Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп’ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп’ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Дослідження застосунків блокчейну та смарт-контрактів засобами мови програмування Rust для оптимізації різногалузевих задач

Виконали: студенти VI курсу, групи САм-61
спеціальності 124 Системний аналіз
(шифр і назва спеціальності)

_______ Семак А. В.
(підпис) (прізвище та ініціали)

_______ Хома С.-З. Ю.
(підпис) (прізвище та ініціали)

Керівник _______ Козбур Г.В.
(підпис) (прізвище та ініціали)

Нормоконтроль _______ Никитюк В.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _______ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _______ Бойко І.В.
(підпис) (прізвище та ініціали)

Рецензент _______ Осухівська Г.М.
(підпис) (прізвище та ініціали)

Тернопіль
2023

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____________ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«28» грудня 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 124 Системний аналіз
(шифр і назва спеціальності)

Студенту Семаку Антонію Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження застосунків блокчейну та смарт-контрактів засобами мови програмування Rust для оптимізації різногалузевих задач

Керівник роботи Козбур Галина Володимирівна, к.т.н., доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 24 » листопада 2023 року № 4/7-1096

2. Термін подання студентом завершеної роботи 29 грудня 2023р.

3. Вихідні дані до роботи Наукові публікації про застосунків блокчейну та смарт-контрактів засобами мови програмування Rust для оптимізації різногалузевих задач

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз існуючих рішень та обґрунтування теми магістерської роботи. 2 Технологія блокчейн. Застосування платформи Solana та мови програмування Rust для розробки смарт-контрактів. 3 Практична реалізація смарт-контрактів у блокчейні Solana 4 Охорона праці та безпека в надзвичайних ситуаціях. Висновки. Перелік джерел. Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1 Титульна сторінка. 2 Тема, Мета, Об'єкт, Предмет дослідження. 3 Завдання дослідження. 4 Актуальність дослідження. 5 Блокчейн. Переваги та недоліки 6 Нода. Методи досягнення консенсусу 7 Інноваційні технології для виборів та ліквідні пули 8 Смарт-контракти. Переваги та недоліки 9 Існуючі рішення вирішення проблеми 10 Мова програмування Rust. Переваги та недоліки 11 Блокчейн Solana. Переваги та недоліки 12 Схема голосування на виборах 13 Смарт-контракт для системи голосування на виборах 14 Схема liquidity pool 15 Створення смарт-контракту для liquidity pool 16 Інтеграційні тести для функціоналу 17 Висновки. 18 Завершальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Сенчишин В.С., доцент		
Безпека в надзвичайних ситуаціях	Клепчик В.М., ст. викладач		

7. Дата видачі завдання 24 листопада 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	25.11.2023	*Виконано*
2.	Підбір наукових джерел про оптимізацію	26.11.2023-28.11.2023	*Виконано*
	різногалузевих задач засобами блокчейну		
3.	Опрацювання наукових публікацій та збір даних по	29.11.2023-1.12.2023	*Виконано*
	темі роботи		
4.	Виконання дослідження згідно мети кваліфікаційної	2.12.2023-4.12.2023	*Виконано*
	роботи		
5.	Оформлення розділу «Аналіз існуючих рішень та	5.12.2023-7.12.2023	*Виконано*
	обґрунтування теми магістерської роботи»		
6.	Оформлення частини розділу «Технологія блокчейн.	8.12.2023-10.12.2023	*Виконано*
	Застосування платформи Solana та мови		
	програмування Rust для розробки смарт-контрактів»,		
	про Visual Studio Code та Linux		
7.	Оформлення частини розділу «Практична реалізація	11.12.2023-13.12.2023	*Виконано*
	смарт-контрактів у блокчейні Solana» про		
	смарт-контракт для системи голосування		
8.	Виконання завдання до підрозділу «Охорона праці»	14.12.2023-15.12.2023	*Виконано*
9.	Виконання завдання до підрозділу «Безпека в	16.12.2023-17.12.2023	*Виконано*
	надзвичайних ситуаціях»		
10.	Оформлення кваліфікаційної роботи	18.12.2023-19.12.2023	*Виконано*
11.	Нормоконтроль	19.12.2023-20.12.2023	*Виконано*
12.	Перевірка на плагіат	21.12.2023	*Виконано*
13.	Попередній захист кваліфікаційної роботи	22.12.2023	*Виконано*
14.	Захист кваліфікаційної роботи	29.12.2023	

Студент ____________ Семак А. В.
(підпис) (прізвище та ініціали)

Керівник роботи ____________ Козбур Г.В.
(підпис) (прізвище та ініціали)

Міністерство освіти і науки України

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____________ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«28» грудня 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 124 Системний аналіз
(шифр і назва спеціальності)

Студенту Хомі Святославу-Зоряну Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження застосунків блокчейну та смарт-контрактів засобами мови програмування Rust для оптимізації різногалузевих задач

Керівник роботи Козбур Галина Володимирівна, к.т.н., доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 24 » листопада 2023 року № 4/7-1096

2. Термін подання студентом завершеної роботи 29 грудня 2023р.

3. Вихідні дані до роботи Наукові публікації про застосунків блокчейну та смарт-контрактів засобами мови програмування Rust для оптимізації різногалузевих задач

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз існуючих рішень та обґрунтування теми магістерської роботи. 2 Технологія блокчейн. Застосування платформи Solana та мови програмування Rust для розробки смарт-контрактів. 3 Практична реалізація смарт-контрактів у блокчейні Solana 4 Охорона праці та безпека в надзвичайних ситуаціях. Висновки. Перелік джерел. Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1 Титульна сторінка. 2 Тема, Мета, Об'єкт, Предмет дослідження. 3 Завдання дослідження. 4 Актуальність дослідження. 5 Блокчейн. Переваги та недоліки 6 Нода. Методи досягнення консенсусу 7 Інноваційні технології для виборів та ліквідні пули 8 Смарт-контракти. Переваги та недоліки 9 Існуючі рішення вирішення проблеми 10 Мова програмування Rust. Переваги та недоліки 11 Блокчейн Solana. Переваги та недоліки 12 Схема голосування на виборах 13 Смарт-контракт для системи голосування на виборах 14 Схема liquidity pool 15 Створення смарт-контракту для liquidity pool 16 Інтеграційні тести для функціоналу 17 Висновки. 18 Завершальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Сенчишин В.С., доцент		
Безпека в надзвичайних ситуаціях	Клепчик В.М., ст. викладач		

7. Дата видачі завдання 24 листопада 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	25.11.2023	*Виконано*
2.	Підбір наукових джерел про оптимізацію	26.11.2023-28.11.2023	*Виконано*
	різногалузевих задач засобами блокчейну		
3.	Опрацювання наукових публікацій та збір даних по	29.11.2023-1.12.2023	*Виконано*
	темі роботи		
4.	Виконання дослідження згідно мети кваліфікаційної	2.12.2023-4.12.2023	*Виконано*
	роботи		
5.	Оформлення розділу «Аналіз існуючих рішень та	5.12.2023-7.12.2023	*Виконано*
	обґрунтування теми магістерської роботи»		
6.	Оформлення частини розділу «Технологія блокчейн.	8.12.2023-10.12.2023	*Виконано*
	Застосування платформи Solana та мови		
	програмування Rust для розробки смарт-контрактів»,		
	про Solana та Rust		
7.	Оформлення частини розділу «Практична реалізація	11.12.2023-13.12.2023	*Виконано*
	смарт-контрактів у блокчейні Solana» про		
	смарт-контракт для liquidity pool		
8.	Виконання завдання до підрозділу «Охорона праці»	14.12.2023-15.12.2023	*Виконано*
9.	Виконання завдання до підрозділу «Безпека в	16.12.2023-17.12.2023	*Виконано*
	надзвичайних ситуаціях»		
10.	Оформлення кваліфікаційної роботи	18.12.2023-19.12.2023	*Виконано*
11.	Нормоконтроль	19.12.2023-20.12.2023	*Виконано*
12.	Перевірка на плагіат	21.12.2023	*Виконано*
13.	Попередній захист кваліфікаційної роботи	22.12.2023	*Виконано*
14.	Захист кваліфікаційної роботи	29.12.2023	

Студент _______ Хома С.-З. Ю.

