Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук

(повна назва)

Кафедра програмної інженерії

(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____ перший (бакалаврський)

Програмна система децентралізованого обміну токенів на платформі Solana

(тема)

Виконав:
здобувач 4 року навчання,

групи ПЗПІ-21-11

Іван МІЛЕННИЙ

(власне ім'я, прізвище)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітньо-професійна програма Програмна інженерія
(повна назва освітньої програми)

Керівник: ст.викладач Гліб ТЕРЕЩЕНКО
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук (або центр післядипломної освіти, _____

або навчально-науковий центр заочної форми навчання)___

Кафедра програмної інженерії

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 – Інженерія програмного забезпечення

(код і повна назва)

Тип програми освітньо-професійна

Освітньо-професійна програма Програмна інженерія

(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Міленному Івану Андрійовичу

(прізвище, ім'я, по батькові)

1. Тема роботи (проекту) Програмна система децентралізованого обміну токенів на платформі Solana

затверджена наказом по університету від «*19*» *травня* 2025 р. № *397Ст*

2. Термін здачі студентом закінченої роботи «13» червня 2025 р.

3. Вихідні дані до роботи методи розробки програмних продуктів, методи розробки розробки блокчейн застосунків, мови програмування.

4. Зміст пояснювальної записки (перелік питань, що потрібно розробити) мета роботи, аналіз проблемної галузі і постановка задачі, опис вимог до програмної системи, опис використовуваних методів та алгоритмів, опис розробленої програмної системи, тестування розробленої системи, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної галузі, огляд існуючих рішень, вибір найбільш придатних аналогів	16.04 – 21.04.2025	*виконано*
2	Створення Специфікації ПЗ	22.04 – 27.04.2025	*виконано*
3	Проектування та розробка ПЗ	28.04 – 20.05.2025	*виконано*
4	Тестування та дослідна експлуатація ПЗ	21.05 – 28.05.2025	*виконано*
5	Написання пояснювальної записки	29.05 – 01.06.2025	*виконано*
6	Перевірка пояснювальної записки керівником, підготовка роботи для проходження перевірки на антиплагіат проходження нормоконтролю	02.06 – 03.06.2025	*виконано*
7	Оцінка роботи рецензентом, отримання відзиву від керівника атестаційної роботи, попередній захист роботи	04.06 – 06.06.2025	*виконано*
8	Здача роботи у електронний архів, допуск роботи до захисту завідувачем кафедри	07.06 – 08.06.2025	*виконано*
9	Захист атестаційної роботи	13.06.2025	

Дата видачі завдання «04» квітня 2025 р.

Здобувач _____  _____

(підпис)

Керівник роботи _____ ст.викладач Гліб ТЕРЕЩЕНКО

(підпис)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра: 152 сторінки, 40 рисунків, 9 таблиць, 1 формула, 16 джерел, 5 додатків.

БЛОКЧЕЙН, ТОКЕН, СМАРТКОНТРАКТ, SOLANA, RUST, ANCHOR, DEFI, DEX

Об’єктом розробки є програмна система для децентралізованого обміну токенів на платформі Solana.

Метою роботи є розробка децентралізованої системи, що дозволяє користувачам безпечно і швидко здійснювати обмін токенів, створювати пули ліквідності, брати участь у запуску нових проектів (launchpool), а також взаємодіяти з цифровими активами без участі посередників.

У межах роботи проведено аналіз вимог до системи, сформовано концептуальну модель предметної області, спроектовано та розроблено ончейн-смартконтракти мовою Rust з використанням фреймворку Anchor, а також офчейн-частину для індексації подій, обробки запитів і надання даних у режимі реального часу, так клієнтський веб додаток.

Результатом роботи є програмна система, що охоплює всі ключові аспекти взаємодії з блокчейном Solana: смартконтракти для управління пулами ліквідності та launchpool, офчейн-компоненти для індексації подій та обробки запитів, а також клієнтський вебінтерфейс для взаємодії з користувачем. Розроблене рішення дозволяє безпечно й ефективно здійснювати обмін токенів у децентралізованому середовищі, сприяє зниженню порогу входу до DeFi-продуктів і демонструє реальні можливості платформи Solana для побудови масштабованих Web3-застосунків.

ABSTRACT

BLOCKCHAIN, TOKEN, SMART CONTRACT, SOLANA, RUST, ANCHOR, DEFI, DEX

The object of the development is a software system for decentralized token exchange on the Solana blockchain platform.

The goal of this work is to develop a decentralized system that enables users to securely and efficiently exchange tokens, create liquidity pools, participate in the launch of new projects (launchpool), and interact with digital assets without intermediaries.

The work includes system requirements analysis, the formation of a conceptual domain model, the design and implementation of on-chain smart contracts in Rust using the Anchor framework, as well as the off-chain infrastructure for event indexing, request handling, and real-time data delivery. A client-facing web application has also been developed.

The final result is a complete software system that covers all key aspects of interaction with the Solana blockchain: smart contracts for managing liquidity pools and launchpools, off-chain components for event indexing and request processing, and a web interface for user interaction. The developed solution allows secure and effective token exchange in a decentralized environment, lowers the entry barrier for users engaging with DeFi products, and demonstrates the practical capabilities of Solana as a platform for building scalable Web3 applications.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

DEX – Decentralized Exchange (Децентралізована біржа)

ETH – Ethereum

USD – United States Dollar (долар США)

DeFi – Decentralized Finance (децентралізовані фінанси)

AMM – Automated Market Maker (алгоритм автоматичного маркет-мейкінгу)

CLMM – Concentrated Liquidity Market Maker (алгоритм концентрованої ліквідності)

CPMM – Constant Product Market Maker (алгоритм постійного добутку)

PoH – Proof of History (доказ історії)

UI – User Interface (інтерфейс користувача)

UX – User Experience (досвід користувача)

UNI – Uniswap Token (токен Uniswap)

RAY – Raydium Token (токен Raydium)

ORCA – Orca Token (токен ORCA)

ВСТУП

У сучасному цифровому середовищі відбувається стрімкий розвиток децентралізованих фінансових технологій. Все більше користувачів прагнуть здійснювати обмін цифрових активів без посередників, централізованого контролю та високих комісій. Водночас зростає попит на рішення, які забезпечують прозорість, доступність і безпечність фінансових операцій. Актуальності набувають децентралізовані біржі, які дозволяють користувачам напряму взаємодіяти з токенами та створювати пули ліквідності.

Серед відомих рішень можна виділити Uniswap, PancakeSwap, Raydium та Orca. Проте жодне з них не поєднує гнучку архітектуру, високошвидкісну обробку подій та launchpool-механізмів в межах однієї системи. Це створює потребу у нових архітектурних рішеннях, які базуються на швидких блокчейнах із дешевими транзакціями.

У зв’язку з цим, Solana виступає перспективною платформою для побудови децентралізованих фінансових застосунків. Завдяки своїй архітектурі, що поєднує алгоритм Proof of History, горизонтальне масштабування та підтримку паралельного виконання транзакцій, Solana забезпечує високу продуктивність та мінімальні комісії, що критично важливо для користувачів DEX-платформ.

Метою даної роботи є розробка програмної системи децентралізованого обміну токенів, яка буде поєднувати в собі ончейн-смартконтракти, офчейн-компоненти для індексації та обробки подій, а також клієнтський вебінтерфейс для інтерактивної взаємодії з користувачем. Запропонована система має забезпечити зручну, масштабовану та безпечну взаємодію з цифровими активами у межах Web3-екосистеми.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Обмін як економічна і соціальна практика виник ще до формування держав і письма, і в кожну історичну епоху він зазнавав змін під впливом розвитку технологій, форм власності, інституцій і довіри між учасниками. Історія обміну між людьми є відзеркаленням еволюції суспільства та економіки – від перших бартерних операцій до високотехнологічних цифрових платформ. Сукупним результатом багатовікової еволюції ринку є народження централізованої інфраструктури, яка забезпечує стандартизацію контрактів, ліквідність, регуляторний контроль та зручність обміну. Саме на цьому підґрунті в кінці 2000-х з’явилася технологія блокчейн і криптовалюти, які кинули виклик традиційній монополії бірж та банків.

На момент проведення дослідження домінуючим інструментом для здійснення токен-обмінів є блокчейн другого покоління – Ethereum, розроблений Віталіком Бутеріним у 2015 році. Ця платформа стала першою, що інтегрувала концепцію Turing-complete смартконтрактів (див. рисунок 1.1), які виконуються у децентралізованому середовищі – на кожному вузлі мережі. Смартконтракти, описані в «Ethereum White Paper» [1], дали змогу створювати програмовані фінансові логіки без потреби в централізованому посереднику. Це стало основою для розвитку таких сфер, як DeFi, NFT, DAO та децентралізовані біржі. Саме завдяки цій архітектурі з’явилися перші DEX-платформи на кшталт Uniswap.

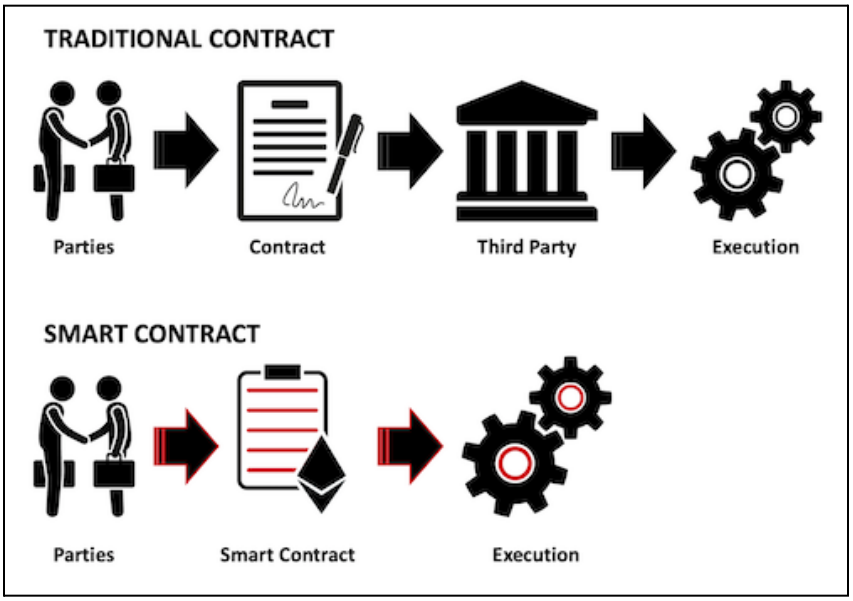

Рисунок 1.1 – Смарт-контракт [2]

Попри своє домінування, Ethereum стикається з рядом викликів, пов’язаних із масштабованістю, високими комісіями за транзакції та відносно низькою пропускною здатністю. Це стало підґрунтям для появи численних блокчейнів третього покоління, найперспективнішим та найвідомішим серед яких є Solana (див. рисунок 1.2). Цей блокчейн може паралельно оброблювати транзакції та використовує унікальний механізм консенсусу Proof of History, який був розроблений та описаний Анатолієм Яковенко у документі «Solana: A New Architecture for a High Performance Blockchain» [3]. Такий підхід дозволив досягати тисяч транзакцій за секунду при затримці в сотні мілісекунд і зберігати середню комісію на рівні ~0,00025 USD.

Рисунок 1.2 – Логотип Solana [4]

Таким чином, Ethereum перевів блокчейн у площину програмованих операцій за активами, а Solana довела, що це можливо робити дешево та швидко. Сформована на цій базі галузь DeFi стала доказом, що обмін і кредитування можуть працювати без банків і брокерів.

Хоча для обміну токенів використовуються блокчейни, вони є лише платформою для обробки транзакцій, сама логіка обміну реалізована в смартконтрактах децентралізованих бірж (DEX). На концептуальному рівні DEX розв’язують дві ключові проблеми. По-перше, зберігання активів лишається у гаманцях користувачів, а контракт лише змінює їхній стан. По-друге, кліринг виконується алгоритмом мережі й не потребує довіри до конкретної організації. Водночас DEX – це нова площина ризику – помилки в коді, атаки на оракули, імперманентні втрати постачальників ліквідності. Щоб мінімізувати їх, сучасні протоколи запроваджують багаторівневі аудити, мультипідписні механізми оновлень та on-chain-страхування. Проте, через швидкість розвитку, гарантії відсутності вразливостей аудити лишаються відкритим питанням.

Таким чином, децентралізовані біржі є новою сходинкою у розвитку фінансових ринків – вони поєднують відкритість коду, самостійне зберігання активів і прозорість механізмів обміну. Водночас DEX ще перебувають у фазі активного становлення: архітектури стрімко еволюціонують, стандарти безпеки формуються, а досвід кінцевого користувача лише наближається до зручності централізованих сервісів. Саме тому створення програмних систем, що використовують технології DEX і розширюють їхню доступність, є актуальним напрямом розробки в межах сучасної блокчейн-економіки.

1.2 Виявлення та вирішення проблеми

Децентралізовані обмінники покликані забезпечити безпечний і прозорий обмін цифровими активами без посередників. Вони повинні задовольняти низку критичних вимог і водночас долати характерні проблеми.

Основні вимоги до DEX:

- прозорість усіх транзакцій завдяки публічному блокчейну;
- швидка індексація історії операцій і поточних даних з блокчейна
- достатня ліквідність для популярних торгових пар;
- низькі комісії та швидке підтвердження угод;
- дружній інтерфейс (UI/UX) і просте підключення гаманців;
- стабільна робота смарт-контрактів і відсутність вразливостей;
- гнучке масштабування під час пікових навантажень.

Поширені проблеми DEX:

- обмежена пропускна здатність мережі та/або високі комісії;
- відсутність швидкої індексації історії операцій і поточних даних з блокчейна;
- уразливості смарт-контрактів, експлойти й шахрайство;
- імперманентні втрати постачальників ліквідності;
- відсутня або обмежена кросчейн інтеграція;
- ризики MEV (front-running, sandwich-атаки);
- регуляторна невизначеність;
- бар’єр входу для новачків через складність DeFi-термінів і гаманців.

Для оцінки того, як сучасні проекти розв’язують ці задачі, розглянемо три найпомітніші DEX – Uniswap, Orca та Raydium.

Uniswap є однією з перших і найвідомішою децентралізованою біржею, що працює на основі блокчейна Ethereum. Платформа стала ключовим каталізатором

розвитку DeFi-сектору, розпочавши нову еру обміну цифрових активів без посередників. Саме в межах Uniswap [5] був уперше реалізований алгоритм постійного добутку (CPMM), що заклав основу для більшості AMM-протоколів, які з'явилися згодом. Надалі, з виходом Uniswap V3, платформа впровадила концепцію концентрованої ліквідності (CLMM), що дозволила провайдерам самостійно визначати цінові діапазони розміщення активів, значно підвищивши ефективність використання ліквідності.

Uniswap орієнтована винятково на спотову торгівлю, без підтримки ордер-буків, деривативів, створення токенів або нативних кросчейн-механізмів. Усі операції виконуються через смарт-контракти Ethereum, що забезпечує високу безпеку, проте накладає обмеження у вигляді високих комісій та обмеженої пропускної здатності мережі.

Uniswap має власний нативний токен UNI, який використовується для управління протоколом та стимулювання учасників екосистеми. Разом із тим, інтерфейс платформи має ряд функціональних обмежень: зокрема, відсутня історія торгів і детальна статистика транзакцій, що ускладнює аналіз для користувачів, які не використовують сторонні сервіси або блокчейн-оглядачі.

Незважаючи на відсутність розширеної функціональності, Uniswap (див. рисунок 1.3) залишається еталонною DEX-платформою і виконує роль інфраструктурного стандарту для децентралізованого трейдингу в екосистемі Ethereum та є найліквіднішею DEX на момент проведення дослідження. Її технологічні нововведення — зокрема алгоритми CPMM і CLMM – лягли в основу значної частини подальших рішень у галузі, а сам проект залишається домінуючим додатком на блокчейні Ethereum.

