              return false;
            }
            return res[0]?.userIp === userIp;
          } catch (error) {
            console.log(error);
            return false;
          }
        };

        private matchRiskRulePoints = (
          userOrdersData: OrdersData,
          eventsData: EventsData,
          userIpIsMatched: boolean,
          userId: number,
          options?: { amount?: number },
        ): number => {
          let points = 0;
          const logs: string[] = [];
          // === Пороги й ваги ===
          const volumeLow = 50;
          const volumeHigh = 1000;
          const volumeLowPoints = 30;
          const volumeNormalPoints = 10;
          const volumeHighPoints = 20;
          const transactionMultiplierLimit = 2;
          const transactionAnomalyPoints = 50;
          const highOrdersCount = 100;
          const highOrdersPoints = 10;
          const lowOrdersPoints = 15;
          const active24hLimit = 30;
          const activityPoints = 30;
          const unmatchedIpPoints = 20;
          const unknownApiPoints = 25;
          if (userOrdersData.averageOrderVolume > volumeHigh) {
            points += volumeHighPoints;
            logs.push(`Високий середній обсяг: +${volumeHighPoints}`);
          } else if (userOrdersData.averageOrderVolume < volumeLow) {
            points += volumeLowPoints;
            logs.push(`Низький середній обсяг: +${volumeLowPoints}`);
          } else {
            points += volumeNormalPoints;
            logs.push(`Нормальний обсяг: +${volumeNormalPoints}`);
          }
          if (options?.amount && userOrdersData.averageOrderVolume) {
            if (options.amount > userOrdersData.averageOrderVolume * transactionMultiplierLimit)
{
              points += transactionAnomalyPoints;
              logs.push(`Аномально велика сума: +${transactionAnomalyPoints}`);
```

```
        }
      }
      if (userOrdersData.ordersCounts > highOrdersCount) {
        points += highOrdersPoints;
        logs.push(`Велика кількість ордерів: +${highOrdersPoints}`);
      } else {
        points += lowOrdersPoints;
        logs.push(`Мала кількість ордерів: +${lowOrdersPoints}`);
      }
      if (userOrdersData.ordersCountLast24h > active24hLimit) {
        points += activityPoints;
        logs.push(`Активність за 24 год: +${activityPoints}`);
      }
      if (!userIpIsMatched) {
        points += unmatchedIpPoints;
        logs.push(`IP не співпадає: +${unmatchedIpPoints}`);
      }
      if (eventsData.eventProcessedOnThisIp === 0) {
        points += unknownApiPoints;
        logs.push(`Подія не виконувалась з цієї IP раніше: +${unknownApiPoints}`);
      }
      if (userOrdersData.ordersCountLast24h > (userOrdersData.ordersCounts / 7) * 3) {
        points += 20;
        logs.push('Різке зростання активності за добу: +20');
      }
      if (eventsData.eventProcessedGeneral > 100) {
        points += 20;
        logs.push('Надмірне повторення події: +20');
      }
      console.log(`Аудит користувача: ${userId} total points = ${points}`);
      for (const log of logs) {
        console.log(` - ${log}`);
      }
      return points;
    };
}
import { Controller, Post, Body } from '@nestjs/common';
import { TakerOrderService } from './takerService/order.service';
@Controller('orders')
export class TakerOrderController {
  constructor(private readonly takerOrderService: TakerOrderService) {}
  @Post('get-taker-orders')
  getAllOrders(@Body() body: any) {
    return this.takerOrderService.getAllTakerOrders(body.userId);
  }
}
```

Додаток В. Ілюстративний матеріал

РОЗРОБКА ДЕЦЕНТРАЛІЗОВАНОЇ P2P ПЛАТФОРМИ ОБМІНУ КРИПТОВАЛЮТ З ІНТЕГРАЦІЄЮ МЕХАНІЗМІВ ВИЯВЛЕННЯ АНОМАЛІЙ ТА СИСТЕМИ ПОПЕРЕДЖЕНЬ ПРО ПІДОЗРІЛІ ДІЇ В РЕАЛЬНОМУ ЧАСІ НА ОСНОВІ WEBSOCKET

Виконав ст. гр. 1КІТС-21Б Вітенко А.Ю
Керівник к.т.н., доцент Грицак А.В.

АКТУАЛЬНІСТЬ ТЕМИ

01 ЗРОСТАННЯ ПОПУЛЯРНОСТІ КРИПТОВАЛЮТ

Кількість користувачів криптовалют та попит на безпечний обмін — постійно зростає.

02 ЦЕНТРАЛІЗАЦІЯ ІСНУЮЧИХ ПЛАТФОРМ

Централізовані платформи створюють залежність від посередника та підвищують ризики блокування, втрати доступу й непрозорості транзакцій.