(підпис) (прізвище та ініціали)

Керівник роботи _______ Козбур Г.В.

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дослідження застосунків блокчейну та смарт-контрактів засобами мови програмування Rust для оптимізації різногалузевих задач // Кваліфікаційна робота освітнього рівня «Магістр» // Семак Антоній Вікторович // Хома Святослав-Зорян Юрійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра комп’ютерних наук, група САм-61 // Тернопіль, 2023 // С. 101, рис. – 50, табл. – 0, кресл. – 18, додат. – 3, бібліогр. – 75.

Ключові слова: смарт-контракт, блокчейн, Rust, VSCode, Linux, DeFi, Solana, Cargo.

Кваліфікаційна робота присвячена розробці смарт-контракту, що реалізує систему голосування та смарт-контракту, що реалізує liquidity pool. В першому розділі кваліфікаційної роботи вислітлено аналіз існуючих рішень. Розглянуто поняття ноди та метод досягнення консенсусу для роботи смарт-контрактів. Проаналізовано поняття смарт-контрактів, їхні переваги і недоліки.

В другому розділі кваліфікаційної роботи описано технологію блокчейн. Досліджено застосування платформи Solana та мови програмування Rust для розробки смарт-контрактів. Подано опис середовища для програмування Visual Studio Code. Розглянуто операційну систему Linux

В третьому розділі кваліфікаційної роботи описано процес практичної реалізації смарт-контрактів у блокчейні Solana. Спроектовано архітектуру смарт-контракту, що реалізує систему голосування та смарт-контракту, що реалізує liquidity pool. Проведено тестування функціоналу розроблених смарт-контрактів за допомогою інтеграційних тестів. Об’єкт дослідження: оптимізація різногалузевих задач за допомогою застосунків блокчейну та смарт-контрактів. Предмет дослідження: застосунки блокчейну та смарт-контракти засобами мови програмування Rust.

ANNOTATION

Research of blockchain applications and smart contracts using Rust programming tools to optimize multi-sectoral problems // The educational level "Master" qualification work // Semak Antonii // Khoma Sviatoslav-Zorian // Ternopil Ivan Pulyuy National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, SAm-61 group // Ternopil, 2023 // P. 101, fig. – 50, tables – 0, posters – 18, annexes – 3, ref. – 75.

Key words: smart contract, blockchain, Rust, VSCode, Linux, DeFi, Solana, Cargo.

Thesis is devoted to the development of a smart contract implementing a voting system and a smart contract implementing a liquidity pool. The first chapter of the thesis analyzes existing solutions. The concept of a node and the method of reaching consensus for the operation of smart contracts are considered. The concept of smart contracts, their advantages and disadvantages are analyzed.

The second chapter of the thesis describes the blockchain technology. The application of the Solana platform and the Rust programming language for the development of smart contracts is investigated. A description of the Visual Studio Code programming environment is given. The Linux operating system is considered.

The third chapter of the thesis describes the process of practical implementation of smart contracts in the Solana blockchain. The architecture of a smart contract that implements the voting system and a smart contract that implements the liquidity pool is designed. The functionality of the developed smart contracts was tested using integration tests. Object of research: optimization of diverse tasks using blockchain applications and smart contracts. Subject of research: blockchain applications and smart contracts using the Rust programming language.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

Блокчейн – розподілена база даних, яка забезпечує високий рівень безпеки та прозорості за рахунок зберігання даних у ланцюжку блоків.

Децентралізовані автономні організації (ДАО) – форма організації, яка базується на технології блокчейн та смарт-контрактах

Uniswap – децентралізована платформа для обміну криптовалют (DEX), що базується на технології смарт-контрактів Ethereum.

Ethereum (ETH) – блокчейн-платформа для смарт-контрактів.

RC-20 (Ethereum Request for Comments 20) – стандарт для розробки токенів на блокчейні Ethereum.

Cardano – інноваційна блокчейн-мережа з доказом частки (PoS), яка перетворюється в платформу для розробки децентралізованих додатків (DApp) з реєстром багатьох активів і перевіреними смарт-контрактами.

Solana – блокчейн-платформа, яка була розроблена для підтримки розподілених додатків (DApps) та криптовалют.

Rust – мова програмування загального призначення, яка виникла у 2010 році та розвивалася компанією Mozilla.

Visual Studio Code (VSCode) – безкоштовний текстовий редактор, розроблений компанією Microsoft.

ЗМІСТ

ВСТУП 8
1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ МАГІСТЕРСЬКОЇ РОБОТИ 10
1.1 Блокчейн 10
1.2 Переваги та недоліки блокчейну 13
1.3 Поняття ноди 14
1.4 Методи досягнення консенсусу 16
1.5 Ліквідні пули 17
1.6 Інноваційні технології для виборів 21
1.7 Смарт-контракти 22
1.8 Переваги та недоліки смарт-контрактів 25
1.9 Аналіз існуючих рішень 26
1.10 Висновок до першого розділу 28
2 ТЕХНОЛОГІЯ БЛОКЧЕЙН. ЗАСТОСУВАННЯ ПЛАТФОРМИ SOLANA ТА МОВИ ПРОГРАМУВАННЯ RUST ДЛЯ РОЗРОБКИ СМАРТ-КОНТРАКТІВ 29
2.1 Блокчейн Solana 29
2.2 Мова програмування Rust 32
2.3 Середовище для програмування Visual Studio Code 37
2.4 Операційна система Linux 44
2.5 Висновок до другого розділу 52
3 ПРАКТИЧНА РЕАЛІЗАЦІЯ СМАРТ-КОНТРАКТІВ У БЛОКЧЕЙНІ SOLANA 53
3.1 Написання смарт-контракту, який реалізує систему голосування..... 53
3.2 Вхідна точка у смарт-контракт, який реалізує систему голосування 54
3.3 Інструкції смарт-контракту, який реалізує систему голосування...... 56
3.4 Помилки смарт-контракту, який реалізує систему голосування........ 57
3.5 Структури даних смарт-контракту, який реалізує систему голосування 57

3.6 Основна логіка смарт-контракту, який реалізує систему голосування ... 59
3.7 Тести смарт-контракту, який реалізує систему голосування ... 62
3.8 Написання смарт-контракту, який реалізує liquidity pool ... 66
3.9 Структури даних смарт-контракту, який реалізує liquidity pool ... 68
3.10 Помилки смарт-контракту, який реалізує liquidity pool ... 68
3.11 Основна логіка смарт-контракту, який реалізує liquidity pool ... 69
3.12 Тести смарт-контракту, який реалізує liquidity pool ... 73
3.13 Висновок до третього розділу ... 76
4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ ... 77
4.1 Вимоги щодо охорони праці при роботі з комп’ютерами. ... 77
4.2 Організація охорони праці працівників на підприємстві ... 79
4.3 Інженерний захист персоналу об’єкту та населення ... 81
4.4 Критичні стани людини ... 83
4.5 Підвищення стійкості роботи об'єктів приладобудівної галузі в воєнний час ... 86
4.6 Висновок до четвертого розділу ... 92
ВИСНОВКИ ... 94
ПЕРЕЛІК ДЖЕРЕЛ ... 96
ДОДАТКИ

ВСТУП

Актуальність теми. Вибрана тема є актуальною через важливість блокчейн-технологій та смарт-контрактів у сучасному світі. Використання мови програмування Rust може призвести до поліпшення безпеки та ефективності у порівнянні з іншими мовами. Оптимізація застосувань для різних галузей може вести до нових інновацій та зменшення витрат. Інтердисциплінарний характер дослідження робить його цікавим та обіцяючим для внесення вагомого внеску у розвиток технологій. Завдяки швидкому розвитку галузі, завжди є потреба в нових дослідженнях для визначення оптимальних підходів та розробки покращених рішень.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Магістр» є дослідження питання підвищення прозорості виборчих процесів та процесів формування курсів для обміну валют. Для досягнення поставленої мети потрібно виконати ряд завдань, зокрема:

– подати інформацію про блокчейн, а також вказати його переваги на недоліки. (Хома С.-З. Ю.);

– розглянути поняття ноди та метод досягнення консенсусу для роботи смарт-контрактів. (Семак А. В.);

– описати інноваційні технології для виборів та ліквідні пули.
(Хома С.-З. Ю.);

– проаналізувати смарт-контракти, їхні переваги і недоліки. (Семак А. В.);

– подати та проаналізувати інформацію про існуючі рішення проблеми. (Хома С.-З. Ю.);

– подати інформацію про блокчейн Solana, а також вказати його переваги на недоліки. (Хома С.-З. Ю.);

– розглянути мову програмування Rust, її недоліки та переваги, а також обґрунтовати її вибір. (Хома С.-З. Ю.);

– проаналізувати програмне середовище Visual Studio Code. (Семак А. В.);

– розглянути операційну систему Linux та її переваги і недоліки. (Семак А. В.);

– спроектувати смарт-контракт для системи голосування на виборах на блокчейні Solana за допомогою мови програмування Rust. (Семак А. В.);

– спроектувати смарт-контракт, що реалізує liquidity pool на блокчейні Solana за допомогою мови програмування Rust. (Хома С.-З. Ю.);

– написати інтеграційні тести для функціоналу. (Хома С.-З. Ю.);

Об’єкт дослідження оптимізація різногалузевих задач за допомогою застосунків блокчейну та смарт-контрактів.

Предмет дослідження. застосунки блокчейну та смарт-контракти засобами мови програмування Rust.

Наукова новизна одержаних результатів кваліфікаційної роботи полягає у тому, що створено початкові умови для інновацій у виборчих системах та системах обміну валют.

Практичне значення одержаних результатів. Спроектовано та розроблено базовий функціонал для смарт-контрактів для виборчої системи та liquidity pool.

Апробація результатів магістерської роботи. Основні результати проведених досліджень обговорювались на XII Міжнародній науково-практичній конференції молодих учених та студентів «Актуальні задачі сучасних технологій» Тернопільського національного технічного універсистету імені Івана Пулюя (м. Тернопіль, 2023 р.), XI науково-технічній конференції «Інформаційні моделі, системи та технології» Тернопільського національного технічного універсистету імені Івана Пулюя (м. Тернопіль, 2023 р.) та .

Публікації. Основні результати кваліфікаційної роботи опубліковано у чотирьох працях конференції (Див. додатки А).

Структура й обсяг кваліфікаційної роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку літератури з 75 найменувань та 3 додатків. Загальний обсяг кваліфікаційної роботи складає 101 сторінки, з них 67 сторінки основного тексту, який містить 50 рисунків.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ МАГІСТЕРСЬКОЇ РОБОТИ

1.1 Блокчейн

Блокчейн – це розподілена база даних, яка забезпечує високий рівень безпеки та прозорості за рахунок зберігання даних у ланцюжку блоків. Кожен блок у ланцюжку містить набір транзакцій, а нові блоки неперервно додаються в хронологічному порядку. Блокчейн відрізняється від традиційних баз даних своєю децентралізованою структурою, що означає, що немає єдиного контролюючого центру, і кожен учасник мережі має копію всієї бази даних. На рисунку 1.1 наведено схему роботи блокчейну [1].

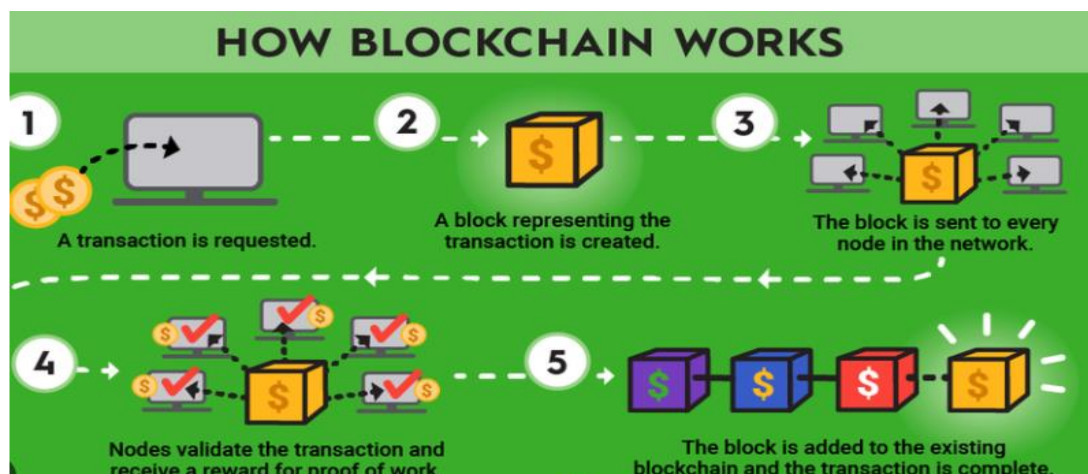

Рисунок 1.1 – Схема роботи блокчейну

Історія блокчейну є фасцинувальною, оскільки вона включає в себе елементи криптографії, комп'ютерних наук та економічних інновацій. Її початок можна відстежити до ранніх робіт у сфері криптографії та розподілених систем. У 1991 році вчені Стюарт Хабер та В. Скотт Сторнетта вперше запропонували концепцію блокчейну як частини своєї роботи по створенню системи, що забезпечувала б неможливість зміни документів після їх створення. Вони використовували хеш-дерева для забезпечення порядку та незмінності документів, що стало прототипом сучасних блокчейн-систем [2].

Хеш-дерево, також відоме як дерево Меркла, є структурою даних, яка використовує хеш-функції для ефективної перевірки цілісності та аутентичності даних у великих наборах. Воно часто використовується в криптографії та системах забезпечення цілісності даних.

Основна ідея полягає в тому, щоб представити велику кількість даних у вигляді дерева, де кожен листок (кінцевий вузол) містить хеш даного блоку даних. Вищі рівні дерева, називані також внутрішніми вузлами, об'єднуються за допомогою хешів своїх дочірніх вузлів. Найвищий вузол, відомий як корінь дерева, містить хеш всієї структури.

Перевірка цілісності виконується шляхом порівняння хеша кореня заздалегідь відомого хеша з новим обчисленим хешем. Якщо яка-небудь частина даних змінюється, змінюється і хеш цієї частини, і це стає помітно у кореневому хеші.

Хеш-дерева застосовуються в різних сферах, таких як блокчейн технології (де вони використовуються для підтвердження цілісності блоків), безпека даних та інші області, де важлива надійність та цілісність інформації.

Однак, справжній прорив в історії блокчейну стався з появою біткойну у 2008 році, коли особа (або група осіб) під псевдонімом Сатоші Накамото опублікувала статтю, в якій описала дизайн децентралізованої цифрової валюти [3]. В цій статті Накамото представив концепцію блокчейну як основу для біткойну, що включала в себе механізм доказу виконаної роботи (Proof of Work) для додавання нових блоків у ланцюг. Це було першим реальним застосуванням блокчейну, і воно відкрило нові горизонти у сфері цифрових валют та децентралізованих фінансових систем.

З того часу блокчейн стрімко розвивався та адаптувався до різноманітних застосувань. Крім фінансового сектору, технологія почала використовуватися у сферах, таких як управління ланцюгами постачання, охорона здоров'я, ідентифікація та верифікація, а також у галузі права і урядування. Інновації в блокчейні, такі як смарт-контракти та децентралізовані автономні організації (DAO), відкрили нові можливості для автоматизації та оптимізації процесів у різних галузях.

Децентралізовані автономні організації (ДАО) – це форма організації, яка базується на технології блокчейн та смарт-контрактах [4]. Це концепція, яка виникла в контексті криптовалют і блокчейн-технологій, особливо популярна під час розвитку ефіріуму. На рисунку 1.2 наведено схему роботи ДАО.

Рисунок 1.2 – Схема роботи ДАО

Система децентралізована, що означає відсутність центральної влади чи управління. Всі рішення приймаються за допомогою голосування членів чи засновників ДАО.