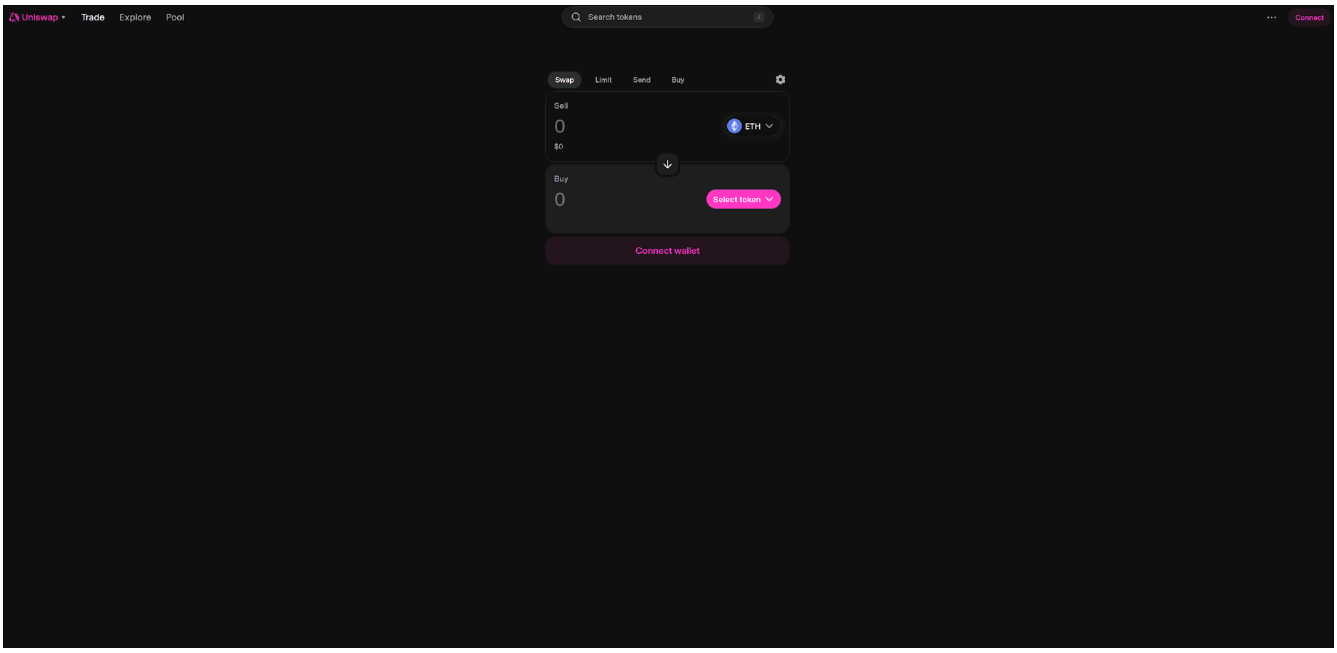

Рисунок 1.3 – Інтерфейс Uniswap [6]

Orca є однією з перших децентралізованих бірж в екосистемі Solana. Архітектура платформи базується на класичному механізмі автоматизованого маркетмейкера, проте згодом було впроваджено підтримку Whirlpool-пулів, які реалізують алгоритм концентрованої ліквідності [7]. Це дає змогу постачальникам ліквідності задавати вузький ціновий діапазон для розміщення активів, що підвищує ефективність капіталу та сприяє зменшенню ризиків розводнення доходу.

Однією з характерних переваг Orca є мінімалістичний та інтуїтивно зрозумілий інтерфейс, що забезпечує доступність як для досвідчених користувачів, так і для новачків у сфері DeFi. Висока пропускна здатність мережі Solana у поєднанні з низькими комісіями робить платформу зручною для щоденної торгівлі, дрібних обмінів та ознайомчих операцій.

Крім того, платформа надає інструмент для створення власних токенів, що становить практичний інтерес для малих проектів, стартапів і децентралізованих спільнот в межах Solana. Однак відсутність нативної кросчейн-інтеграції обмежує

можливість прямої взаємодії з іншими блокчейн-мережами та звужує потенційний обсяг залученої ліквідності.

Orca має власний нативний токен ORCA, який виконує роль токена управління. Його власники можуть брати участь у процесах ухвалення рішень щодо розвитку протоколу. Окрім цього, токен використовується для стимулювання учасників екосистеми.

Узагальнюючи, Orca є прикладом класичної DEX-платформи, орієнтованої на простоту, високу швидкодію та базову функціональність. Її довготривала присутність на ринку, підтримка сучасних механізмів управління ліквідністю та інструменти токенізації роблять Orca (див. рисунок 1.4) привабливим рішенням для широкого кола користувачів в межах екосистеми Solana.

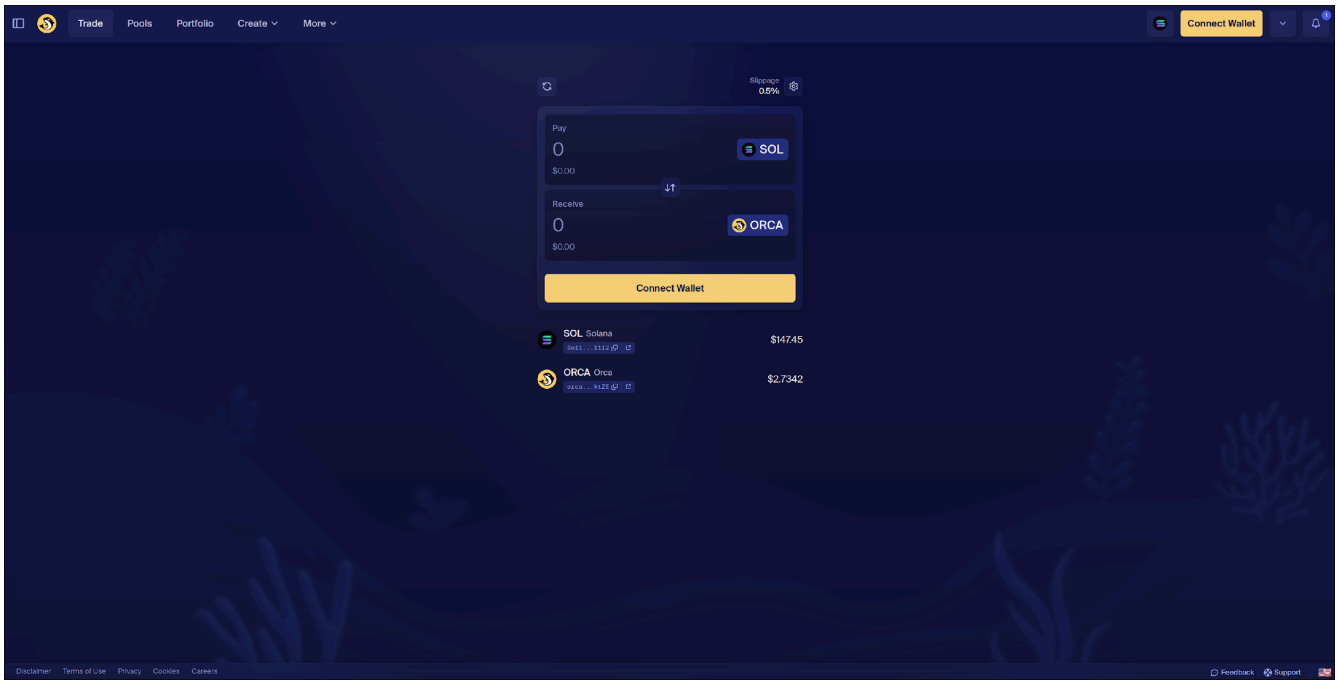

Рисунок 1.4 – Інтерфейс Orca [8]

Raydium є однією з найбільш технологічно розвинених децентралізованих бірж в екосистемі Solana, архітектура якої забезпечує надзвичайно високу

швидкість обробки транзакцій та мінімальні комісії завдяки особливостям базової мережі.

Відмінною рисою платформи є інтеграція з ордер-буком OpenBook, що поєднує елементи традиційної централізованої торгівлі з AMM-ліквідністю [9]. Така гібридна модель дає змогу користувачам взаємодіяти як із ліквідністю на платформі, так і з глобальною книгою ордерів, що сприяє більш точному формуванню ринкових цін та підвищенню ефективності виконання угод. Також Raydium впровадив підтримку пулів з концентрованою ліквідністю.

Raydium має власний токен RAY, який грає ключову роль у функціонуванні платформи. Насамперед він використовується для управління розвитком протоколу. Крім того, токен застосовується в стейкінгу, дозволяючи користувачам отримувати додаткову винагороду за блокування активів. RAY також використовуються для стимулювання пулів ліквідності, яке робить пул більш привабливим для провайдерів. Таким чином, токен RAY виконує як утилітарну, так і економічну функцію в межах Raydium.

Окрім спотової торгівлі, платформа реалізує підтримку деривативів, зокрема ф’ючерсних контрактів, що відкриває додаткові можливості для професійних учасників ринку та складніших торгових стратегій. Для забезпечення кросчейн сумісності Raydium використовує протокол Wormhole, що дає змогу здійснювати перекази цифрових активів між Solana, Ethereum, Binance Smart Chain, Polygon та іншими мережами. Це сприяє збільшенню ліквідності та залученню користувачів поза межами нативного блокчейну.

На момент проведення дослідження Raydium (див. рисунок 1.5) є флагманською DEX-платформою в мережі Solana та найтехнологічнішою серед усіх DEX.

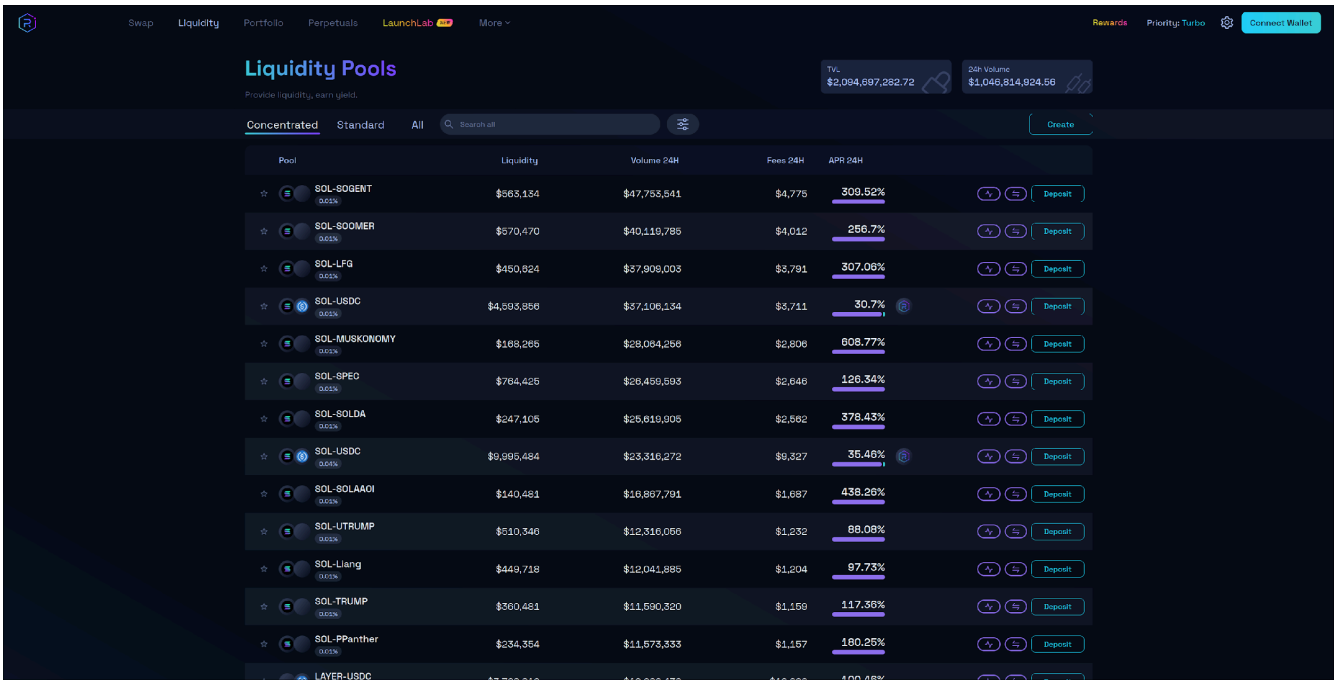

Рисунок 1.5 – Інтерфейс Raydium [10]

Метою подальшого етапу дослідження є проектування та розробка децентралізованого обмінника, архітектура якого враховуватиме як переваги, так і недоліки проаналізованих рішень.

Система повинна забезпечувати високу швидкодію шляхом використання блокчейну з великою пропускною здатністю, низькими комісіями та підтримкою паралельної обробки транзакцій. Архітектура смарт-контрактів має бути оптимізована для мінімізації затримок навіть за умов пікового навантаження, що дозволить досягати стабільного часу підтвердження в межах кількох секунд. Смарт-контракти повинні бути ефективними, оптимізованими та зберігати витрати на транзакції на мінімальному рівні.

Інтерфейс користувача повинен бути простим, інтуїтивно зрозумілим та доступним для широкого кола користувачів. Для цього слід впровадити мінімалістичний дизайн, а також зручну навігацію по ключових функціях: обмін токенів, управління активами, взаємодія з пулами ліквідності. Додатково має бути

передбачена локальна валідація введених даних, підказки для користувачів та повідомлення.

Індексація даних з блокчейну повинна здійснюватися через серверну частину системи, яка відповідатиме за зчитування, обробку та кешування необхідної інформації. Такий підхід дозволить зменшити навантаження на клієнтський додаток та блокчейн, прискорити доступ до історії транзакцій, даних про пули ліквідності та інші ключові параметри, а також забезпечити можливість аналітики без прямої взаємодії користувача з повноцінними вузлами мережі.

Система повинна підтримувати запуск нових проектів через механізм публічного стейкінгу, який надає змогу користувачам розподіляти власні активи на підтримку нових ініціатив. Таке рішення сприятиме децентралізованому фінансуванню та залученню спільноти до розвитку екосистеми. Проекти зможуть ініціювати пули винагород та залучати підтримку без централізованих посередників.

Платформа має реалізовувати автоматичне управління ліквідністю через пули обміну, що базуються на алгоритмі CPMM. Такий підхід дозволяє забезпечити постійну ліквідність незалежно від обсягу ордерів, а також гарантує просту й надійну модель ціноутворення. Постачальники ліквідності повинні мати можливість додавати або вилучати активи з пулів, отримуючи пропорційну частку комісій.

Ключовим елементом внутрішньої економіки платформи має стати токен VON, який виступатиме засобом доступу до розширених функцій системи. Зокрема, участь у лаунчпулах потребуватиме певної кількості VON, що буде блокуватись у вигляді депозиту для стейкінгу.

1.3 Постановка задачі

Розробка програмної системи децентралізованого обміну токенів на базі блокчейну Solana передбачає створення інтегрованого рішення, що поєднує в собі смарт-контракти, серверну частину для індексації та кешування даних, а також клієнтський вебінтерфейс для взаємодії з платформою. На відміну від традиційних централізованих бірж, які базуються на контролі з боку окремих компаній, бізнес-логіка запропонованої система має бути повністю децентралізованою.

Реалізація такої системи супроводжується низкою технічних викликів. Зокрема, необхідно забезпечити коректну взаємодію між централізованими та децентралізованим компонентами системи. Додаткові труднощі виникають під час розробки безпечних смарт-контрактів із врахуванням специфіки платформи Solana. Індексація подій у реальному часі вимагає високої продуктивності серверної частини. Крім того, фронтенд повинен не лише ефективно взаємодіяти з API, а й забезпечувати коректне та швидке відображення оновлень у режимі реального часу.

Вибір блокчейну Solana зумовлений її високою пропускною здатністю, підтримкою паралельного виконання транзакцій та низькими комісіями. Завдяки використанню алгоритму Proof of History і механізмів горизонтального масштабування, Solana дозволяє створювати високонавантажені застосунки. Це критично важливо для децентралізованих бірж, де навіть незначні затримки або підвищені витрати на комісію можуть призводити до відтоку користувачів.

Окрім вибору блокчейну, значна увага приділяється інструментам і фреймворкам розробки. Смарт-контракти реалізовуватимуться мовою Rust із використанням фреймворку Anchor, що забезпечує декларативну модель розгортання, чітку типізацію та зменшує ризик помилок завдяки вбудованим валідаціям. Серверна частина базуватиметься на фреймворку Axum, який дозволяє

досягти високої продуктивності при роботі з API, а також підтримує асинхронні операції через Tokio Runtime. Для зберігання структурованих даних буде використано ScyllaDB – високошвидкісну базу даних, сумісну з Cassandra API, яка дозволяє обробляти великі обсяги записів із мінімальними затримками.