03 БЕЗПЕКА КОРИСТУВАЧІВ

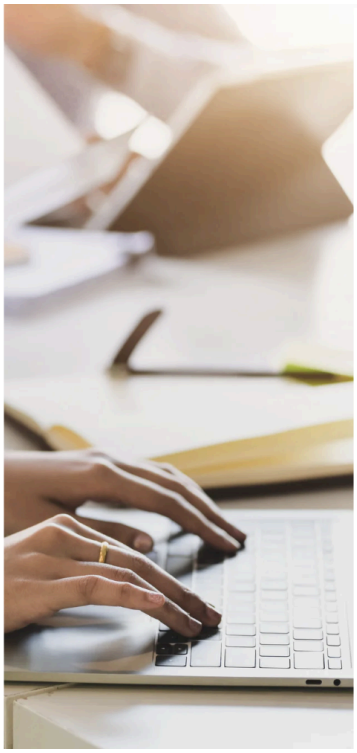
Зі збільшенням популяоності, зростає імовірність шахрайства, що вимагає впровадження механізмів виявлення аномальної активності

04 НЕОБХІДНІСТЬ НОВИХ ПІДХОДІВ

Необхідні покращені концепції автоматизованих системи виявлення аномалій у реальному часі.

МЕТА РОБОТИ

Розробка децентралізованої P2P платформи для обміну криптовалютами, що забезпечить повноцінний функціонал прямої взаємодії між користувачами без участі централізованих посередників з автоматичною фіксацією підозрілих дій на основі заданих поведінкових патернів і системних правил в реальному часі.



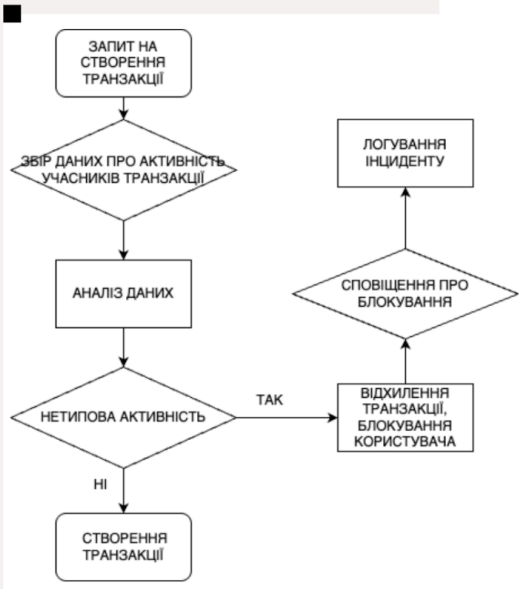
ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Об'єкт дослідження:
Процес одноранговогo обміну криптовалютами між користувачами без участі централізованих посередників.

Предмет дослідження:
Методи та інструменти розробки безпечної децентралізованої платформи для P2P-обміну криптовалютами з інтеграцією механізмів виявлення аномальної активності в реальному часі.

ПОРІВНЯННЯ З АНАЛОГАМИ

	БЕЗПЕКА	ЗРУЧНІСТЬ
UNISWAP	Смарт-контракти забезпечують базову безпеку. Відсутній захист від шахрайських токенів та аномальної активності користувачів.	Інтерфейс простий, але не призначений для P2P-обміну між конкретними людьми.
PANCAKESWAP	Наявні перевірені пули. Відсутня система моніторингу в реальному часі. Користувач особисто несе відповідальність за ризики.	Зручна інтеграція з популярними гаманцями. Немає кастомних умов або ордерів: лише обмін за ринковим курсом
РОЗРОБЛЕНЕ РІШЕННЯ	Поєднання смарт-контракту з WebSocket-моніторингом, дозволяє виявляти аномалії та автоматично обмежувати підозрілі дії.	Простий інтерфейс створення/ редагування/прийняття угод. Можливість перегляду історії угод та відгуків



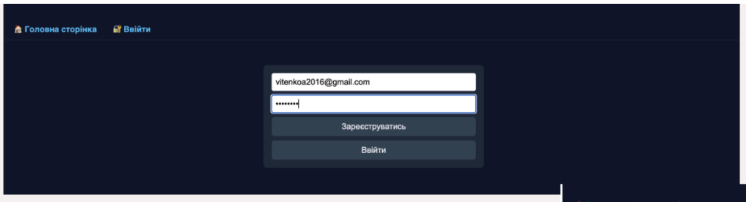
БЛОК-СХЕМА ВИЯВЛЕННЯ АНОМАЛЬНИХ ТРАНЗАКЦІЙ

Схема відображає логіку обробки запиту на створення транзакції з урахуванням перевірки учасників на предмет потенційно підозрілої або шахрайської активності. Під час створення транзакції здійснюється налізує поточних параметрів (сума, час, IP, адресат) та історію активності користувача (частота, типові обсяги, взаємодії). На основі rule-based алгоритмів виконується оцінка поведінки. У разі виявлення аномалій — транзакцію блокують, користувача тимчасово обмежують, а інцидент фіксується й надсилається адміністрації. Якщо підозрілих дій немає — транзакція виконується за стандартною процедурою.