ДАО функціонує автономно за певними правилами та умовами, встановленими у смарт-контрактах. Це означає, що виконання функцій і прийняття рішень відбувається автоматично за певних обставин.

ДАО використовують смарт-контракти, які програмуються на блокчейні. Ці контракти містять правила та умови функціонування ДАО і виконуються автоматично при виникненні певних подій чи умов.

Важливі рішення в ДАО приймаються за допомогою голосування учасників. Кожен учасник може мати певну кількість голосів в залежності від його внеску чи статусу в організації.

Блокчейн-технологія забезпечує високий рівень транспарентності та відстеження всіх операцій та рішень ДАО. Це сприяє відкритості та довірі в середовищі.

ДАО може керувати різними ресурсами, такими як фінанси, проекти, технічні рішення тощо, використовуючи смарт-контракти та голосування.

Ідея ДАО зацікавлює галузі, де потрібна велика ступінь автономії та відсутність централізованого керівництва, наприклад, у фінансовому секторі, управлінні проектами або групами розробників [5]. Однак, варто зазначити, що існують виклики та ризики, пов'язані з безпекою та вразливістю ДАО до атак або помилок в смарт-контрактах.

Технологія блокчейну знайшла застосування в різних сферах, від фінансових послуг до ланцюгів постачання. У фінансовій сфері вона дозволяє здійснювати безпечні та прозорі транзакції без необхідності посередників, як, наприклад, у криптовалютах. У ланцюгах постачання блокчейн забезпечує простежуваність продукції, від сировини до кінцевого споживача, підвищуючи таким чином довіру та ефективність.

Однак, існують і виклики, пов'язані з блокчейном. Серед них – масштабування мереж, енерговитратність деяких блокчейн-систем та необхідність розробки нових правових рамок для регулювання цієї сфери. Крім того, блокчейн може зіткнутися з проблемами приватності, оскільки, хоча він забезпечує анонімність транзакцій, історія цих транзакцій залишається публічно доступною на незмінному ланцюжку.

З урахуванням цих факторів, актуальність блокчейну як технології залишається високою, оскільки вона продовжує розвиватися та адаптуватися до різноманітних потреб сучасного світу.

1.2 Переваги та недоліки блокчейну

Блокчейн представляє собою технологію, яка кардинально змінює підходи до зберігання, обробки та передачі даних, пропонуючи унікальні характеристики, які відрізняють її від традиційних систем. Однією з основних особливостей блокчейну є децентралізація. Він не залежить від одного центрального сервера або адміністратора; замість цього дані розподіляються по мережі комп'ютерів, що забезпечує велику стійкість до збоїв та атак [6]. Кожен блок у ланцюзі містить пакет транзакцій, які криптографічно захищені і пов'язані з попередніми блоками, що робить ланцюг незмінним.

Іншою значущою характеристикою блокчейну є прозорість. Хоча учасники мережі можуть зберігати анонімність, самі транзакції є повністю прозорими і верифікованими всіма учасниками мережі. Це забезпечує високий рівень відкритості та довіри, особливо важливий у фінансових угодах та контрактах.

Безпека є ще однією ключовою перевагою блокчейну. Використання складних криптографічних алгоритмів робить блокчейн дуже стійким до зовнішніх атак та спроб фальсифікації [7]. Ця характеристика робить блокчейн ідеальним для забезпечення безпеки фінансових транзакцій та зберігання конфіденційних даних.

Крім того, блокчейн сприяє високій ефективності та швидкості транзакцій. Відсутність необхідності в центральному посереднику, як-от банку або іншій фінансовій установі, дозволяє здійснювати транзакції безпосередньо між сторонами, значно знижуючи час та витрати.

Незмінність даних у блокчейні є ще однією важливою особливістю. Після того, як інформація додана до блокчейну, вона не може бути змінена або видалена, що гарантує цілісність та достовірність записів. Це особливо важливо для юридичних документів, прав власності, та інших важливих записів.

У порівнянні з традиційними системами, блокчейн пропонує величезні переваги в плані безпеки, прозорості, та ефективності. Ці характеристики роблять його ідеальним для широкого спектру застосувань, від фінансових операцій до управління ланцюгами постачання, від охорони здоров'я до ідентифікації та верифікації [8]. Хоча блокчейн має свої виклики, такі як масштабування та енергетична ефективність, його потенціал та переваги продовжують спонукати до інновацій та впровадження у все нових сферах.

1.3 Поняття ноди

Ноди в блокчейні відіграють ключову роль у функціонуванні та підтримці цієї технології. Нода – це будь-який комп'ютер, який підключений до мережі

блокчейну і бере участь у її процесах. Вони виконують різні функції, включаючи зберігання даних, обробку транзакцій та участь у механізмі консенсусу.

Кожна нода має копію повного ланцюжка блоків і регулярно синхронізується з іншими нодами, щоб забезпечити актуальність інформації. Коли новий блок створюється, він передається по мережі, і кожна нода оновлює свою копію блокчейну. Таким чином, всі дані залишаються консистентними та прозорими для всіх учасників мережі.

Механізм консенсусу в блокчейні – це спосіб, за допомогою якого ноди домовляються про валідність транзакцій та блоків. Існує кілька видів механізмів консенсусу, найвідомішими з яких є Proof of Work (PoW) та Proof of Stake (PoS). В PoW ноди, відомі як майнери, використовують обчислювальну потужність для вирішення складних математичних завдань і, таким чином, добувають нові блоки. В PoS учасники мережі блокують певну кількість своїх токенів як ставку і отримують право додавати блоки в міру того, як вони тримають та використовують свої токени.

Дані в мережі блокчейну поширюються за допомогою процесу, званого розгортанням блоків. Коли нода приймає новий блок, вона перевіряє його на валідність і додає до своєї копії блокчейну. Після цього блок поширюється до інших нод, які також перевіряють його та додають до своїх ланцюжків. Цей процес гарантує, що всі ноди мають однакову інформацію і що блокчейн залишається незмінним та безпечним.

Історія нод в блокчейну тісно пов'язана з розвитком самої технології блокчейну [9]. З появою перших криптовалют, таких як біткойн, виникла необхідність у мережі комп'ютерів, які б могли підтримувати роботу блокчейну. Ці комп'ютери стали першими нодами, що забезпечували функціонування та безпеку мережі [10]. З часом кількість нод та їх функціональність зросли, що дозволило блокчейну стати більш ефективним, гнучким і безпечним.

У сучасному світі ноди в блокчейні є критично важливими для підтримки інфраструктури DeFi, забезпечення роботи різноманітних криптовалютних мереж та інших блокчейн-застосувань. Вони забезпечують стабільність, безпеку

та неперервність роботи блокчейн-мереж, що є фундаментальним для довіри та прийняття цієї технології в усьому світі.

1.4 Методи досягнення консенсусу

Механізми Proof of Work (PoW) та Proof of Stake (PoS) є двома основними методами досягнення консенсусу в блокчейн-мережах [11]. Вони відіграють критичну роль у забезпеченні безпеки та стабільності різних криптовалют та децентралізованих систем.

Концепція PoW була вперше описана у 1993 році в роботі Цімoрі Двай і Майкла Мерріта, але її сучасне застосування у блокчейні започатковане Сатоші Накамото з виникненням біткойну у 2009 році. PoW став основою для безпеки біткойну і багатьох інших ранніх криптовалют.

У PoW ноди (майнери) використовують свою обчислювальну потужність для розв'язання складних криптографічних завдань [12]. Перша нода, яка знаходить рішення (доказ роботи), отримує право додати новий блок до блокчейну та отримує винагороду у вигляді криптовалюти. Цей процес вимагає значної обчислювальної потужності та енергії.

Головною метою PoW є запобігання подвійним витратам (double spending) та забезпечення безпеки мережі. Великі витрати енергії та обчислювальної потужності слугують як гарантія чесності майнерів, оскільки будь-які спроби шахрайства або атаки стають невигідними.