Фронтенд-платформа буде реалізована за допомогою бібліотеки React та фреймворку Next.js. Для взаємодії з блокчейном використовуватиметься бібліотека @solana/web3.js.

1.4 Мета роботи

Метою даної роботи є розробка програмної системи децентралізованого обміну токенів, що функціонує на базі блокчейну з високою пропускною здатністю та мінімальними комісіями. Система повинна забезпечити користувачам можливість здійснювати швидкий, прозорий та безпечний обмін цифрових активів, створювати пули ліквідності та підтримувати нові проекти без участі посередників.

Для досягнення поставленої мети необхідно виконати наступні завдання:

- реалізація алгоритмів роботи пулу ліквідності;
- створення токена платформи;
- реалізація механізму обміну токенів через пули ліквідності;
- реалізація контракту лаунчпулу та логіки стейкінгу;
- розробка інтерфейсу для взаємодії з пулами ліквідності;
- розробка інтерфейсу для взаємодії з launchpool;
- реалізація індексації подій із блокчейну через бекенд;
- розробка API для доступу до індексованих даних;
- написання тестів для смарт-контрактів і бекенду.

2 ФОРМУВАННЯ ВИМОГ ДЛЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Загальна характеристика продукту

Розроблювана програмна система є децентралізованим застосунком для обміну токенів, що функціонує на основі блокчейну Solana. Метою її створення є забезпечення користувачів зручним і безпечним інструментом для виконання операцій із цифровими активами без потреби у централізованих посередниках.

Назва програмної системи – Vondex. Вона поєднує в собі швидкість, відкритість та надійність. Продукт розробляється з урахуванням сучасних викликів і тенденцій ринку децентралізованих фінансів. Його концепція спирається на прагнення зробити фінансові інструменти доступними для ширшого кола користувачів, незалежно від їхнього місця розташування, наявності банківського обслуговування чи рівня досвіду у сфері криптовалют.

На відміну від централізованих бірж, де управління активами здійснюється через єдину керуючу структуру, Vondex покладається на децентралізовану логіку взаємодії. Усі дії користувача – обмін токенів, додавання чи вилучення активів із пулів ліквідності, або участь у підтримці проектів – відбуваються згідно з чітко визначеними правилами, які вже вбудовані в логіку самої системи. Ці правила не можуть бути змінені або обійдені кимось довільно, що забезпечує високий рівень надійності, зменшує ризик зловживань і гарантує дотримання принципів рівного доступу до фінансових можливостей.

Функціональне призначення Vondex не обмежується лише виконанням обмінних операцій. Система також виступає майданчиком для підтримки нових проектів через механізм лаунчпулів – публічних стейкінгових кампаній, які дозволяють користувачам підтримувати розвиток певного токена, отримуючи за це винагороди. Такий підхід стимулює розвиток внутрішньої економіки платформи та сприяє формуванню активної спільноти навколо перспективних ініціатив.

Цільова аудиторія Vondex охоплює як досвідчених користувачів криптовалютного ринку – інвесторів, трейдерів, розробників, так і новачків, які лише починають взаємодію з цифровими активами. Оскільки система пов’язана із виконанням фінансових операцій, вона орієнтована на повнолітню аудиторію, що відповідає чинним нормативним обмеженням у галузі обігу криптовалют. При цьому особлива увага приділяється зручності інтерфейсу, наочності навігації та зрозумілості базових процесів – що дозволяє зменшити бар’єр входу для користувачів без попереднього досвіду.

Середовищем розгортання системи є Solana Devnet та Solana Testnet, яка забезпечує необхідний рівень масштабованості, низькі комісії та високу швидкодію. Ці властивості блокчейну дозволяють системі стабільно функціонувати навіть за умов високого навантаження, забезпечуючи короткий час підтвердження транзакцій і зручність взаємодії для кінцевого користувача. Клієнтська частина застосунку, а також серверна підсистема, що відповідає за індексацію подій та обробку даних, будуть розгорнуті у окремих серверних додатках. Такий підхід дозволяє забезпечити стабільну роботу користувацького інтерфейсу, оперативне оновлення інформації та швидкий доступ до історії транзакцій, пулів ліквідності й інших динамічних параметрів платформи.

Таким чином, Vondex позиціонується як сучасний децентралізований застосунок. Він поєднує у собі зручність користування, доступність фінансових інструментів та потенціал для розширення у рамках безпечного й надійного середовища.

2.2 Технічні особливості

Програмна система Vondex розробляється як децентралізована вебплатформа, орієнтована на забезпечення високої продуктивності, стабільності та масштабованості в умовах постійної взаємодії з блокчейн-мережею Solana.

Архітектура системи має бути побудована з чітким розподілом функцій між клієнтською, серверною та блокчейн-частиною.

Блокчейн-інфраструктурою платформи будуть Solana Devnet та Solana Testnet – тестова мережа, яка забезпечує наближені до основної мережі параметри роботи. Це дасть змогу реалізовувати, перевіряти й удосконалювати всі ключові функції без фінансових витрат на транзакції, зберігаючи при цьому умови реального середовища.

Клієнтська частина застосунку буде реалізована як односторінковий вебінтерфейс із використанням сучасного JavaScript-фреймворку Next.js, що забезпечує динамічне оновлення даних, інтерактивність, серверний рендеринг і гнучку модульність. Застосунок повинен мати підтримку криптогаманців, для відправки транзакцій напряму без посередництва або попередньої реєстрації. Основна взаємодія з блокчейном буде здійснюватись за допомогою бібліотеки @solana/web3.js.

Серверна частина відповідатиме за індексацію подій із блокчейну, кешування даних, обробку метаданих токенів, зберігання історії транзакцій та інших динамічних об’єктів. Вона буде реалізована мовою програмування Rust з використанням високопродуктивного вебфреймворку Axum, що забезпечує асинхронну обробку великої кількості запитів при мінімальних затримках. Для збереження структурованих і проіндексованих даних була обрана масштабована розподілена база даних ScyllaDB, яка може ефективно працювати з великим обсягом записів у реальному часі. Комунікація між клієнтським інтерфейсом та сервером повинна здійснюватись через REST API, а оновлення критичних даних, таких як статус транзакцій або змінні ліквідності, повинні передаватися у режимі реального часу через WebSocket-підключення.

Оскільки Vondex працює з реальними гаманцями користувачів і виконує взаємодію з блокчейном напряму, всі транзакції мають бути заздалегідь підписані

самим користувачем. Серверна частина не повинна зберігати приватних ключів або іншої конфіденційної інформації, дотримуючись принципів Web3-дизайну.

Клієнтська та серверна частини застосунку можуть бути розгорнуті локально, що дозволяє розробникам, тестувальникам і адміністраторам розгорнути повну інстанцію системи на власному пристрої.

Для коректної роботи програмної системи необхідно дотримуватись наведених вимог до середовища:

- наявність сучасного веббраузера з підтримкою WebAssembly і розширень для криптогаманців (рекомендується Google Chrome або Mozilla Firefox);
- активне підключення до Інтернету для доступу до мережі Solana, взаємодії з вузлами блокчейну та отримання оновлень у реальному часі.

2.3 Інтерфейс веб-клієнта

Інтерфейс вебклієнта програмної системи Vondex буде розроблений з урахуванням функціональної простоти, інтуїтивної зрозумілості та мінімалістичного дизайну. Його структура повинна забезпечувати швидкий доступ до основних модулів платформи: обміну токенів, керування активами, участі у пулах ліквідності та підтримки проектів через launchpool-механізм.

Компонування елементів інтерфейсу повинно відповідати єдиній логічній моделі, яка мінімізуватиме когнітивне навантаження на користувача. Основні функціональні блоки будуть згруповані за контекстом використання та розміщені в зоні прямої візуальної доступності.

Навігація буде організована у вигляді горизонтального або бічного меню з фіксованими розділами, що дасть змогу користувачам миттєво перемикатися між функціональними секціями без перезавантаження сторінки.

Особливу увагу буде приділено зворотному зв’язку – інтерфейс міститиме підказки, повідомлення про помилки, підтвердження успішного виконання дій, а

також попередження про потенційні ризики (наприклад, при взаємодії з неперевіреним токеном). Усі повідомлення повинні подаватися стисло й недвозначно, з чітким поясненням причин і подальших дій.

Інтерфейс повинен забезпечувати баланс між технічною складністю функціоналу та зручністю використання, створюючи комфортне середовище як для досвідчених користувачів, так і для новачків у сфері децентралізованих фінансів.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ

3.1 Концептуальне моделювання

Концептуальне моделювання предметної області дозволяє діаграми прецедентів і побудувати моделі даних для програмної системи.

Предметна область передбачає наявність двох основних типів користувачів.

Першим типом користувача є неавторизований користувач. Такий користувач має обмежений функціонал і не може здійснювати жодних змін у системі. Він повинен мати можливість переглядати наявні пули ліквідності та лаунчпули, здійснювати пошук, а також виконати під’єднання криптогаманця через інтегрований адаптер. Цей тип користувача є початковим станом взаємодії з платформою.

Описані функціональні потреби повністю відповідають необхідному функціоналу для даного типу користувача. Вони проілюстровані у вигляді діаграми прецедентів на рисунку 3.1.

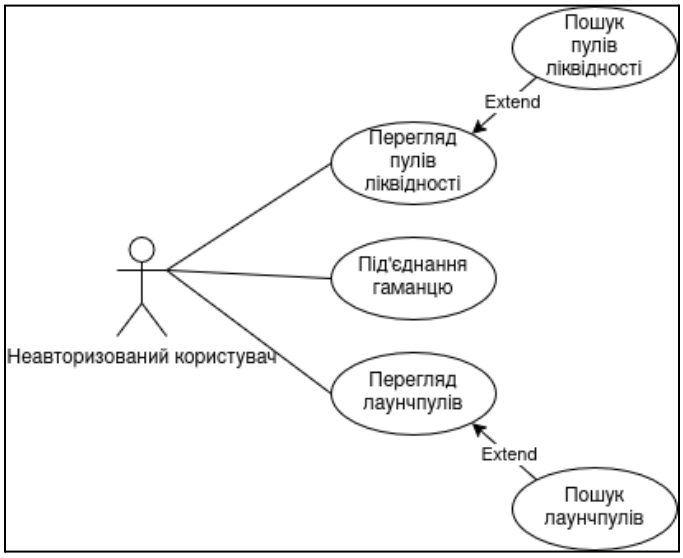

Рисунок 3.1 – Діаграма прецедентів для неавторизованого користувача (рисунок виконаний самостійно)

Другим типом користувача є авторизований користувач. Це основний тип користувача, для якого відкривається повний спектр можливостей децентралізованої платформи. Користувач повинен мати змогу переглядати всі існуючі пули ліквідності на платформі, здійснювати вибір пулу для торгівлі, проводити обмін токенів у вибраному пулі, додавати або вилучати ліквідність, збирати комісії для платформи, а також створювати нові пули ліквідності. Крім цього, користувач повинен мати доступ до власного профілю, у якому відображається історія взаємодій із платформою та активи, що йому належать. Додатково він повинен мати можливість переглядати доступні лаунчпули, брати участь у них через механізм стейкінгу токенів VON, а також отримувати відповідні нагороди.

Перелічені функціональні можливості відповідають повному переліку дій, передбачених для авторизованого користувача, і зображені у вигляді діаграми прецедентів на рисунку 3.2.

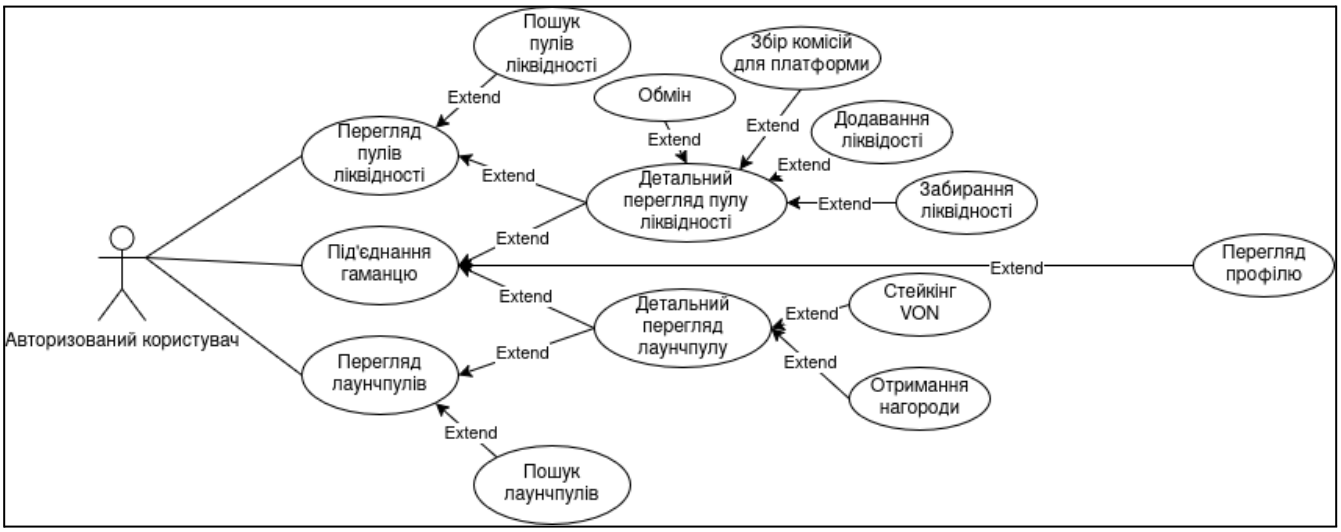

Рисунок 3.2 – Діаграма прецедентів для авторизованого користувача (рисунок виконаний самостійно)

Адміністратор launchпулів є підтипом авторизованого користувача, який наділений розширеними правами для управління launchпулами на платформі. Після під'єднання гаманця адміністратор отримує доступ до сторінки адміністратора launchпулів, де має змогу переглядати поточні launchпули, здійснювати пошук, ініціалізувати нові launchпули та запускати їх. Окрім цього, адміністратор може переглядати, додавати нові конфігурації launchпулів та змінювати їх статус, а також передавати свої адміністративні права іншій адресі. Цей тип користувача забезпечує контроль та гнучкість у керуванні публічними стейкінговими кампаніями на платформі.

Відповідні функціональні можливості адміністратора launchпулів подано у вигляді діаграми прецедентів на рисунку 3.3.