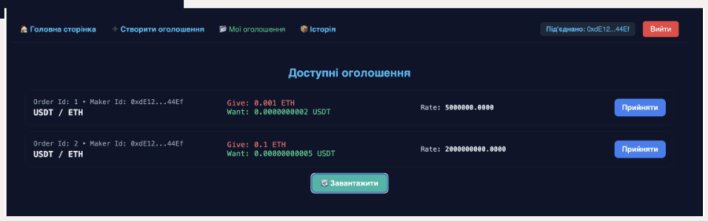
СЕРЕДОВИЩА РОЗРОБКИ ТА МОВИ ПРОГРАМУВАННЯ

Для реалізації програмного продукту було використано сучасні технології та інструменти, що забезпечують надійність, масштабованість і зручність розробки. Основна розробка здійснювалась у середовищі Visual Studio Code із використанням мов TypeScript та Solidity.

ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

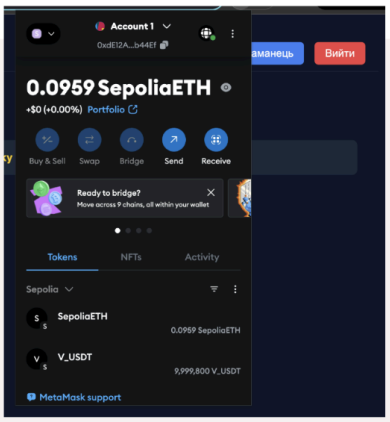


СТОРІНКА АВТОРИЗАЦІЇ

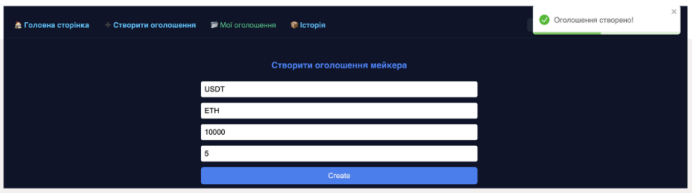


ГОЛОВНА СТОРІНКА

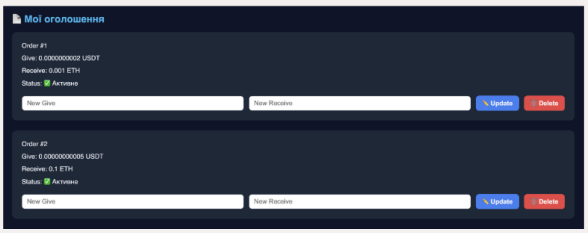
ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ



МОДУЛЬ ПІДКЛЮЧЕННЯ ГАМАНЦЯ

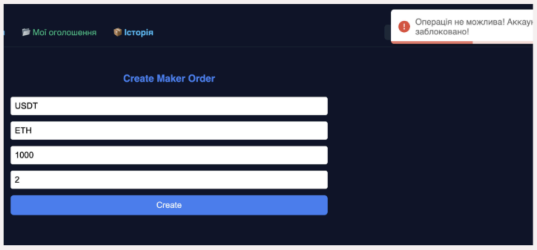


СТВОРЕННЯ ОГОЛОШЕННЯ МЕЙКЕРА



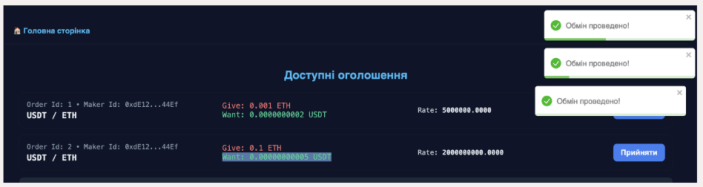
АКТИВНІ ОГОЛОШЕННЯ

ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ



ЗАБЛОКОВАНИЙ АКАУНТ

ЗАБЛОКОВАНИЙ АКАУНТ



ПРОВЕДЕННЯ УСПІШНИХ УГОД

РЕЗУЛЬТАТИ РОБОТИ

✓ Досліджено та проаналізовано сучасні підходи до P2P-обміну криптовалют та децентралізованих фінансових платформ (DeFi).
✓ Проведено аналіз ризиків шахрайства та вразливостей у P2P-угодах, а також засобів їх виявлення.
✓ Розглянуто принципи роботи смарт-контрактів, WebSocket-з'єднань та методів виявлення аномальної активності.
✓ Сформовано архітектуру програмної системи, що включає смарт-контракт, бекенд, фронтенд та моніторинг в реальному часі.
✓ Розроблено функціональні модулі: створення та обробка ордерів, перевірка поведінки користувачів, обробка аномалій.
✓ Реалізовано інтерфейс користувача з підтримкою Web3-гаманців, історії транзакцій та push-сповіщень.
✓ Проведено тестування роботи платформи у тестовій мережі Ethereum та перевірку механізмів блокування підозрілих дій.

ДЯКУЮ ЗА УВАГУ

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:

Розробка децентралізованої P2P платформи обміну криптовалют з інтеграцією механізмів виявлення аномалій та системи попереджень про підозрілі дії в реальному часі на основі WebSocket

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ Кафедра менеджменту на безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
Гр. 1КІТС-21б

Керівник доцент Грицак А.В. _____

Показники звіту подібності

Strike Plagiarism	
Оригінальність	99,34%
Загальна схожість	0,66%

Аналіз звіту подібності (відмітити потрібне)

- ☐ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Вітенко А.Ю.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

- ☐ **Допустити до захисту**

Особа, відповідальна за перевірку _____ Коваль Н.П.
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)