PoS був запропонований як альтернатива PoW для вирішення проблем енерговитратності. Його початки можна відслідкувати до 2011 року, а перша повноцінна реалізація відбулася у криптовалюті Peercoin.

В PoS валідатори (замість майнерів) вибираються для створення нового блоку на основі кількості валюти, яку вони "заклали" (stake) як заставу. Чим більше токенів закладено, тим більша ймовірність бути вибраним для додавання блоку [13]. Валідатори отримують винагороду у формі транзакційних комісій.

Основна мета PoS – зменшити енерговитратність та зробити мережу більш ефективною і екологічно чистою. PoS також пропонує більш демократичний та

економічно вигідний підхід, оскільки він дозволяє учасникам мережі мати частку у валідації транзакцій, не витрачаючи значних ресурсів на обчислювальну потужність.

І PoW, і PoS мають свої унікальні переваги та недоліки. PoW забезпечує високий рівень безпеки, але є критикованим за високі енерговитрати. PoS пропонує більш екологічний та економічно доступний підхід, але все ще розвиває свої механізми безпеки [14]. Обидва механізми продовжують еволюціонувати і адаптуватися, щоб забезпечити кращий баланс між безпекою, ефективністю та стійкістю в блокчейн-мережах.

1.5 Ліквідні пули

Ліквідні пули становлять один з ключових компонентів децентралізованих фінансів (DeFi), впроваджуючи новітній підхід до обміну активами і забезпечення ліквідності. Ліквідний пул – це сукупність фондів, зібраних у формі криптовалют або токенів, забезпечених учасниками пулу, відомими як ліквідні провайдери [15]. Ці пули дозволяють користувачам обмінювати токени безпосередньо через смарт-контракти, минаючи традиційних посередників, таких як біржі. На рисунку 1.3 наведено схему роботи Defi.

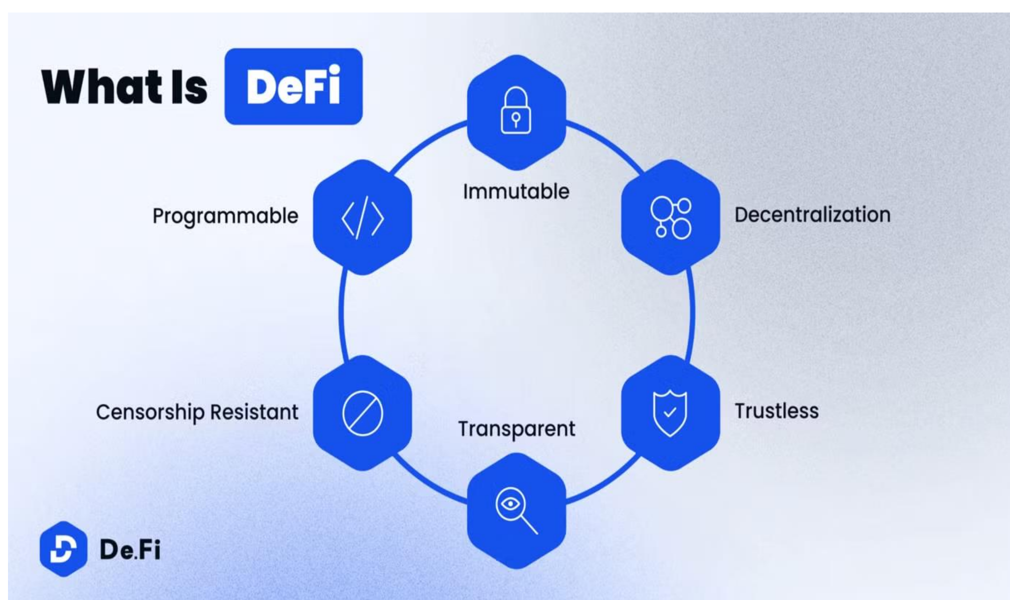

Рисунок 1.3 – Схема роботи Defi

Історія ліквідних пулів почалася з розвитком сектору DeFi, де була виявлена потреба у децентралізованих механізмах для обміну токенів. Це привело до створення перших ліквідних пулів, які надавали можливість обміну без необхідності залучення традиційних біржових посередників. Завдяки інноваційним алгоритмам ціноутворення та смарт-контрактам, ліквідні пули забезпечили більш ефективний, прозорий і доступний спосіб торгівлі криптовалютами.

Реальним прикладом використання ліквідних пулів може служити платформа Uniswap, яка дозволяє користувачам ставати ліквідними провайдерами, вкладаючи свої токени в пул і отримуючи в замін частку комісійних зборів з торгівлі. Такі платформи як Uniswap, Balancer та інші використовують унікальні алгоритми для визначення цін і забезпечення ліквідності, дозволяючи ефективно обмінювати токени без центрального посередника.

Uniswap – це децентралізована платформа для обміну криптовалют (DEX), що базується на технології смарт-контрактів Ethereum. Вона була запущена у листопаді 2018 року і стала популярною завдяки своєї інноваційної моделі автоматичного формування ліквідності (AMM) та відкритому вихідному коду.

Автоматичне формування ліквідності – це процес, за допомогою якого фінансова система чи підприємство створює механізми для забезпечення доступності достатньої кількості грошей або легко ліквідних активів для вирішення фінансових зобов'язань та забезпечення нормального функціонування. Цей процес має стратегічне значення для забезпечення фінансової стабільності та ефективного ведення бізнесу.

Моніторинг фінансового стану підприємства використовують системи моніторингу, які автоматично аналізують їхні фінансові показники, враховуючи такі аспекти, як обсяги оборотних коштів, рівень заборгованості та прогнозні показники.

З використанням аналітичних інструментів та моделей прогнозування автоматично визначаються очікувані потреби у ліквідності в майбутньому [16].

Це може включати в себе планування платежів, управління запасами та інші фінансові витрати.

Оптимізація оборотних коштів системи автоматичного управління фінансами допомагають оптимізувати рівень оборотних коштів, забезпечуючи, що гроші ефективно використовуються для покриття поточних зобов'язань та оптимізації фінансової продуктивності.

Автоматизовані системи можуть враховувати різноманітні фінансові ризики, такі як зміни в обсягах продажів, валютні ризики, зміни процентних ставок тощо, і розробляти стратегії для їхнього ефективного управління.

Використання різноманітних фінансових інструментів, таких як кредитні лінії, фінансові деривативи, може бути включено в автоматизовані стратегії для забезпечення додаткових резервів ліквідності у потрібний момент.

Ефективне керування грошовими потоками включає в себе автоматичне вирішення питань щодо виплат, внутрішньофірмового обміну грошей та інших операцій.

Автоматичне формування ліквідності дозволяє підприємствам ефективно управляти своєю фінансовою стабільністю, забезпечуючи здатність вчасно вирішувати фінансові зобов'язання та використовувати можливості для розвитку.

Uniswap надає послуги обміну криптовалют без необхідності традиційних ринкових сторін [17]. Замість цього вона використовує AMM, що дозволяє користувачам торгувати без інтермедіарів, вносячи свої активи у смарт-контракт пулу ліквідності.

Код Uniswap відкритий, що означає, що будь-хто може переглядати, вдосконалювати та використовувати його. Це сприяє розвитку спільноти та інноваціям в екосистемі.

Uniswap використовує токени на основі стандарту ERC-20 Ethereum, дозволяючи користувачам легко торгувати різними криптовалютами. RC-20 (Ethereum Request for Comments 20) – це стандарт для розробки токенів на блокчейні Ethereum [18]. Цей стандарт описує базовий набір правил та функцій, які повинен виконувати кожен токен, створений на основі цього стандарту. ERC-

20 дозволяє взаємодіяти токенам між собою та з іншими різновидами токенів на Ethereum-платформі. На рисунку 1.4 наведено схему роботи ERC-20 Ethereum.