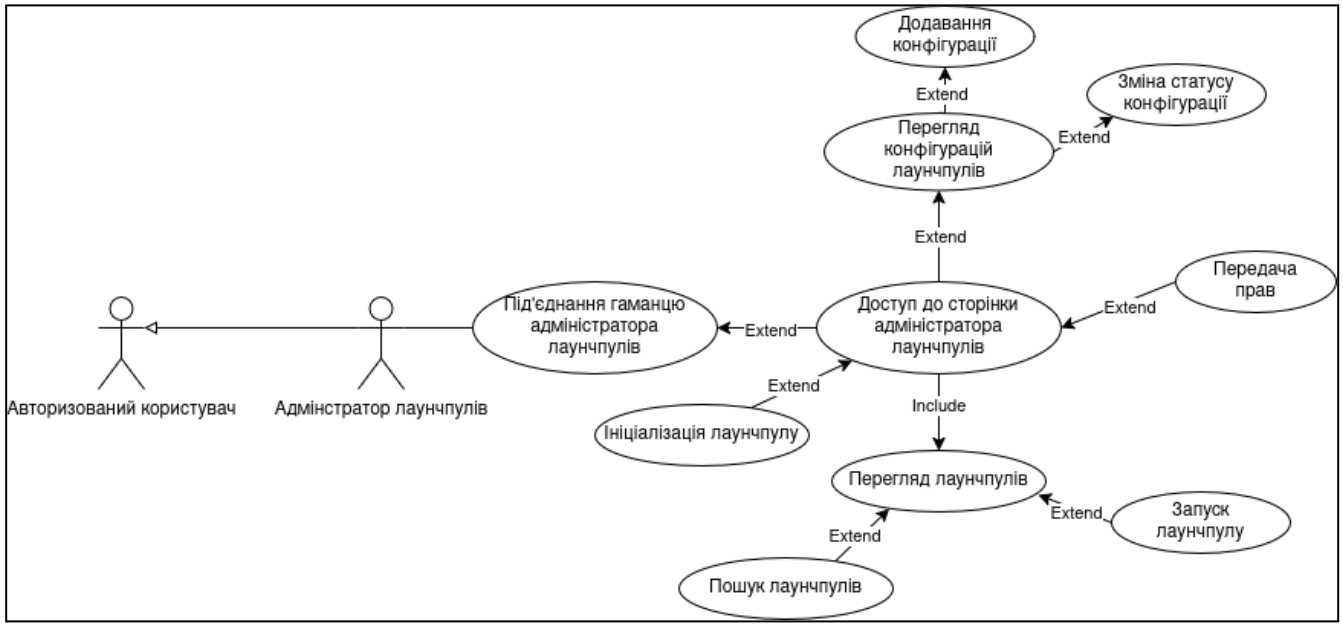

Рисунок 3.3 – Діаграма прецедентів для адміністратора launchпулів (рисунок виконаний самостійно)

Адміністратор пулів ліквідності є підтипом авторизованого користувача, який відповідає за налаштування та контроль параметрів пулів ліквідності на

платформі. Після під’єднання гаманця та автентифікації, такий користувач отримує доступ до адміністративної сторінки, де має змогу переглядати конфігурації пулів ліквідності, а також додавати нові конфігурації та змінювати їх статус. Крім цього, адміністратор має можливість передавати права на адміністрування іншій адресі. Наявність цього типу користувача є критично важливою для централізованого реагування в разі зловживань, помилок або потенційних атак з боку інших адміністраторів.

Відповідні функціональні можливості адміністратора пулів ліквідності подано у вигляді діаграми прецедентів на рисунку 3.4.

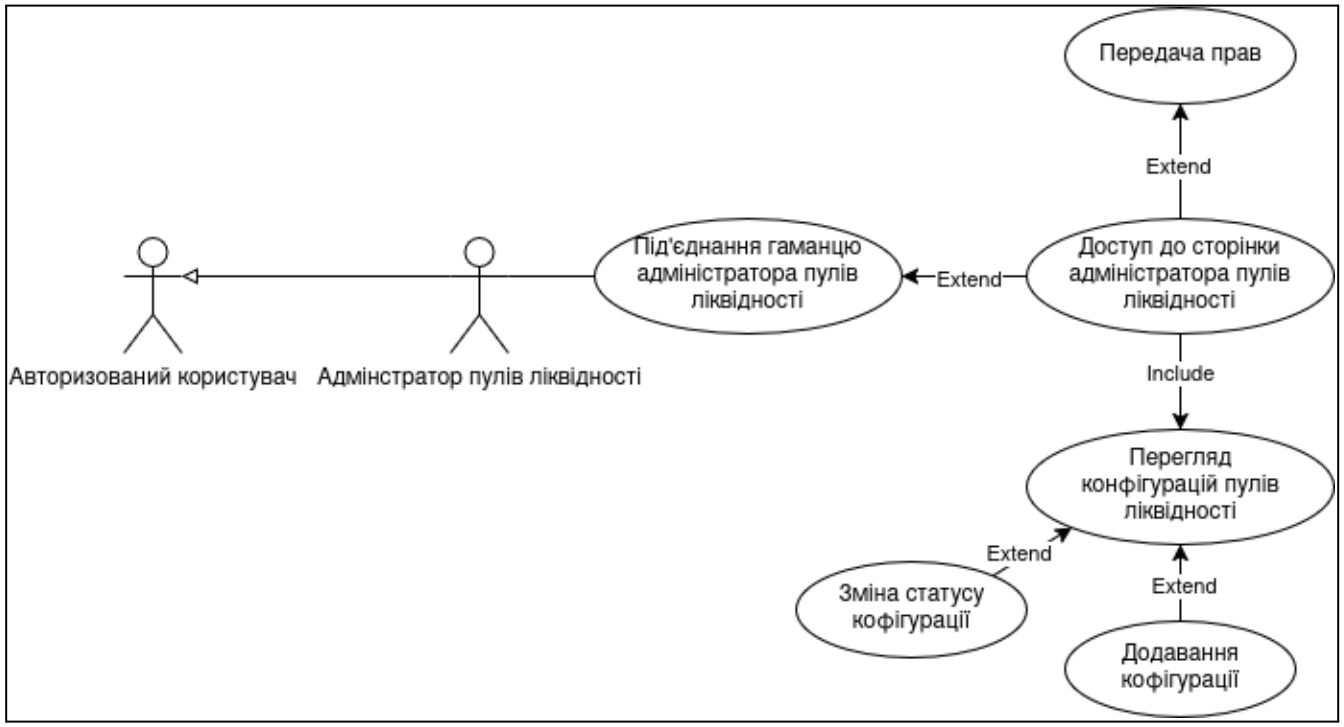

Рисунок 3.4 – Діаграма прецедентів для адміністратора пулів ліквідності (рисунок виконаний самостійно)

Головний адміністратор – це найвищий рівень адміністративного доступу в системі, який успадковує повноваження всіх підтипів адміністраторів. Він має

змогу виконувати всі дії, доступні адміністраторам лаунчпулів і пулів ліквідності, а також здійснювати контроль за їхніми ролями та повноваженнями. Головний адміністратор виконує функції управління всією платформою на рівні технічного налаштування й стратегічного контролю. Цей тип користувача необхідний для централізованого адміністрування децентралізованих компонентів екосистеми.

Відповідні функціональні можливості адміністратора пулів ліквідності подано у вигляді діаграми прецедентів на рисунку 3.5.

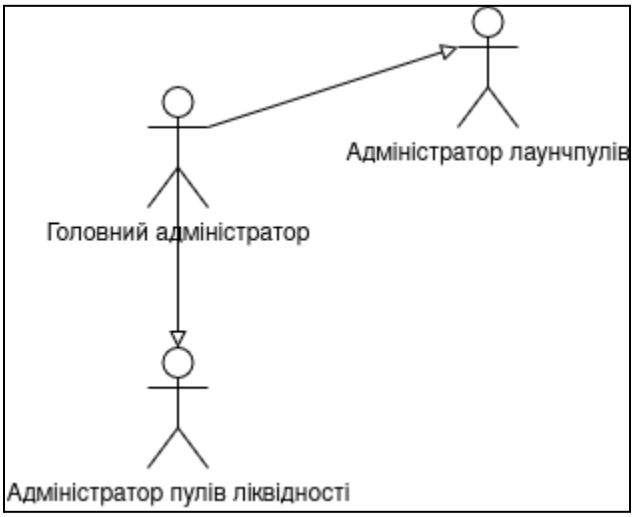

Рисунок 3.5 – Діаграма прецедентів для головного адміністратора (рисунок виконаний самостійно)

У процесі концептуального моделювання предметної області було визначено основні типи користувачів децентралізованої платформи обміну токенів та їх функціональні можливості.

3.2 Архітектура ончейн-частини

Ончейн частина програмної системи реалізована за допомогою двох смартконтрактів, кожен із яких забезпечуватиме окрему бізнес-логіку – один для роботи з пулами ліквідності, інший для роботи з лаунчпулами [11]. Обидва

контракти розгорнуті на блокчейні Solana, що забезпечує високу пропускну здатність, мінімальні затримки та детермінований порядок виконання транзакцій завдяки паралельній моделі обчислень Solana Runtime.

Смартконтракти написані з використанням мови програмування Rust. Ця мова є нативною для платформи Solana, оскільки сам блокчейн реалізовано саме на Rust. Враховуючи складність прямого використання низькорівневих бібліотек Solana, які потребують написання великого обсягу шаблонного коду для перевірки підписів, десеріалізації облікових записів і обробки помилок, прийнято рішення використовувати фреймворк Anchor.

Anchor забезпечить певний рівень абстракції та дозволить зосередитися безпосередньо на розробці бізнес-логіки смартконтрактів. Замість ручної обробки кожної інструкції, Anchor автоматично згенерує необхідний код для перевірки транзакцій, серіалізації/десеріалізації, а також перевірки підписів для доступу до акаунтів.

У контексті розробки на Solana існують дві основні концепції – акаунти та інструкції.

Акаунти є базовою одиницею зберігання даних у Solana. Кожен акаунт має власну адресу, структуру даних, розмір і набір прав доступу. У межах даної платформи акаунти використовуються для збереження інформації про ліквідність, ціну, учасників лаунчпулів, налаштування пулів тощо. З технічної точки зору акаунти відіграють роль ончейн еквіваленту таблиць у реляційних базах даних. Вони гарантують незмінність, безпечність і прозорість зберігання даних, оскільки кожна зміна вмісту є результатом валідованої транзакції. Існує також особливий тип акаунтів – Program Derived Address (PDA), які використовуються для забезпечення контрольованого доступу до структур даних смартконтрактів. PDA – це детерміновано створена адреса акаунту, яка не має приватного ключа та може бути змінена лише відповідною програмою. Саме за допомогою PDA забезпечено

контроль над пулами ліквідності, лаунчпулами, конфігураціями та записами користувачів. Це дозволить виключити можливість несанкціонованих змін з боку сторонніх користувачів.

Інструкції є основним механізмом взаємодії з ончейн логікою програм у Solana. Вони визначають дії, які виконуються програмою в результаті виклику певної транзакції. Кожна транзакція у Solana може містити одну або кілька інструкцій, що виконуються послідовно та атомарно. Інструкції можуть містити вхідні параметри, перелік акаунтів та чи є вони змінними в даній інструкції, а також адресу на програми, яка повинна її виконати. У контексті розроблюваної програми всі ключові дії користувачів – додавання ліквідності, обмін токенів, участь у лаунчпулі – реалізовані через відповідні інструкції смартконтрактів.

У межах ончейн частини реалізовано дві окремі Anchor-програми – програма для роботи з пулами ліквідності та програма для роботи з лаунчпулами.

Програма для роботи з пулами ліквідності забезпечує функціонування децентралізованого автоматизованого маркет-мейкера на основі алгоритму Constant Product Market Maker. У цьому алгоритмі співвідношення двох активів регулюється рівнянням з формули 3.1.

$$x \times y = k, \tag{3.1}$$

де x і y – кількості токенів кожного з типів, що зберігаються у пулі;

k – постійна, що зберігає значення добутку резервів і забезпечує баланс.

Всі пули ліквідності є PDA, що надає права редагування виключно для програми. У структурі кожного пулу передбачено три окремі сховища. Два з них використовуються для зберігання кожного з двох токенів, що формують торгову пару. Третє сховище призначене для LP-токенів, які видаються користувачам при додаванні ліквідності. Токени, з якими працюватимуть пули, відповідають стандарту SPL, при цьому реалізовано підтримку як класичної Token-програми,

так і новітньої Token-2022, яка розширює можливості управління та конфігурації токенів. Взаємодія з токенами здійснюється через CPI-виклики до відповідної програми токенів.

Користувачі зможуть здійснювати обмін токенів у пулі, при якому один токен буде депоновано, а інший – видано відповідно до математичної моделі CPMM. Для забезпечення точних обчислень використано бітову арифметику та тип Q64.128, у якому перші 8 байт представляють цілу частину, а останні 16 – дробову. Це дозволить уникнути втрат при розрахунку коренів квадратних із добутку резервів і співвідношення кількостей токенів.

Усі користувачі, які додаватимуть ліквідність у пул, отримують LP-токени, що відображають їхню частку в загальному обсязі пулу. LP-токени також реалізовані як токени SPL, які можна зберігати на звичайному гаманці. При вилученні ліквідності користувач повертє LP-токени назад до пулу, після чого отримує відповідну кількість кожного з двох токенів, пропорційно своїй частці у резерві.

Комісія за обмін встановлена в конфігурації пулу і розподіляється між провайдерами ліквідності. Для цього комісія з кожного обміну додається до загальної ліквідності. У момент вилучення ліквідності, провайдер отримує й свою частку з частиною накопичених комісій.

Окрему увагу приділено механізму захисту користувача від прослизання – явища, коли реальна ціна обміну відрізняється від очікуваної через зміну співвідношення резервів у процесі виконання транзакції. Для цього користувач при обміні вказує максимально допустиму різницю в токенах, і у разі її перевищення транзакція автоматично скасовується.

Архітектура програми передбачає використання трьох основних ончейн структур даних, які реалізовані у вигляді PDA.

Основним елементом виступє обліковий запис AmmsConfigsManager – унікальна структура, яка відповідє за централізоване створення, зміну та моніторинг об’єктів типу AmmsConfig. Оскільки платформа підтримуватиме різні конфігурації пулів ліквідності, цей менеджер виконує роль адміністративного шлюзу для контролю параметрів новостворених пулів.

Структура AmmsConfig зберігає специфічні параметри конфігурації для груп пулів. До таких параметрів належать значення комісій, які застосовуються під час обміну токенів, а також публічний ключ стороннього гаманця, на який надходить частина комісій, що належить платформі. Одну конфігурацію можна прив’язати до кількох пулів ліквідності, що забезпечить уніфікацію налаштувань.

Кожен пул ліквідності реалізовано як структуру CpmmPool, яка зберігає інформацію про токени, резерви, статус, пов’язані сховища, накопичені комісії та публічний ключ відповідного AmmsConfig, від якого успадковується значення відсотків комісій.

Таким чином, кожна структура AmmsConfigsManager має зв’язок «один до багатьох» з AmmsConfig, а кожна AmmsConfig – зв’язок «один до багатьох» із CpmmPool.

Запропонована модель даних забезпечить розділення обов’язків між рівнем керування, конфігурації та операціями у пулах ліквідності. Це зробить програму розширюваною, модульною та зручною для оновлення без порушення цілісності існуючих пулів.

Відповідну діаграму класів наведено на рисунку 3.6.

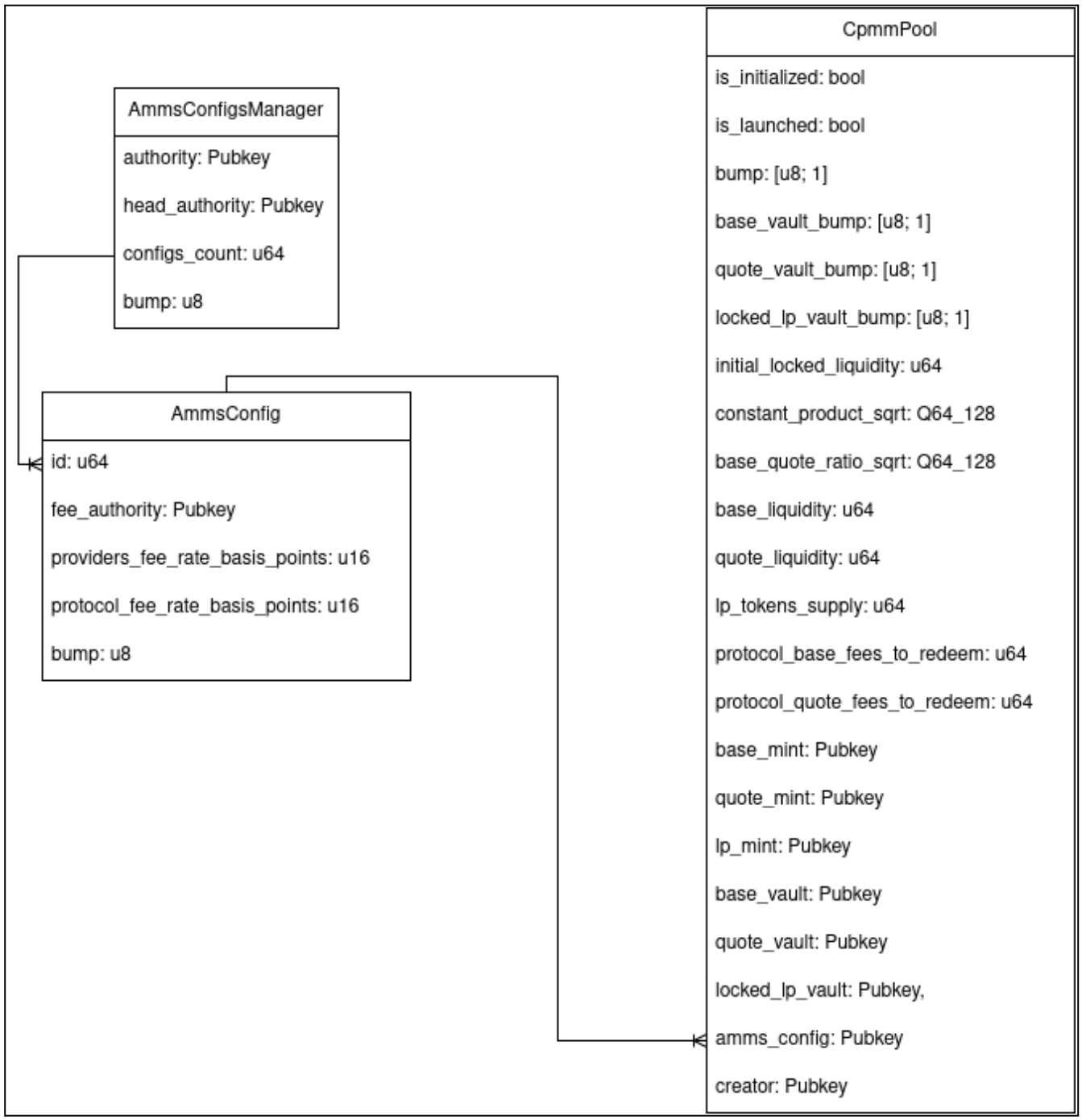

Рисунок 3.6 – Діаграма класів програми пулів ліквідності (рисунок виконаний самостійно)

Для функціонування програми реалізовано набір інструкцій, які охоплюють як дії адміністратора, так і взаємодію звичайного користувача з пулами ліквідності.