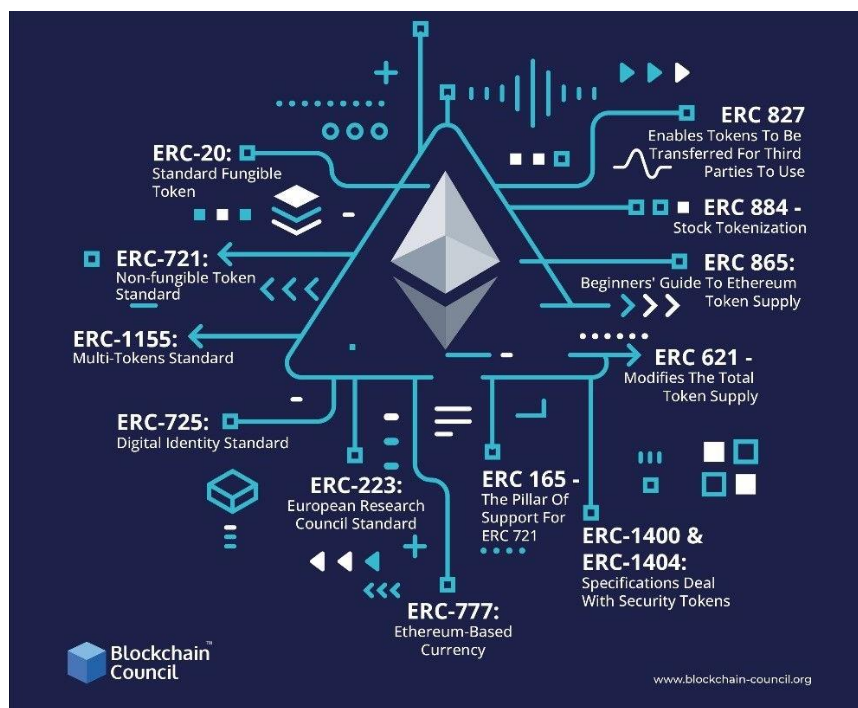

Рисунок 1.4 – Схема роботи ERC-20 Ethereum

Основні аспекти ERC-20 включають у себе баланси та перекази. Кожен учасник системи має баланс свого ERC-20 токену і може пересилати їх іншим учасникам. Доступ до балансів дозволяє іншим учасникам легко отримувати інформацію про баланс та стан транзакцій для будь-якого ERC-20 токену.

Також можливість відправки токенів є важливою функцією. Кожен, хто утримує ERC-20 токен, може відправляти їх іншим користувачам. Запит на баланс дозволяє іншим учасникам запитувати баланс конкретного гаманця. Запит на переведення токенів дозволяє учасникам запитувати та авторизовувати переведення токенів від свого імені.

Ці функції роблять ERC-20 дуже корисним для створення та взаємодії з різними токенами на платформі Ethereum. Такий стандарт сприяє створенню стандартизованих та сумісних токенів, які можуть використовуватися в різних додатках і сервісах в екосистемі Ethereum.

Uniswap відіграє важливу роль у розвитку децентралізованих фінансів (DeFi) та надає користувачам можливість торгувати криптовалютами без посередників та централізованих обмінників.

Переваги ліквідних пулів включають забезпечення високої ліквідності для криптовалютних активів, нижчі комісії порівняно з традиційними біржами, та можливість заробляти пасивний дохід від комісійних зборів як ліквідний провайдер. Крім того, вони сприяють децентралізації та демократизації фінансових ринків, оскільки будь-яка особа може стати ліквідним провайдером без значних вступних бар'єрів.

Проте, існують і мінуси. Наприклад, існує ризик тимчасових втрат (impermanent loss), коли ціна токенів у ліквідному пулі змінюється, що може призвести до втрати вартості для ліквідних провайдерів порівняно з утриманням активів поза пулом [19]. Також існує ризик смарт-контрактів, пов'язаний з можливими помилками або уразливостями у коді, що може призвести до втрати коштів.

Загалом, ліквідні пули є важливою інновацією у світі DeFi, пропонуючи нові можливості для трейдерів та інвесторів, але також потребують розуміння ризиків та обережного підходу до інвестування.

1.6 Інноваційні технології для виборів

На сучасних виборах використовуються різноманітні іноваційні технології для поліпшення процесу голосування, підвищення безпеки виборчих систем та забезпечення більшої участі виборців. Ось деякі з таких технологій:

Електронне голосування (e-voting): Це технологія, яка дозволяє виборцям голосувати електронним шляхом, часто за допомогою комп'ютерів або мобільних пристроїв. Електронне голосування може полегшити процес голосування та підвищити швидкість підрахунку голосів, але також вимагає високого рівня безпеки для уникнення можливості шахрайства.

Блокчейн для голосування: Технологія блокчейн може забезпечити високий рівень безпеки та недоторканості голосів. Кожен голос фіксується в блокчейні, що робить його надзвичайно важким для підробки чи зміни.

Електронна ідентифікація та верифікація: Сучасні технології дозволяють використовувати електронні засоби для ідентифікації виборців та перевірки їхніх особистих даних [20]. Це може допомогти уникнути подвійного голосування та інших видів шахрайства.

Мобільні додатки для голосування: Розробка мобільних додатків для голосування може сприяти більшій участі виборців, особливо серед молоді та тих, хто знаходиться далеко від місця проживання.

Штучний інтелект для аналізу даних: Використання штучного інтелекту для аналізу великих обсягів даних може допомогти виявляти та усувати аномалії в голосуванні, а також надавати аналітику для політичних досліджень.

Системи онлайн-запитань та відповідей: Ці системи дозволяють виборцям отримувати інформацію про кандидатів та питання, що стосуються голосування, зручним способом.

Соціальні мережі та медіа: Використання соціальних мереж для популяризації інформації про вибори, а також спілкування кандидатів з виборцями.

Необхідно зауважити, що впровадження та використання цих технологій повинні супроводжуватися високими стандартами безпеки, прозорості та конфіденційності, щоб уникнути можливих загроз для виборчого процесу.

1.7 Смарт-контракти

Смарт-контракт представляє собою комп'ютерний еквівалент традиційних договорів, який виконується автоматично за допомогою спеціальної програми (алгоритму) [21]. Умови угоди між покупцем і продавцем записуються безпосередньо в коді. Смарт-контракти дозволяють здійснювати надійні та автономні транзакції та угоди між різними анонімними сторонами без участі

центрального органу, правової системи чи зовнішнього механізму контролю виконання. На рисунку 1.5 наведено схему роботи smart contract.

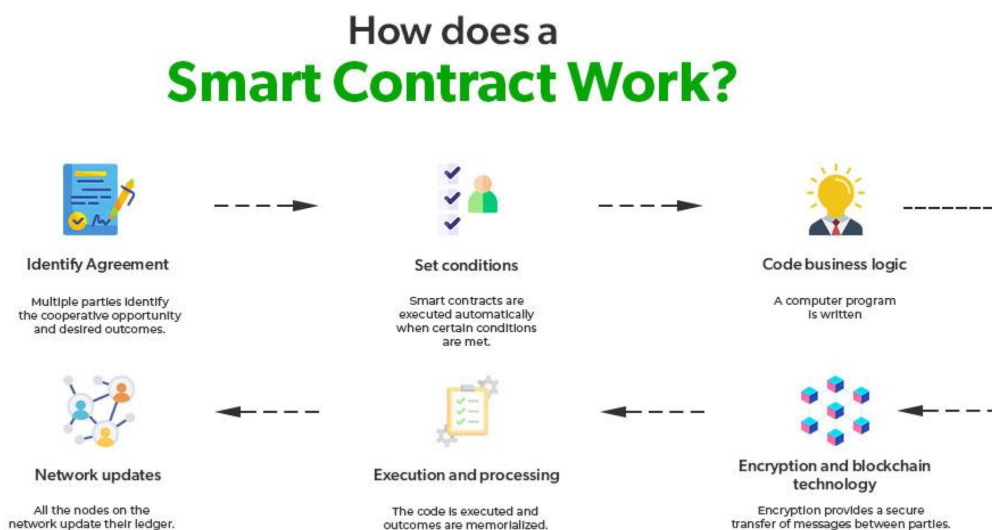

Рисунок 1.5 – Схема роботи смарт-контракту

Об'єктом смарт-контракту може бути лише те, що знаходиться в його середовищі існування, або об'єкт, до якого забезпечено прямий доступ без участі людини. Умови смарт-контракту повинні мати математичний опис, який може бути запрограмований в середовищі його існування, і визначає логіку виконання пунктів угоди.

Схоже на традиційні контракти, смарт-контракти визначають правила та штрафи навколо угоди, автоматично забезпечуючи їх виконання. Вони є ключовим елементом екосистем на основі блокчейну, таких як Ethereum та Cardano, і забезпечують надійні, автономні, децентралізовані та прозорі угоди. Вони також допомагають у зменшенні або видаленні посередників, забезпечуючи виконання угод безпосередньо за кодом, що регулює транзакції [22]. Смарт-контракти виступають основним елементом розвитку блокчейну та децентралізованих додатків (dApps). Їх переваги включають ефективність у виконанні угод і відсутність потреби в посередниках чи зовнішніх органах для забезпечення виконання умов.

Ідентичність кожної особи визначається різноманітністю факторів, серед яких ключове значення мають репутація, особисті дані та цифрові активи.

Ефективне використання цифрової ідентичності відкриває нові перспективи та можливості для людини [23]. Особливо важливо враховувати, що цифрова ідентифікація може служити як інструмент для захисту особистості від небажаних втручань і дозволяти особі обирати, з ким ділитися своєю інформацією.

Розумні контракти, застосовані у фінансовому секторі, можуть значно поліпшити надання послуг, зокрема в сферах іпотеки та позик. Забезпечуючи безперервний процес та автоматизацію, смарт-контракти здатні сприяти ефективній взаємодії сторін, що призводить до безперебійного завершення угод. Наприклад, смарт-контракт для іпотеки може контролювати розрахунки та визволяти нерухомість після повного погашення кредиту.

Управління ланцюгом поставок отримує значний заряд ефективності завдяки використанню блокчейн-розумних контрактів. Система, що використовує смарт-контракти, може забезпечити повну прозорість та відстеження товарів у ланцюжку поставок, що є важливим для оптимізації управління запасами.

Смарт-контракти також виявляють свою ефективність у галузі урядових систем голосування, забезпечуючи безпеку та відстеження голосів, що робить систему менш схильною до маніпуляцій. Це може забезпечити надійність та прозорість виборчих процесів.

Додатки для децентралізованих фінансів (DeFi) відкривають нові перспективи у фінансовому секторі, надаючи користувачам доступ до різноманітних послуг, таких як кредитування, позики та торгівля. Застосування смарт-контрактів в DeFi додатках сприяє створенню ефективних та надійних фінансових інструментів, що можуть знизити бар'єри для входу на ринок фінансових послуг.

У юридичній галузі смарт-контракти можуть стати юридично обов'язковими контрактами, що визначають бізнес-угоди [24]. Ця інноваційна технологія відкриває нові можливості для ефективного управління юридичними угодами та може допомогти у зниженні витрат на юридичні послуги та посередників.

Наявність децентралізованого середовища, конкретно визначеного предмету договору та умов виконання є необхідними елементами для успішного використання смарт-контрактів. Враховуючи це, можна стверджувати, що використання смарт-контрактів в різних галузях може призвести до значних покращень у вирішенні різноманітних завдань та оптимізації бізнес-процесів.

1.8 Переваги та недоліки смарт-контрактів

Смарт-контракти є автономними, самовиконувальними, і надійними, зберігаючись і розповсюджуючись в децентралізованій мережі блокчейнів. У порівнянні з паперовими аналогами, вони мають кілька переваг.

Автономність дозволяє учасникам укладати договори без посередників. Відсутність посередників для підтвердження чи посвідчення договору робить процес більш ефективним.

Технології блокчейну забезпечують захист документів, роблячи важкими втрату чи підміну. Децентралізованість блокчейну додає надійності, ускладнюючи взлом та підміну смарт-коду.

Смарт-контракти також вигідні у швидкості виконання. Оскільки всі параметри визначені в контрактах, вони виконуються швидше, ніж традиційні методи.

В аспекті затрат смарт-контракти ефективні, оскільки дозволяють укладати договори без посередників, що економить час та гроші.

Безперебійна робота в цифровій мережі, яку забезпечує блокчейн, робить угоди стійкими та безперервними.

Автоматизовані контракти допомагають уникнути помилок, що можуть виникати при ручному заповненні документів, і виключають людський фактор при проведенні транзакцій.

Проте, смарт-контракти не позбавлені недоліків. Їхній програмний код може містити помилки та вразливості, а побудова складного алгоритму вимагає експертності [25]. Втрата ключів доступу може створити проблеми, і неможливість зміни умов угоди може бути обмеженням. Окрім того, відсутність

законодавчої бази у багатьох юрисдикціях може ускладнити визнання та захист прав сторін, використовуючи смарт-контракти.

1.9 Аналіз існуючих рішень

Ethereum (ETH) – це блокчейн-платформа для смарт-контрактів, яку створив колектив криптографів у 2015 році. На сьогоднішній день Ethereum основною метою використання в цифровому бізнесі є створення та розгортання децентралізованих фінансових програм та стейблкоїнів[8]. Найбільший внесок. Ethereum в блокчейн-технологію – це винайдення смарт-контрактів.

Переваги Ethereum включають те, що це найбільша блокчейн-платформа для смарт-контрактів у сфері бізнесу, є захищеним рішенням для DeFi і NFT, було першопроходцем у смарт-контрактній технології, а майбутнє оновлення ETH 2.0 має підвищити продуктивність мережі.

Серед недоліків Ethereum можна відзначити обмеження швидкості, що часто призводить до перевантаження мережі, високі комісії для розробників, які роблять транзакції DEX і DeFi дорогими, неорганізований і повільний розвиток, використання PoW, яке сприяє зміні клімату, і однорівневий протокол, який обробляє обчислення і розрахунки одночасно.

Cardano – це інноваційна блокчейн-мережа з доказом частки (PoS), яка перетворюється в платформу для розробки децентралізованих додатків (DApp) з реєстром багатьох активів і перевіреними смарт-контрактами. Побудований з використанням формальних методів розробки для забезпечення надійності, Cardano має за мету досягти масштабованості, сумісності та стійкості. Cardano розроблено для використання великомасштабних DApps, які будуть основою економіки майбутнього.

Cardano відрізняється від інших блокчейнів тим, що його розробка відбувається відкрито і прозоро. Усі дослідження та технічні специфікації, що стоять за Cardano, опубліковані для загального використання. Технологію розробляє IOHK, наглядає за розвитком та просуванням Cardano – Cardano Foundation, а комерційне впровадження забезпечує Emurgo.

Cardano використовує академічні дослідження, формальні методи розробки, мову програмування Haskell для безпечного програмування, і має протокол PoS, який забезпечує високий рівень безпеки та використовує менше енергії порівняно з PoW. Крім того, Cardano впроваджує безперебійне оновлення замість традиційних хардфорків, що дозволяє плавно переходити на новий протокол, зберігаючи історію блоків [26]. Мережу Cardano підтримує майже 3000 розподілених пулів активів, забезпечуючи високий рівень децентралізації. Крім того, Cardano створює функціональне середовище для бізнес-використання, дозволяючи розробникам створювати розумні контракти і DApps, що працюють незалежно від транзакцій токенів ADA.

Solana – це блокчейн-платформа, яка була розроблена для підтримки розподілених додатків (DApps) та криптовалют. Вона була запущена в 2020 році командою розробників із США. Основна мета Solana – надати швидку, ефективну та масштабовану інфраструктуру для децентралізованих застосунків та фінансових послуг.

Висока швидкість та масштабованість: Solana використовує комбінацію технічних інновацій, таких як технологія Proof of History (PoH) та консенсус Proof of Stake (PoS), щоб забезпечити високу продуктивність та швидкість обробки транзакцій. Це дозволяє платформі масштабуватися і обробляти тисячі транзакцій за секунду.

Завдяки своїм технічним рішенням, Solana може забезпечувати низькі вартості транзакцій, що робить її привабливою для користувачів та розробників.