Інструкції для клієнта:

- ініціалізація пулу ліквідності та його прив’язка до конфігурації.
- запуск пулу з початковими резервами;
- додавання ліквідності у пул;
- вилучення ліквідності разом із часткою накопичених комісій;
- збір комісій для протоколу;
- здійснення обміну токенів у пулі з урахуванням очікуваного результату та максимально допустимого прослизання.

Інструкції для адміністратора:

- створення керуючої структури для конфігурацій пулів;
- оновлення адміністратора та головного адміністратора керуючої структури;
- створення нової конфігурації;
- зміна статусу конфігурації;
- зміна адреси для отримання частки протокольної комісії.

Програма для роботи з лаунчпулами забезпечує механізм публічного запуску токенів через модель попереднього розподілу із залученням стейкінгу токенів VON.

Механізм роботи лаунчпулів передбачає кілька послідовних етапів. Після створення пулу та його запуску, користувачі зможуть брати участь у ньому шляхом стейкінгу певної кількості токенів VON на визначений у конфігурації термін. Відповідний обсяг токенів переміщено на спеціальне сховище та створено обліковий PDA, який відображає позицію користувача. Цей аккаунт містить інформацію про кількість застейканих токенів, час початку участі, а також обсяг винагороди, на яку має право користувач.

Після завершення терміну активності лаунчпулу, програма не здійснюватиме автоматичний розподіл токенів проекту між учасниками. Натомість, розрахунок частки кожного користувача відбувається безпосередньо під час закриття позиції користувачем. Розрахунок частки виконується на основі загального обсягу стейкінгу та параметрів, визначених у конфігурації пулу. Розподіл токенів проекту здійснюється пропорційно внеску. Залежно від налаштувань, винагорода може бути надана повністю або частково – за механізмом поступового розблокування.

Користувач матиме можливість взяти участь у лаунчпулі лише один раз для кожного гаманця. Після відкриття позиції користувач не зможе її скасувати та отримати застейкані токени до завершення лаунчпулу. Водночас, якщо конфігурація дозволятиме, користувач зможе додати до вже існуючої позиції додаткову кількість токенів, збільшуючи свій внесок.

Архітектура програми передбачає використання чотирьох основних ончейн структур даних, які реалізовані у вигляді PDA.

Основним елементом виступає обліковий запис LaunchpoolsConfigsManager – унікальна структура, яка відповідає за централізоване створення, зміну та моніторинг об’єктів типу LaunchpoolsConfig. Оскільки платформа підтримуватиме різні конфігурації лаунчпулів, цей менеджер виконує роль адміністративного шлюзу для контролю параметрів новостворених пулів.

Структура LaunchpoolsConfig зберігатиме глобальні параметри конфігурації, які будуть успадковувати окремі лаунчпули. Серед таких параметрів: токен, що використовується для стейкінгу, мінімальний та максимальний обсяг участі.

Кожен окремий лаунчпул реалізовано як структуру Launchpool. У ній зберігатимуться такі параметри як адреса токену, обсяг токенів для видачі, час початку та завершення, загальна кількість учасників, загальна кількість застейканих токенів, кількість розподілених токенів, статус активності, а також публічний ключ LaunchpoolsConfig, від якого будуть взяті налаштування.

Для обліку участі користувача в конкретному лаунчпулі використовується структура StakePosition. Це окремий PDA, який створюється для кожного унікального гаманця-учасника в межах одного лаунчпулу. У ньому буде зберігатися обсяг застейканих токенів, мітка часу останнього внеску, накопичена винагорода, статус отримання винагороди.

Для відзтеження статусів Launchpool та StakePosition використані відповідні перерахування – LaunchpoolStatus та PositionStatus.

Таким чином, кожна структура LaunchpoolsConfigsManager матиме зв’язок «один до багатьох» з LaunchpoolsConfig, а кожна LaunchpoolsConfig – зв’язок «один до багатьох» із Launchpool.

Запропонована модель даних забезпечить розділення обов’язків між рівнем керування, конфігурації та операціями у лаунчпулах. Це зробить програму розширюваною, модульною та зручною для оновлення без порушення цілісності існуючих пулів.

Відповідну ER-модель даних програми наведено на рисунку 3.7.

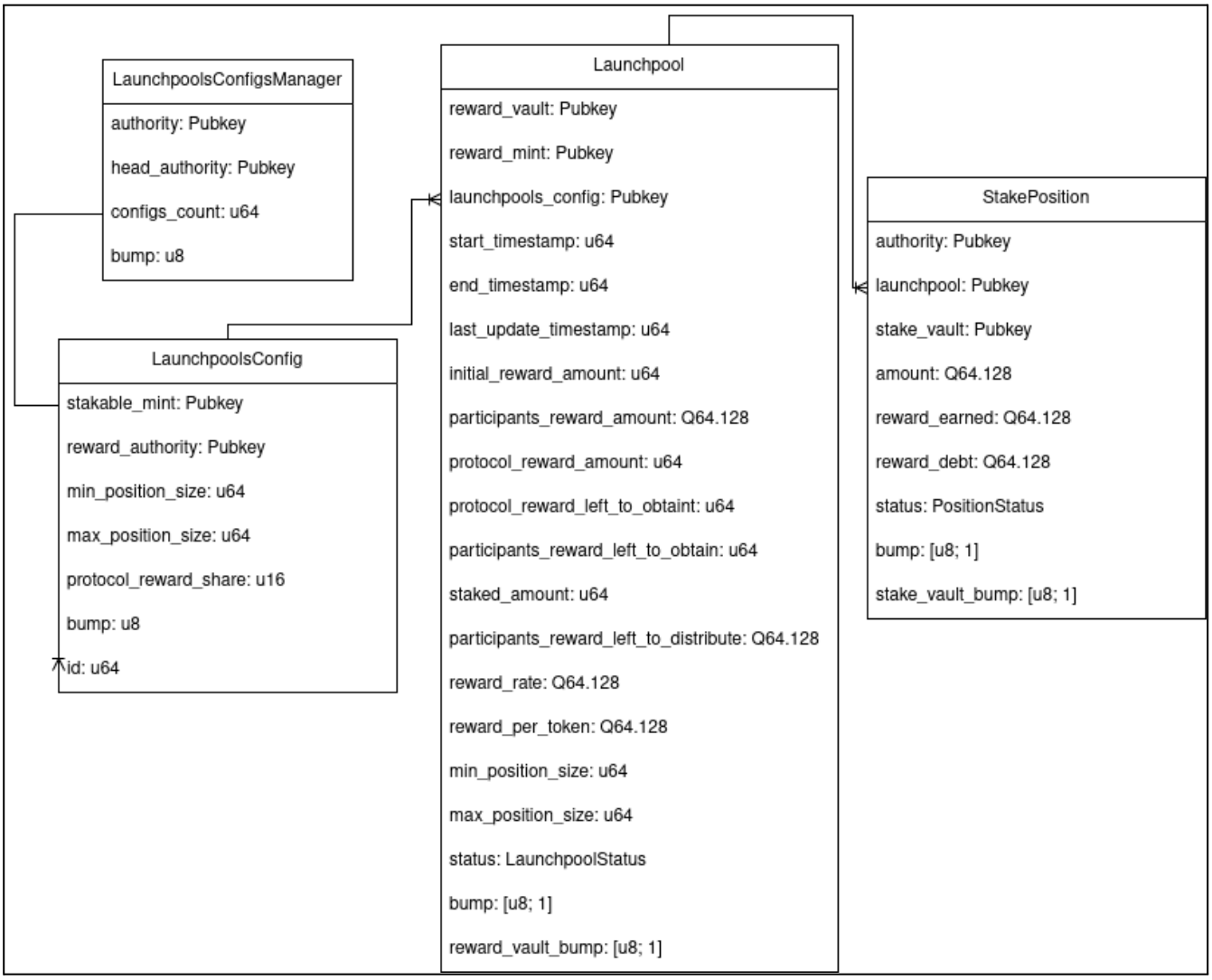

Рисунок 3.7 – Діаграма класів програми лаунчпулів (рисунок виконаний самостійно)

Для функціонування програми реалізовано набір інструкцій, які охоплюють як дії адміністратора, так і взаємодію звичайного користувача з лаунчпулами.

Інструкції для клієнта:

- відкриття позиції;
- збільшення позиції;
- закриття позиції.

Інструкції для адміністратора:

- ініціалізація лаунчпулу та його прив’язка до конфігурації;
- запуск лаунчпулу;
- створення керуючої структури для конфігурацій лаунчпулів;
- оновлення адміністратора та головного адміністратора керуючої структури;
- створення нової конфігурації;
- зміна адреси для отримання протокольної частки токенів.

3.3 Архітектура офчейн-частини

Офчейн частина програмної системи Vondex включатиме низку критично важливих компонентів, які забезпечуватимуть взаємодію між користувачем та ончейн-логікою смартконтрактів.

Першим компонентом офчейн-частини програмної системи Vondex стане окремий індексатор (або Listener) реалізований мовою програмування Rust, який буде відповідати за безперервне прослуховування транзакцій ончейн-програм. Сервер використовуватиме стандартні механізми WebSocket-підключення до RPC-інтерфейсу – зокрема метод programSubscribe. Завдяки цьому система зможе бачити всі події в обох програмах, що виникають у результаті виконання транзакцій: зміни стану акаунтів користувачів, оновлення пулів ліквідності, відкриття та закриття позицій у лаунчпулі, зміну конфігурацій та адміністративних параметрів тощо.

Кожна отримана подія проходитиме попередню фільтрацію на основі унікальних ідентифікаторів програм і структур акаунтів, після чого оброблятиметься та десеріалізуватиметься відповідно до описів IDL-файлів, що використовуються Anchor-фреймворком. У результаті буде сформовано компактну, структуровану модель події, яка відображатиме ключові атрибути транзакції: час,

тип, джерело, пов’язані акаунти, обсяг активів, статус виконання, ідентифікатор користувача тощо.

Призначенням цього модуля буде побудова власного механізму індексації подій, що дозволить уникнути залежності від сторонніх індексаторів або централізованих API. Усі оброблені події передаватимуться до ScyllaDB, де будуть зберігатися у вигляді хронологічного журналу дій, доступного для подальшого аналізу, агрегації та відображення на клієнтській частині.

Другим ключовим компонентом офчейн-частини стане система зберігання на базі ScyllaDB – високопродуктивної розподіленої бази даних, яка буде використовуватися як швидкісний кеш для збереження подій, отриманих у результаті прослуховування транзакцій Anchor-програм. Завдяки внутрішній архітектурі, побудованій на основі моделі Apache Cassandra, ScyllaDB забезпечуватиме лінійне горизонтальне масштабування, високу пропускну здатність і мінімальну затримку доступу навіть при значному обсязі даних.

Основним призначенням цього модуля буде зберігання подій, що надходитимуть від індексатор для відображення статистики та історії транзакцій. Подія, що буде згенерована в результаті обробки транзакції Anchor-програмою, зберігатиметься у вигляді окремого запису.

ScyllaDB дозволятиме ефективно індексувати дані за кількома ключами – зокрема за публічними ключами користувачів, типом події, програмою-джерелом, часом або статусом виконання. Це забезпечуватиме миттєвий доступ до інформації, необхідних для відображення поточного стану системи або побудови історичних графіків. Наприклад, клієнт зможе отримати повну історію торгів в певному пулі ліквідності без звернення до блокчейну.

ScyllaDB стане основою для забезпечення швидкого доступу до історичних даних суттєво знижуючи навантаження на Solana RPC.

Третім компонентом офчейн-частини програмної системи Vondex стане високопродуктивний бекенд-сервер, що буде реалізований мовою програмування Rust із використанням асинхронного фреймворку Axum. Цей сервер виконуватиме роль обчислювального вузла, який відповідатиме за побудову транзакцій, взаємодію з RPC-інтерфейсом Solana, обробку запитів клієнтів, а також трансляцію подій у реальному часі через WebSocket-канал на фронтенд-застосунок.

Основною функцією сервера стане підготовка транзакцій для виконання дій у межах Anchor-програм, розгорнутих на блокчейні Solana. Після отримання відповідного запиту з клієнтської частини через REST API, сервер здійснюватиме обчислення параметрів транзакції: визначатиме список обов'язкових акаунтів, ідентифікуватиме потрібні інструкції, формуватиме вхідні параметри та створюватиме серіалізований об'єкт транзакції у форматі, сумісному з бібліотекою @solana/web3.js. Згенерований об'єкт повертатиметься клієнту для підписання безпосередньо в криптогаманці. Таким чином, бекенд не зберігатиме приватних ключів, не здійснюватиме підписання транзакцій і не втручатиметься в процеси авторизації.

Додатково сервер забезпечуватиме доступ до актуального стану ончейн-структур через прямі RPC-запити. У разі, якщо користувач відкриватиме сторінку певного пулу ліквідності або лаунчпулу, сервер звертатиметься до RPC-вузла Solana, зчитуватиме вміст відповідних акаунтів, десеріалізуватиме їх та повертатиме на фронтенд у зручному форматі. Весь процес виконуватиметься асинхронно з використанням неблокуючих механізмів, що дозволить обробляти велику кількість паралельних запитів із низькою затримкою.

Окрім REST-інтерфейсу, сервер також підтримуватиме власний WebSocket API, через який транслюватиме події, що були попередньо збережені у ScyllaDB. Бекенд отримуватиме відповідні оновлення з бази та передаватиме їх на веб-клієнт

у реальному часі. Такий підхід дозволить користувачам оперативно отримувати повідомлення про нові транзакції, зміни в пулах ліквідності, тощо. З метою зменшення навантаження на інфраструктуру ScyllaDB і RPC-вузли, сервер також матиме інмеморі-кеш. Це дозволить значно скоротити кількість прямих звернень до бази даних і блокчейну.

Таким чином, бекенд-сервер забезпечуватиме інтерактивну взаємодію з клієнтом, генерацію транзакцій, доступ до ончейн-даних і трансляцію подій, що зробить його ключовим вузлом між користувачем, системою зберігання та блокчейн-програмами.

Останнім компонентом стане клієнтська частина програмної системи Vondex, яка буде реалізована у вигляді односторінкового вебзастосунку з використанням фреймворку Next.js, що поєднує можливості клієнтського та серверного рендерингу, забезпечуючи високу продуктивність, гнучкість та інтерактивність. Усі динамічні дані, що відображатимуться у клієнтському інтерфейсі, надходитимуть через REST- та WebSocket-інтерфейси від бекенд-сервера, а транзакції підписуватимуться користувачем локально через криптогаманець.

Інтерфейс буде відповідати акторам та їх можливостям, які були визначені в рамках концептуального моделювання. Саме клієнтська частина реалізовуватиме підтримку декількох ролей користувачів, кожна з яких матиме власний рівень доступу до функціональності платформи.

Загалом клієнтська частина виступатиме інтерактивною оболонкою платформи, яка відображатиме поточний стан ончейн-екосистеми, забезпечуватиме ініціацію транзакцій і виконуватиме роль головного інструмента взаємодії користувача з програмною системою Vondex.

Підсумовуючи, офчейн-архітектура програмної системи Vondex є мікросервісною, де кожен сервіс виконує чітко окреслену роль: WebSocket-сервер

забезпечує безперервне отримання та обробку подій з блокчейну, система зберігання ScyllaDB виконує функцію швидкісного кеша та індексатору, а бекенд-сервер відповідає за формування транзакцій, обробку запитів клієнтів і трансляцію даних у реальному часі. Фронтенд частина виступає інтерфейсом для взаємодії з системою. Така архітектура забезпечуватиме масштабованість та швидкість.

3.4 Побудова діаграми компонентів та розгортання

На основі описаних ончейн- та офчейн- частин програмної системи Vondex було побудовано діаграми компонентів та розгортання (див. рисунок 3.8 та 3.9).

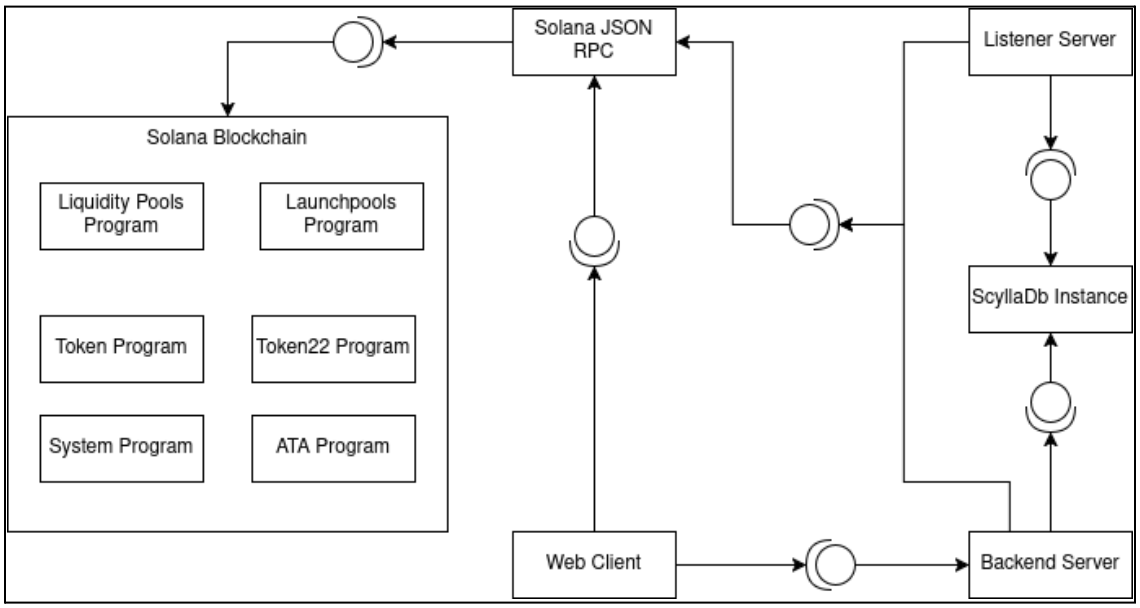

Рисунок 3.8 – Діаграма компонентів (рисунок виконаний самостійно)

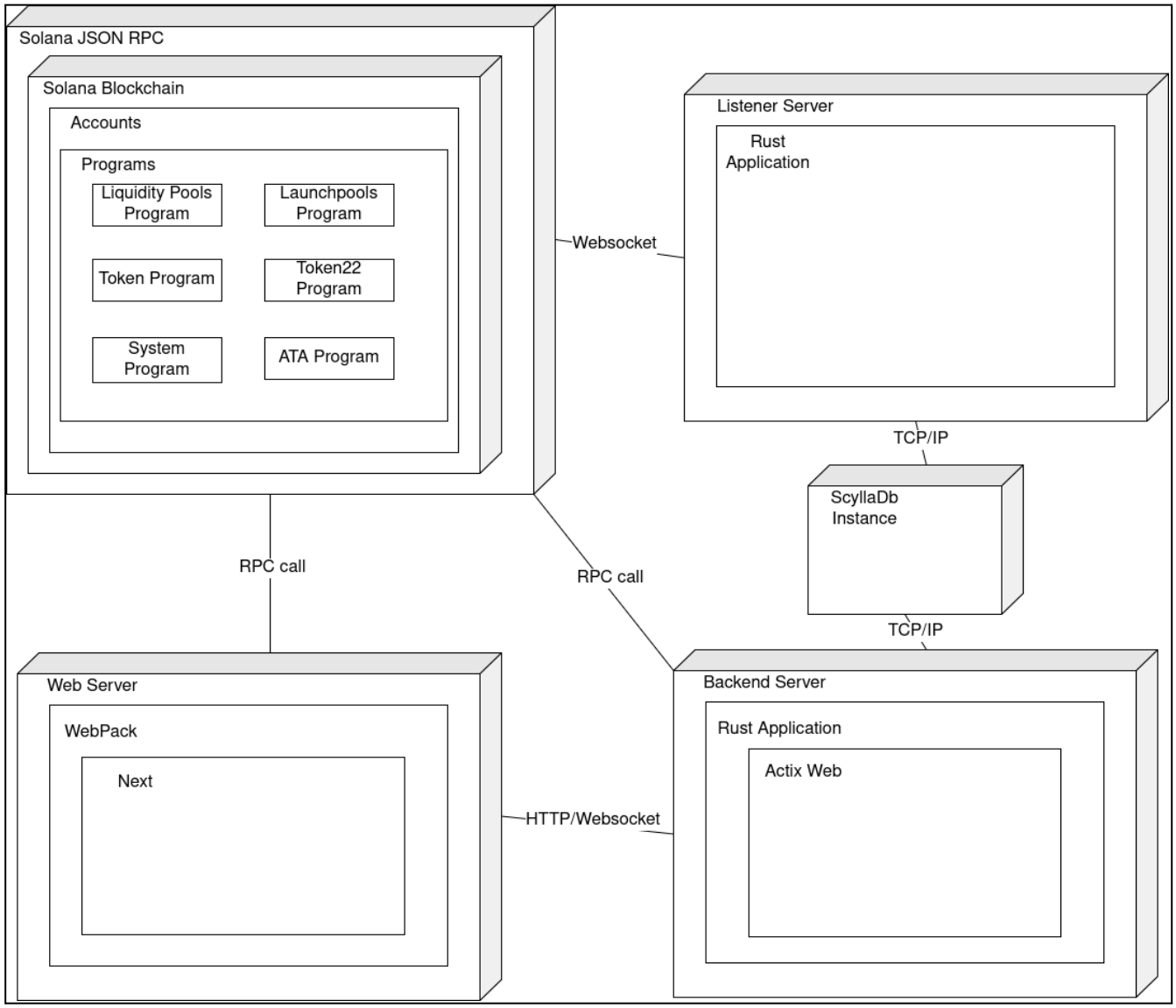

Рисунок 3.9 – Діаграма розгортання (рисунок виконаний самостійно)

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Бібліотека утиліт

Оскільки розроблювана система складається з декількох частин реалізованих мовою програмування Rust, було прийнято рішення перенести код, який використовується кількома компонентами в бібліотеку utilities.

Першим модулем є модуль math, що відповідає за математичні типи та операції, необхідні для обробки фінансових даних. З огляду на специфіку предметної області, математичні алгоритми системи мають гарантувати високу точність та стабільність для всіх можливих комбінацій вхідних параметрів, включно з граничними значеннями.

Одним із ключових викликів є неможливість використання дробових чисел при реалізації смартконтрактів, адже середовище виконання вимагає повної детермінованості результатів. У блокчейні кожен смартконтракт виконується на всіх валідаторах, і всі вони мають отримати абсолютно однаковий результат. Використання чисел з плаваючою комою унеможливлює це, оскільки результат таких операцій залежить від особливостей апаратного забезпечення та середовища виконання.

Для вирішення цих проблем було прийнято рішення реалізувати тип для числа в форматі Q64.128 – 64 біта для цілої беззнакової частини та 128 біт для дробової [12]. Для його реалізації використовується Rust-бібліотека uint – набір макросів для створення цілих чисел довільної розрядності. У реалізації застосовано два типи з цієї бібліотеки: U192, який зберігає значення Q64.128, та U384, який використовується як допоміжний при виконанні операцій множення з метою уникнення переповнення.

Хоча типи U192 та U384 підтримують базові арифметичні операції за замовчуванням, їхнє пряме використання в рамках типу Q64_128 не забезпечує

коректну обробку чисел з фіксованою дробовою частиною. Формат Q64.128 передбачає представлення числа як беззнакового 192-бітного значення, де старші 64 біти відповідають цілій частині, а молодші 128 – дробовій. Тому для коректної роботи з цим форматом були реалізовані спеціальні методи перетворення та представлення значень із використанням побітових зсувів та маскувань. Такий підхід гарантує як точність, так і високу продуктивність обчислень.

З оглядом на майбутні потреби смартконтрактів для Q64_128 було реалізовано наступні арифметичні, математичні та конвертаційні операції: додавання, віднімання, множення, ділення, округлення до u64, перевірка на нуль та одиницю, перетворення з/до u64, u128, f64, U384, побітове отримання цілої та дробової частини, серіалізація в big-endian та little-endian формати, обчислення модуля різниці, квадрат, квадратний корінь, перевірені та насичені варіанти множення і ділення, а також спеціальний метод – корінь із частки (див. Додаток В.1).

Таким чином, реалізація типу Q64_128 забезпечує точне, стабільне та детерміноване обчислення з фіксованою дробовою точкою, що є критично важливим для фінансових операцій у смартконтрактах. Завдяки використанню розширених цілих типів та власноруч реалізованим арифметичним і математичним методам, система здатна ефективно обробляти широкий спектр вхідних даних, включно з граничними значеннями, без втрати точності чи ризику недетермінованої поведінки на різних вузлах мережі.

Типи Q64_128, U192 та U384 доступні в модулі math бібліотеки utilities там можуть бути легко імпортовані в будь який компонент за потреби.

Другим модулем є модуль solana, який відповідає за формування інструкцій для системних програм блокчейну Solana. Його було винесено в окрему частину бібліотеки, оскільки обидва смартконтракти системи виконують однакові операції з токенами – такі як трансфери, створення або спалення токенів, перевірка

дозволів чи ініціалізація акаунтів – саме через виклики системних програм Token Program, Token 2022 Program, Associated Token Program та System Program [13].

Оскільки модулі math та solana використовуються в різних контекстах – ончейн та офчейн відповідно – було реалізовано механізм умовної компіляції. Це дозволяє явно вказувати, які саме частини бібліотеки мають бути включені під час збірки. Такий підхід унеможливлює ситуації, коли, наприклад, офчейн-код включає модуль solana, що в даному контексті не використовується й лише збільшує розмір збірки та створює зайві залежності.

Таким чином, бібліотека utilities забезпечує повторне використання критично важливого функціоналу між різними компонентами системи, зменшує дублікацію коду та підвищує узгодженість логіки між ончейн- та офчейн-частинами. Її модульна структура з підтримкою умовної компіляції дозволяє гнучко інтегрувати лише необхідні частини в кожен з наступних компонентів – смартконтракт пулів ліквідності, смартконтракт лаунчпулів, індексатор транзакцій та бекенд-сервер – що сприяє більшій ефективності розробки та супроводу системи.

4.2 Смартконтракти

Для реалізації механізму пулів ліквідності, які забезпечують обмін токенами та розподіл ліквідності між користувачами, та механізму лаунчпулів, які забезпечують стейкінг токенів та розподіл нагород між користувачами, було використано Anchor – Rust-фреймворк для розробки смартконтрактів у блокчейні Solana [14]. Anchor значно спрощує процес створення смартконтрактів, дозволяючи зосередитись на бізнес-логіці. Завдяки системі макросів та типів мови Rust, фреймворк автоматично обробляє перевірки підписів, серіалізацію/десеріалізацію даних та управління акаунтами, що значно знижує ризик помилок і пришвидшує розробку.

Реалізовані смартконтракти складаються з інструкцій, кожна з яких складається функції-обробника, контексту з відповідними акаунтами та лог-події. Функція-обробник містить основну бізнес-логіку інструкції. Вона отримує вхідні параметри та контекст з акаунтам, виконує перевірки, оновлює стан акаунтів або викликає інші програми. Контекст – це спеціальна структура, яка визначає перелік облікових записів, необхідних для виконання інструкції. У контексті вказується, які акаунти є обов'язковими, які мають бути підписантами, дані яких можуть бути змінені, а також інші перевірки. Лог-подія – це структура, яка фіксує результат виконання інструкції у вигляді запису в логах транзакції. Вона дозволяє відстежувати ключові події, такі як створення пулу, додавання ліквідності або закриття позиції. Події реєструються через макрос emit!, що спрощує офчейн-моніторинг та індексацію даних.

Для зберігання стану використано спеціалізовані структури даних для акаунтів, схематично зображені на діаграмах класів (рисунки 3.6 та 3.7, розділ 3.2). Усі математичні обчислення реалізовано із застосуванням типу Q64_128, що забезпечує високу точність при роботі з дробовими значеннями. Взаємодія з системними програмами реалізована з використанням функцій з модуля solana бібліотеки utilities.

Для смартконтракту пулів ліквідності було реалізовано 13 інструкцій: ініціалізація менеджера конфігурацій пулів, оновлення адміністратора менеджера, оновлення головного адміністратора менеджера, ініціалізація конфігурації пулу, оновлення адреси збору комісій конфігурації, оновлення ставки комісії для провайдерів конфігурації, оновлення протокольної ставки комісії конфігурації, ініціалізація пулу ліквідності, запуск пулу ліквідності, додавання ліквідності до пулу, вилучення ліквідності з пулу, обмін токенів в пулі, збір комісій з пулу (див. Додаток В.2).

Для смартконтракту launчпулів було реалізовано 14 інструкцій: ініціалізація конфігурації launчпулу, ініціалізація менеджера конфігурацій launчпулів, оновлення тривалості конфігурації, оновлення мінімального та максимального розміру позиції в конфігурації, оновлення протокольної частки винагороди в конфігурації, оновлення адреси отримувача винагород в конфігурації, оновлення адміністратора менеджера, оновлення головного адміністратора менеджера, ініціалізація launчпулу, запуск launчпулу, відкриття стейк-позиції, збільшення стейк-позиції, закриття стейк-позиції, збір протокольної винагороди (див.Додаток В.3).

4.3 Клієнти смартконтрактів

Оскільки смартконтракти розгорнуті в блокчейні Solana, маючи доступ до їхнього Anchor IDL взаємодія з ними можлива з використанням інструментів будь-якої клієнтської бібліотеки Solana – зокрема, через Solana CLI, JavaScript-бібліотеки @solana/web3.js або Rust-клієнт solana-sdk. Anchor IDL це файл в JSON форматі, в якому описаний інтерфейс смартконтракту, який зазвичай публікується у відкритий доступ.

Хоча така взаємодія є можливою вона не є зручною і потребує великого обсягу ручного, шаблонного коду, особливо при роботі на відповідних мовах програмування. Для спрощення інтеграції було використано інструмент Codama [15], який дозволяє транслювати Anchor IDL у більш зручний формат – Codama IDL. Codama IDL – це набір файлів мовою TypeScript або Rust, що описують усі інструкції, облікові структури та типи смартконтракту у вигляді готового до використання коду. Ці файли можна вставити безпосередньо у проєкт, написаний відповідною мовою, що значно спрощує взаємодію з контрактами, усуває необхідність ручної серіалізації та ймовірність помилок при формуванні транзакцій.

Такий спосіб взаємодії значно зручніший порівняно з попереднім, однак він має обмеження у масштабуванні. Модулі, згенеровані Codama, необхідно вручну вставляти в кожен проект, де планується використання смартконтрактів, а також окремо завантажувати всі залежності. Це ускладнює підтримку, повторне використання та автоматизацію процесу розробки. Для вирішення цієї проблеми було прийнято рішення трансформувати Codama IDL у повноцінну бібліотеку під час генерації, яка зберігає весь згенерований код, залежності та інтерфейси, доступні для підключення через звичайний пакетний менеджер: Cargo для Rust та npm для Typescript.

Для того щоб трансформувати Codama IDL у повноцінні клієнтські бібліотеки, стандартний процес генерації було розширено та адаптовано до потреб проєкту (див.Додаток В.4). Замість того, щоб вручну копіювати згенеровані файли в кожен застосунок, скрипт автоматично формує повноцінну структуру бібліотеки – окремо для Rust і JavaScript. У процесі генерації до кожної бібліотеки додаються службові метадані, визначаються залежності та налаштовується структура, яка дозволяє зручно підключати бібліотеку до будь-якого іншого проекту. Таким чином, клієнтський код стає багаторазово придатним до використання, легко інтегрується в будь-яке середовище та масштабується без додаткових ручних дій.

Таким чином, згенеровані бібліотеку забезпечують швидку, безпечну та масштабовану взаємодію з смартконтрактами, а їхня типобезпечність знижує ризик помилок під час розробки та тестування. Це суттєво спрощує інтеграцію контрактів у бекенд-сервер та індексатор транзакцій.

4.4 Індексатор транзакцій та сховище подій

Як було зазначено раніше, взаємодія зі смартконтрактом може здійснюватися з будь-якого застосунку та будь-якою третьою стороною. Це ускладнює збереження історії подій (наприклад, історії торгів у пулі ліквідності) на відміну

від класичних Web2-систем, де всі зміни централізовано фіксуються серверною частиною. Для вирішення цієї проблеми було розроблено офчейн-додаток для індексації транзакцій, у яких беруть участь програми пулів ліквідності або лаунчпулів. Індексація здійснюється шляхом підписки на WebSocket-потік від Solana RPC, що дозволяє обробляти події в реальному часі.

Застосунок реалізовано мовою Rust з використанням асинхронного фреймворку Tokio, що забезпечує високу продуктивність і масштабованість. Обробка лог-подій, які генеруються кожною інструкцією смартконтрактів, здійснюється за допомогою типів, згенерованих з Codama IDL. Для зручності десеріалізації з байтового формату було реалізовано декларативний макрос, який автоматично генерує перерахування всіх типів подій, що використовується для їх класифікації та збереження до сховища подій (див. Додаток В.5).

Сховище подій реалізовано у вигляді окремого контейнера з базою даних ScyllaDB – високопродуктивної розподіленої NoSQL-системи, спеціально оптимізованої для обробки великої кількості записів із мінімальними затримками. Ця технологія була обрана завдяки її швидкодії, низькій латентності та підтримці горизонтального масштабування з реплікацією, що є критично важливим для системи реального часу, яка індексує події у блокчейні.

У базі даних ScyllaDB створено два окремих keyspace – по одному для кожного смартконтракту. Один відповідає за події, пов’язані з пулами ліквідності, інший за події лаунчпулів (див. Додаток В.6). Такий поділ дозволяє логічно ізолювати дані, спростити обробку запитів та забезпечити незалежну масштабованість кожного напряму. Всередині кожного keyspace дані організовані за типами подій, що дозволяє ефективно реалізувати пошук, фільтрацію та аналітику на стороні клієнта.

Запис до сховища здійснюється виключно індексатором транзакцій, що гарантує цілісність даних та запобігає сторонньому втручанню. Сховище виконує

дві основні функції. По-перше, воно акумулює історичні події, зокрема обміни токенів у пулах ліквідності, що дає змогу легко будувати історичну аналітику, по-друге, воно слугує офчейн-кешем для часто використовуваних, але статичних або рідко змінюваних даних із блокчейну (наприклад, налаштувань конфігурацій або описів пулів). Це знижує навантаження на RPC і значно пришвидшує офчейн-взаємодію з контрактами.

Сховище використовується виключно для потреб клієнтської частини системи – для зручності, швидкодії та побудови інтерфейсів. Воно не порушує принципи децентралізації, оскільки не є джерелом істини, не приймає рішень і не містить жодних персональних даних. Усі дані, що зберігаються в ньому, вже існують на блокчейні та можуть бути підтверджені ончейн. Якщо користувач не довіряє клієнтському додатку або бекенду, він завжди має можливість взаємодіяти зі смартконтрактом напряму, минаючи офчейн-компоненти й покладаючись виключно на блокчейн Solana.

4.5 Бекенд-сервер

Бекенд-сервер, реалізований за допомогою Axum і Tokio, є асинхронним вебзастосунком, який обробляє HTTP-запити та підтримує WebSocket-з’єднання. На відміну від класичних серверів, він не зберігає дані самостійно, а працює як проміжна ланка між клієнтом, кешем, сховищем подій та блокчейном. Його завдання полягає в отриманні актуальної інформації з доступних джерел і трансформації її у зручний для клієнта формат.

Окрім обробки запитів на отримання даних, сервер відповідає за формування транзакцій для взаємодії зі смартконтрактами. У процесі генерації транзакції сервер використовує інформацію з інмеморі кешу, ScyllaDB та RPC-блокчейну, оскільки для побудови валідної інструкції потрібні дані про поточний стан акаунтів (див. Додаток В.7). Сформовані транзакції можуть

включати не одну, а кілька інструкцій, що об’єднуються в межах однієї операції для забезпечення атомарності та зменшення кількості підтверджень. Саме через складність формування вони були винесені на сторону бекенду, тоді як підпис і відправка залишаються на стороні клієнта.

Такий підхід забезпечує розділення відповідальності між клієнтом і сервером, зберігаючи децентралізований характер взаємодії. Формування транзакцій на бекенді дозволяє уникнути складної логіки на клієнтській стороні та прискорює побудову складних транзакцій. Водночас підпис та відправка транзакції залишаються повністю на боці користувача, що гарантує безпеку та контроль над власними коштами.

4.6 Веб-клієнт

Веб-клієнт реалізовано за допомогою фреймворку Next.js, який було обрано завдяки його високій продуктивності, підтримці серверного рендерингу та широкій екосистемі, що робить його одним із найкращих рішень для створення інтерфейсів Web3-застосунків на платформі Solana. Для взаємодії з блокчейном використано бібліотеку @solana/web3.js та пакет @solana/wallet-adapter, який забезпечує зручне підключення гаманців користувача. На поточному етапі реалізовано підтримку гаманця Phantom. Підключення до гаманця та RPC-інфраструктури реалізовано через власноруч створені провайдери та React-контексти, що забезпечують розділення логіки, повторне використання компонентів та гнучкість при масштабуванні клієнтської частини (див. Додаток В.8).

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Модульне тестування математичних операцій

Для тестування математичних операцій, реалізованих у методах типу Q64_128, було обрано підхід модульного тестування з використанням його підтипу – фазз-тестів. Такий підхід дозволяє не лише перевірити правильність обчислень на заздалегідь заданих даних, а й виявити помилки при обробці широкого діапазону випадкових значень, зокрема граничних і некоректних. Для проведення тестування використано стандартний інструментарій пакету Cargo, що входить до екосистеми мови програмування Rust, та бібліотека proptest для fuzz-тестування.

Повний чек-лист тестових випадків для типу Q64_128 наведено в додатку Г в таблиці Г.1.

На основі сформованого чек-листа було реалізовано автоматизовані модульні та фазз-тести. Для кожного пункту списку було створено відповідні тест-кейси з результатами тестування. Приклади отриманих тест-кейсів наведено в таблицях 5.1 та 5.2.

Таблиця 5.1 – Тест-кейс перевірки серіалізації/десеріалізації типу Q64_128

Інформація про тестовий випадок	
Ідентифікатор тестового випадку	Q64_128-02
Власник тесту	Міленний І. А.
Місцезнаходження тесту (шлях)	./utilities/math/q64_128.rs
Дата останнього перегляду	28.05.2025
Технічна вимога, що тестується	Q64_128-ANCHOR
Конфігурація засобів тестування	cargo test
Взаємозалежність тестових випадків	Проходження тесту Q64_128-01

Продовження Таблиці 5.1

Мета тесту		Перевірити коректність серіалізації/десеріалізації типу Q64_128	
Методика тестування			
Налаштування прогону тесту	cargo test		N/A *
Крок	Дія	Очікуваний результат	Відмітка (V)*
1	Створити екземпляр Q64_128 з конкретним числом	Об’єкт Q64_128 з коректним значенням	(V)
2	Серіалізувати через AnchorSerialize	Вектор байт	(V)
3	Десеріалізувати через AnchorDeserialize	Об’єкт збігається з оригіналом	(V)
4	Конвертувати у LE/BE байти та повторити десеріалізацію	Об’єкт збігається з оригіналом	(V)
Очистка після прогону тесту	Не проводиться		
Результати тесту			
Тестувальник: Міленний І. А.	Дата прогону тесту: 28.05.2025	Результат тесту (P/F/B)*: ПРОЙДЕНО (P)	

Таблиця 5.2 – Тест-кейс фазз-перевірки операції ділення з перевіркою

Інформація про тестовий випадок			
Ідентифікатор тестового випадку	Q64_128-24		
Власник тесту	Міленний І. А.		
Місцезнаходження тесту (шлях)	./utilities/math/q64_128.rs		
Дата останнього перегляду	28.05.2025		
Технічна вимога, що тестується	Q64_128-CHDIV		
Конфігурація засобів тестування	cargo test + proptest (100_000 випадкових прогонів)		
Взаємозалежність тестових випадків	Проходження тесту Q64_128-16		
Мета тесту	Перевірити роботу функції `checked_div` для випадкових значень, включаючи 0		
Методика тестування			
Налаштування прогону тесту	cargo test		N/A *
Крок	Дія	Очікуваний результат	Відмітка (V)*
1	Згенерувати пару випадкових значень Q64_128	Значення згенеровані без помилок	(V)
2	Виконати операцію `checked_div(q1, q2)`	Якщо q2 = 0 → результат = None	(V)

Продовження Таблиці 5.2

3	Порівняння результату операції з очікуваним результатом	Результат дорівнює, або відрізняється на допустиму похибку	(V)
Очистка після прогону тесту	Не проводиться		
Результати тесту			
Тестувальник: Міленний І. А.	Дата прогону тесту: 28.05.2025	Результат тесту (P/F/B)*: ПРОЙДЕНО (P)	

Результати проходження всіх тестів наведені на рисунку 5.1.

```
test math::q64_128::tests::unit_tests::square_and_sqrt::test_square_fractional_number ... ok
test math::q64_128::tests::unit_tests::square_and_sqrt::test_square_large_fractional_number ... ok
test math::q64_128::tests::unit_tests::edge_cases::test_extreme_values ... ok
test math::q64_128::tests::unit_tests::square_and_sqrt::test_square_small_fraction ... ok
test math::q64_128::tests::unit_tests::square_and_sqrt::test_square_operations ... ok
test math::q64_128::tests::fuzz_tests::test_invalid_f64_conversion ... ok
test math::q64_128::tests::fuzz_tests::test_types_conversion_round_trip ... ok
test math::q64_128::tests::fuzz_tests::test_u128_sqrt_and_square_round_trip ... ok
test math::q64_128::tests::fuzz_tests::test_q64_128_sqrt_and_square_round_trip ... ok
test math::q64_128::tests::fuzz_tests::test_checked_add ... ok
test math::q64_128::tests::fuzz_tests::test_checked_mul ... ok
test math::q64_128::tests::fuzz_tests::test_checked_div ... ok
test math::q64_128::tests::fuzz_tests::test_checked_sub ... ok
test math::q64_128::tests::fuzz_tests::test_abs_diff ... ok
test math::q64_128::tests::fuzz_tests::test_q64_128_checked_div_sqrt_and_square_round_trip ... ok

test result: ok. 40 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 31.74s

   Doc-tests utilities

running 0 tests
```

Рисунок 5.1 – Результат виконання тестів для Q64_128 (рисунок виконаний самостійно)

5.2 Інтеграційне тестування смартконтракту пулів ліквідності

Для перевірки коректної взаємодії компонентів смартконтракту пулів ліквідності було застосовано інтеграційне тестування, що дозволяє оцінити роботу всієї системи в реальних умовах виконання. На відміну від модульного, цей тип тестування охоплює всю зв’язку між інструкціями, акаунтами, та внутрішнім станом контракту, забезпечуючи цілісну перевірку поведінки Anchor-програми.

Тестування проводиться у повноцінному віртуальному середовищі, що імітує поведінку блокчейну Solana, з використанням фреймворку Anchor та бібліотек Mocha і Chai для побудови тестових сценаріїв на стороні TypeScript. Інструкції створення об'єктів, їх оновлення, взаємодії між обліковими записами та обробки результатів – усе тестується у зв’язку, в межах повного життєвого циклу програми. Такий підхід дозволяє переконатися, що всі складові програмної логіки працюють узгоджено в єдиному середовищі, а також виявити помилки, які не проявляються під час ізольованого модульного тестування.

Повний чек-лист тестових випадків для смартконтракту пулів ліквідності наведено в додатку Г в таблиці Г.2.

На основі сформованого чек-листа було реалізовано автоматизовані інтеграційні тести. Для кожного пункту списку було створено відповідні тест-кейси з результатами тестування. Приклади отриманих тест-кейсів наведено в таблицях 5.3 та 5.4.

Таблиця 5.3 – Тест-кейс перевірки ініціалізація пулу ліквідності з токеном та токеном 2022 з дозволеним розширенням TransferFeeConfig

Інформація про тестовий випадок			
Ідентифікатор тестового випадку	LIQ-CON-41		
Власник тесту	Міленний І. А.		
Місцезнаходження тесту (шлях)	./onchain/test/liquidity-pool/cp-amm.test.ts		
Дата останнього перегляду	29.05.2025		
Технічна вимога, що тестується	LIQ-CON-INITTF		
Конфігурація засобів тестування	anchor test		
Взаємозалежність тестових випадків	Проходження тесту LIQ-CON-40		
Мета тесту	Перевірити ініціалізацію пулу ліквідності з токеном 2022, що має розширення TransferFeeConfig		
Методика тестування			
Налаштування прогону тесту	anchor test		N/A *
Крок	Дія	Очікуваний результат	Відмітка (V)*
1	Підготувати вхідні дані для ініціалізації CpAmm, включаючи токен 2022 з TransferFeeConfig	Усі дані сформовані без помилок	(V)

Продовження Таблиці 5.3

2	Виконати інструкцію ініціалізації через Anchor	Транзакція відправлена без помилок	(V)
3	Перевірити створені акаунтів: lp_mint, vaults, CpAmm	Усі акаунти існують та мають очікувані значення	(V)
4	Перевірити стан об'єкта CpAmm	Поля CpAmm ініціалізовано відповідно до специфікації	(V)
Очистка після прогону тесту	Не проводиться		
Результати тесту			
Тестувальник: Міленний І. А.	Дата прогону тесту: 29.05.2025	Результат тесту (P/F/B)*: ПРОЙДЕНО (P)	

Таблиця 5.4 – Тест-кейс перевірки помилки при обміні в пулу ліквідності з перевищенням прослизання

Інформація про тестовий випадок			
Ідентифікатор тестового випадку	LIQ-CON-53		
Власник тесту	Міленний І. А.		
Місцезнаходження тесту (шлях)	./onchain/test/liquidity-pool/cp-amm.test.ts		
Дата останнього перегляду	29.05.2025		
Технічна вимога, що тестується	LIQ-CON-EXSLIP		
Конфігурація засобів тестування	anchor test		
Взаємозалежність тестових випадків	Проходження тесту LIQ-CON-52		
Мета тесту	Перевірити, що транзакція обміну не пройде при перевищенні прослизання		
Методика тестування			
Налаштування прогону тесту	anchor test		N/A *
Крок	Дія	Очікуваний результат	Відмітка (V)*
1	Отримати актуальні дані пулу CpAmm	Дані отримані без помилок	(V)

Продовження Таблиці 5.4

2	Підготувати вхідні параметри swap з нульовим slippage tolerance та сформувати інструкцію		
3	Виконати транзакцію обміну		
Очистка після прогону тесту	Не проводиться		
Результати тесту			
Тестувальник: Міленний І. А.	Дата прогону тесту: 29.05.2025	Результат тесту (P/F/B)*: ПРОЙДЕНО (P)	

Результати проходження всіх тестів наведені на рисунку 5.2.

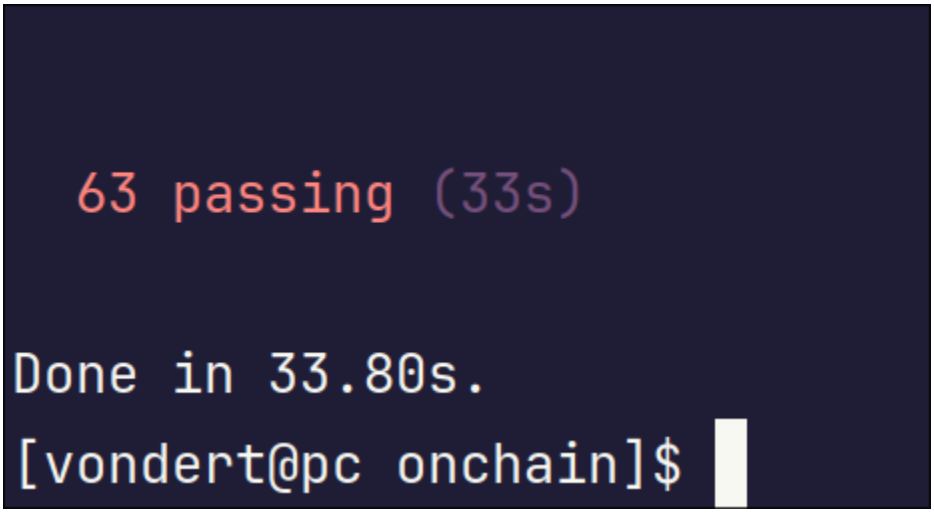

Рисунок 5.2 – Результат виконання тестів для смартконтракту пулів ліквідності (рисунок виконаний самостійно)

5.3 Інтеграційне тестування смартконтракту лаунчпулів

Для перевірки коректної взаємодії компонентів смартконтракту лаунчпулів також було застосовано інтеграційне тестування та такий самий набор інструментів як для тестування смартконтракту пулів ліквідності.

Повний чек-лист тестових випадків для смартконтракту лаунчпулів наведено в додатку Г в таблиці Г.3.

На основі сформованого чек-листа було реалізовано автоматизовані інтеграційні тести. Для кожного пункту списку було створено відповідні тест-кейси з результатами тестування. Приклади отриманих тест-кейсів наведено в таблицях 5.5 та 5.6.

Таблиця 5.5 – Тест-кейс перевірки помилки ініціалізації менеджера конфігурацій із некоректним головним авторитетом

Інформація про тестовий випадок	
Ідентифікатор тестового випадку	LNCH-CON-02
Власник тесту	Міленний І. А.
Місцезнаходження тесту (шлях)	./onchain/test/launchpool/launchpools-configs-manager.test.ts
Дата останнього перегляду	30.05.2025
Технічна вимога, що тестується	LNCH-CON-UEINITMAN
Конфігурація засобів тестування	anchor test
Взаємозалежність тестових випадків	Проходження тесту LNCH-CON-01

Продовження Таблиці 5.5

Мета тесту		Перевірити, що ініціалізація менеджера конфігурацій із некоректним головним авторитетом завершується помилкою.	
Методика тестування			
Налаштування прогону тесту	anchor test		N/A *
Крок	Дія	Очікуваний результат	Відмітка (V)*
1	Сформувати вхідні дані з неправильним головним авторитетом	Вхідні дані сформовано без помилок	(V)
2	Створити інструкцію ініціалізації менеджера конфігурацій	Інструкцію створено коректно	(V)
3	Відправити транзакцію з ініціалізацією	Транзакція завершується помилкою	(V)
Очистка після прогону тесту	Не проводиться		
Результати тесту			
Тестувальник: Міленний І. А.	Дата прогону тесту: 30.05.2025	Результат тесту (P/F/B)*: ПРОЙДЕНО (P)	

Таблиця 5.6 – Тест-кейс перевірки помилки оновлення авторитету винагороди без авторизації

Інформація про тестовий випадок			
Ідентифікатор тестового випадку	LIQ-CON-21		
Власник тесту	Міленний І. А.		
Місцезнаходження тесту (шлях)	./onchain/test/liquidity-pool/launchpools-config.test.ts		
Дата останнього перегляду	30.05.2025		
Технічна вимога, що тестується	LIQ-CON-UNUPRA		
Конфігурація засобів тестування	anchor test		
Взаємозалежність тестових випадків	Проходження тесту LIQ-CON-20		
Мета тесту	Перевірити, що оновлення авторитету винагороди без авторизації завершується помилкою.		
Методика тестування			
Налаштування прогону тесту	anchor test		N/A *
Крок	Дія	Очікуваний результат	Відмітка (V)*
1	Сформувати вхідні дані з некоректним підписантом	Дані отримані без помилок	(V)

Продовження Таблиці 5.6

2	Створити інструкцію оновлення з цими даними	Інструкцію створено коректно	(V)
3	Відправити транзакцію з ініціалізацією	Транзакція завершується помилкою	(V)
Очистка після прогону тесту	Не проводиться		
Результати тесту			
Тестувальник: Міленний І. А.	Дата прогону тесту: 30.05.2025	Результат тесту (P/F/B)*: ПРОЙДЕНО (P)	

Результати проходження всіх тестів наведені на рисунку 5.3.

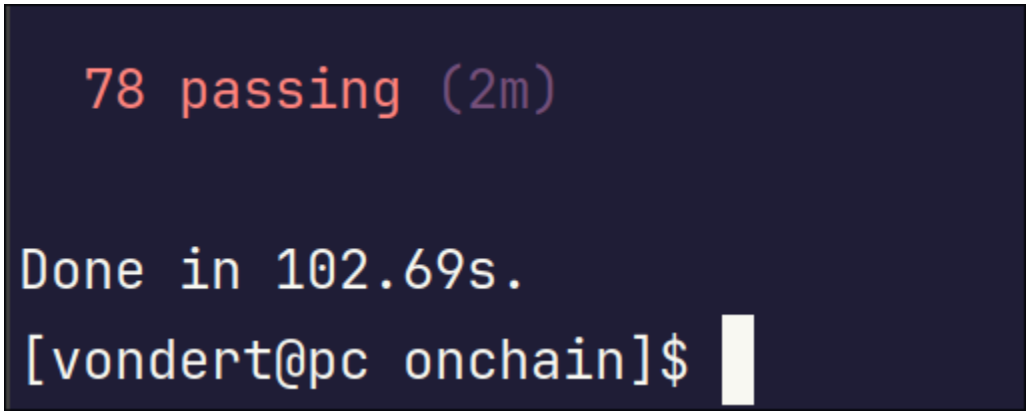

Рисунок 5.3 – Результат виконання тестів для смартконтракту лаунчпулів (рисунок виконаний самостійно)

5.4 Модульне тестування індексатора транзакцій

Для тестування індексатора було обрано підхід модульного тестування (unit-тестів), оскільки перевірка передбачає десеріалізацію подій із логів транзакцій. Особливу увагу приділено макросам, що відповідають за кодогенерацію методів десеріалізації. Для тестування було використано стандартний інструментарій пакету Cargo, що входить до екосистеми мови програмування Rust.

Результати проходження всіх тестів наведені на рисунку 5.4.

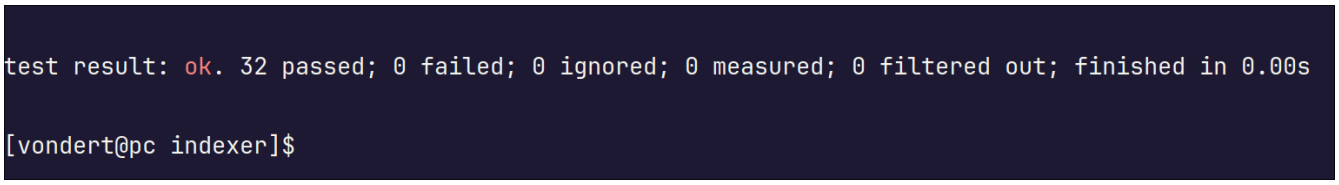

Рисунок 5.4 – Результат виконання тестів для індексатора транзакцій (рисунок виконаний самостійно)

5.5 Інтеграційне тестування бекенд-сервера

Для тестування бекенд-сервера, зокрема його частини, що відповідає за формування транзакцій, було прийнято рішення реалізувати комплексний інтеграційний тест, який охоплює всі компоненти системи, окрім веб-клієнта. Самі тести реалізовано з використанням фреймворків Anchor, Mocha та Chai. Інтеграційні тести смартконтрактів, описані в підрозділах 5.2 та 5.3, були адаптовані: тепер замість безпосереднього виклику інструкцій, вони надсилають HTTP-запити до бекенд-сервера, який повертає транзакцію у вигляді base64. Таким чином, тестується повний цикл взаємодії між серверною логікою та блокчейном, що дозволяє перевірити правильність обробки запитів і генерації транзакцій на рівні бекенда.

Результати проходження всіх тестів наведені на рисунку 5.5.

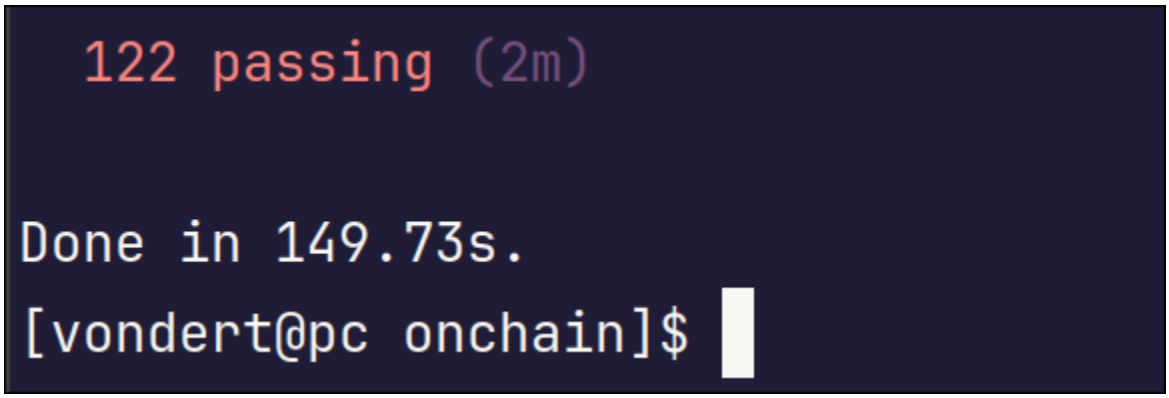

Рисунок 5.5 – Результат виконання тестів для бекенд-сервера (рисунок виконаний самостійно)

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було проаналізовано предметну галузь децентралізованого обміну токенів, спроектовано та розроблено програмну систему Vondex – децентралізованої платформи обміну токенів, що функціонує на базі блокчейну Solana. Розроблена система відповідає сучасним вимогам до децентралізованих фінансових застосунків.

Vondex є взаємодією шести ключових компонентів, які утворюють цілісну децентралізовану екосистему. Ончейн-частина реалізована у вигляді двох окремих Anchor-програм, відповідальних за роботу з пулами ліквідності та лаунчпулами. Офчейн-сервіси включають Listener-сервер для індексації подій у реальному часі та високопродуктивний бекенд-сервер на базі Axum, що відповідає за обробку клієнтських запитів і формування транзакцій. Система зберігання використовує ScyllaDB – розподіленій базі даних, яка виконує роль кеша та джерела проіндексованих історичних даних. Клієнтська частина створена з використанням Next.js, що забезпечує інтерактивну взаємодію з користувачем і доступ до всіх функціональних можливостей платформи.

Важливою особливістю розробленої системи є чіткий розподіл ролей користувачів, що забезпечує безпеку. Система підтримує підписання транзакцій на боці клієнта, не зберігаючи приватних ключів, що відповідає принципам Web3-дизайну.

Проектування компонентів здійснювалося з урахуванням високих вимог до масштабованості, швидкодії та реального навантаження, що дозволяє адаптувати платформу як до початкових запусків проектів, так і до високонавантажених торгових операцій.

У перспективі можливо доповнення системи механізмами кросчейн-обміну, більш продвинутими пулами ліквідності, деревативною торгівлею, індексацією

метаданих токенів та нативним стекінгом VON. Це дозволить Vondex стати універсальним інструментом для роботи з цифровими активами нв мережі Solana.

Таким чином, результати роботи підтверджують доцільність і перспективність побудови децентралізованих обмінників на блокчейні Solana, для забезпечення швидкого, прозорого та безпечного обміну токенів.

Апробація отриманих результатів відбулася у вигляді представлення матеріалів на І Міжнародній науково-технічній конференції «Сучасні інформаційні технології та системи штучного інтелекту» MIT@AIS-2025 (див. дод. Д), що засвідчило актуальність і практичну цінність проведеного дослідження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Бутерін В. Ethereum White Paper: A Next-Generation Smart Contract and Decentralized Application Platform / Віталік Бутерін. – 2014. – 36 с. – [Електронний ресурс]. – Режим доступу: https://ethereum.org/en/whitepaper/

2. 1Kosmos. Smart Contracts: Smart Contracts – [Електронний ресурс]. – Режим доступу: https://www.1kosmos.com/article/smart-contracts/

3. Яковенко А. Solana: A New Architecture for a High Performance Blockchain / Анатолій Яковенко. – San Francisco: Solana Labs, 2019. – 32 с. – [Електронний ресурс]. – Режим доступу: https://solana.com/solana-whitepaper.pdf

4. Solana Developers. Developer Courses – [Електронний ресурс]. – Режим доступу: https://solana.com/ru/developers/courses

5. Uniswap Developers. Uniswap V3 Development Book, 2017. – [Електронний ресурс]. Режим доступу: https://uniswapv3book.com/milestone_0/constant-function-market-maker.html

6. Uniswap Developers. Головна сторінка DEX Uniswap – [Електронний ресурс]. – Режим доступу: https://app.uniswap.org/swap

7. Orca Developers. Orca Protocol Documentation – [Електронний ресурс]. – Режим доступу: https://docs.orca.so/

8. Orca Developers. Головна сторінка DEX Orca – [Електронний ресурс]. – Режим доступу: https://www.orca.so/

9. Raydium Developers. Hybrid AMM Developer Documentation, 2020. – [Електронний ресурс]. – Режим доступу: https://github.com/raydium-io/raydium-docs/blob/master/dev-resources/raydium-hybrid-amm-dev-doc.pdf

10. Raydium Developers. Головна сторінка DEX Raydium – [Електронний ресурс]. – Режим доступу: https://raydium.io/liquidity-pools/

11. Кириченко І. В., Терещенко Г. Ю., Груздо І. В. Застосування симетричних алгоритмів в блокчейні // Біоніка інтелекту. – 2020. – № 1 (94). – С. 33–39. – Харків: ХНУРЕ. – doi:10.30837/bi.2020.1(94).11

12. Міленний І. А., Кириченко І. В., Терещенко Г. Ю. Децентралізована система торгівлі токенами з високоточними обчисленнями // І МНПК «Сучасні інформаційні технології та системи штучного інтелекту MIT@AIS-2025», – 2025. – с. 58.

13. Solana Developers. Solana Programs – [Електронний ресурс]. – Режим доступу: https://solana.com/ru/docs/core/programs

14. Solana Foundation. Anchor Docs – [Електронний ресурс]. – Режим доступу: https://www.anchor-lang.com/docs

15. Codama Developers. Codama Repository – [Електронний ресурс]. – Режим доступу: https://github.com/codama-idl/codama

16. Github репозиторій проекту – [Електронний ресурс]. – Режим доступу: https://github.com/NureMilennyiIvan/2025_B_PI_PZPI-21-11_Milennyi_I_A