Solana використовує внутрішню криптовалюту SOL для здійснення транзакцій та участі в консенсусі. SOL також може використовуватися для отримання прав на управління мережею та голосування за покращення.

Децентралізовані фінанси (DeFi) та Додатки Solana виводить на новий рівень можливості для DeFi-проектів та DApps завдяки своїй високій продуктивності та швидкій обробці транзакцій.

Solana здобула популярність серед розробників та користувачів завдяки своїм технічним рішенням, спрямованим на вирішення проблем швидкості та масштабованості, що часто виникають у блокчейн-системах.

1.10 Висновок до першого розділу

В першому розділі кваліфікаційної роботи розглянуто поняття блокчейну. Описано його історію, виявлено переваги та недоліки. Дано визначення поняттю ноди, описано сферу її застосування. Надано опис методів досягнення консенсусу, а саме Proof of Work (PoW) та Proof of Stake (PoS). Розглянуто ліквідні пули та інноваційні технології для виборів. Описано смарт-контракти, детально проаналізовано причину їх винекнення, переваги та недоліки у порівнянні із стадиційними контрактами. Проаналізовано існуючі рішення у блокчейнах Ethereum, Cardano та Solana.

2 ТЕХНОЛОГІЯ БЛОКЧЕЙН. ЗАСТОСУВАННЯ ПЛАТФОРМИ SOLANA ТА МОВИ ПРОГРАМУВАННЯ RUST ДЛЯ РОЗРОБКИ СМАРТ-КОНТРАКТІВ

2.1 Блокчейн Solana

Solana – це блокчейн-платформа, яка виникла з метою розв'язання проблем швидкості та масштабованості, що стали актуальними для багатьох блокчейнів [27]. Цифрові технології стали основою для створення нових продуктів, цінностей, властивостей і, відповідно, закладають підвалини для отримання конкурентних переваг на більшості ринків [28]. Саме тому, заснована у 2020 році, Solana пропонує технологічні рішення, спрямовані на поліпшення продуктивності та забезпечення ефективного використання ресурсів мережі. На рисунку 2.1 наведено візуальний опис роботи Solana.

Рисунок 2.1 – Візуальний опис роботи Solana

Proof of History (PoH): Однією з ключових інновацій у блокчейні Solana є використання механізму Proof of History (PoH). Цей механізм дозволяє перед тим, як транзакція включається до блоку, фіксувати точний час та порядок подій.

Таке передзаписування подій спрощує процес консенсусу, що робить транзакції більш ефективними та забезпечує швидке включення їх до блокчейну.

До переваг Solana можна віднести низку факторів:

– Низькі комісії та витрати: Завдяки високій продуктивності та швидкому часу обробки транзакцій, Solana може запропонувати низькі комісії [29]. Це робить платформу більш доступною для користувачів та розробників, особливо в умовах, коли інші блокчейни можуть стикатися із завищеними витратами на транзакції.

– Ефективне використання часу та ресурсів: Використання PoH і Tower BFT дозволяє Solana оптимально використовувати час та ресурси мережі. Це робить можливим обробку великої кількості транзакцій за короткий час, забезпечуючи високу пропускну здатність.

– Підтримка різноманітних додатків: Висока швидкість та низькі витрати роблять Solana привабливою платформою для розробників додатків, особливо тих, які потребують швидкої та вартісно-ефективної обробки транзакцій, таких як фінансові додатки, ігри та різноманітні децентралізовані сервіси.

– Багатофункціональність середовища: Solana спроектована так, щоб створювати найбільш багатофункціональне середовище для фінансових, ділових та комерційних операцій [30]. Це означає, що платформа не обмежується лише обробкою одного типу валюти чи одного типу транзакцій. Вона підтримує інтеракцію з кількома видами валют та видів транзакцій, роблячи її універсальною для різноманітних фінансових застосунків.

– Міжланцюгові перекази та токени: Solana розробляється для підтримки міжланцюгових переказів, що дозволяє взаємодіяти з різними блокчейнами. Це важливо для ефективного обміну активами та інформацією між різними блокчейнами, сприяючи інтеграції та забезпечуючи більш широкі можливості для користувачів та розробників.

– Підтримка різних мов смарт-контрактів: Solana пропонує гнучкість у виборі мов програмування для смарт-контрактів [31]. Це робить платформу більш доступною для розробників, оскільки вони можуть використовувати ті

мови програмування, з якими вони вже знайомі. Такий підхід сприяє розвитку різноманітних додатків та послуг на основі блокчейну.

– Взаємодія з централізованими установами: Solana розробляється також з урахуванням можливості взаємодії з централізованими банківськими установами. Це стає важливим аспектом для того, щоб забезпечити легітимність та зручність використання платформи у фінансовому секторі.

– Доказ частки (Proof of Stake, PoS): Solana використовує консенсус Proof of Stake, що вказує на те, що вагомість у вирішенні консенсусу надається учасникам мережі, які мають певну кількість монет [32]. Цей підхід є більш екологічно чистим порівняно з Proof of Work (PoW), який використовується, наприклад, у Bitcoin.

– Самодостатність системи: Стійкість Solana означає, що система розроблена таким чином, щоб бути самодостатньою та резилієнтною. Це включає в себе здатність протистояти атакам та забезпечувати безпеку мережі. Доказ частки дозволяє спільноті самостійно контролювати та управляти системою, що робить її менш вразливою до централізованих атак.

– Заохочення участі спільноти: Щоб стимулювати зростання та розвиток децентралізованого способу, Solana розроблено так, щоб дозволяти спільноті активно брати участь у підтримці та розвитку мережі. Спільнота може запропоновувати та впроваджувати покращення, а також брати участь у процесі забезпечення стійкості та безпеки мережі.

– Система казначейства: Solana використовує систему казначейства, щоб підтримувати стійкість та розвиток мережі. Це означає, що фінансування для розвитку та підтримки мережі забезпечується з різних джерел, таких як нові монети, які щойно були випущені, відсотки винагород пулу ставок та комісії за транзакції. Цей підхід дозволяє постійно поповнювати резерви, не потребуючи централізованого фінансування.

– Гарантія високої пропускної здатності: Однією з ключових особливостей Solana є здатність обробляти велику кількість транзакцій без втрати продуктивності мережі.

– Використання різних методів масштабування: Solana впроваджує різні методи для забезпечення масштабованості, такі як стиснення даних [33]. Це дозволяє оптимізувати обсяг даних, що передаються в мережі, та забезпечує більш ефективне використання ресурсів.

– Технологія Hydra: Solana працює над впровадженням технології Hydra, яка є концепцією бічного ланцюга. Це дозволяє мережі функціонувати кількома бічними ланцюгами паралельно, що підвищує загальну масштабованість мережі та дозволяє обробляти ще більше транзакцій одночасно.

– Розвиток технології блокчейн: Solana ставить перед собою завдання розвивати технологію блокчейн в контексті стійкості та взаємодії з іншими блокчейнами та фінансовими установами.

2.2 Мова програмування Rust

Rust – це мова програмування загального призначення, яка виникла у 2010 році та розвивалася компанією Mozilla. Вона визначається своєю високою ефективністю, надійністю та здатністю до паралельного програмування [34]. Rust відрізняється від інших мов програмування завдяки вбудованій системі управління пам'яттю, яка забезпечує високий рівень безпеки виконання програм та запобігає багатьом типам помилок, таким як витоки пам'яті чи невизначені поведінки. Мова володіє сучасним синтаксисом та підтримує функціональне та об'єктно-орієнтоване програмування. На рисунку 2.2 наведено принцип роботи Rust.

Рисунок 2.2 – Принцип роботи Rust

Rust використовує концепцію власності для керування пам'яттю. Кожна змінна має свою власність, і тільки один власник може існувати в конкретний момент часу. Також до особливостей цієї мови можна віднести:

– Позички та Посилання (Borrowing): Замість передачі власності, Rust використовує позички, щоб дозволити функціям користуватися значеннями без створення копій. Це сприяє уникненню зайвого використання пам'яті.

– Структури даних: Rust підтримує визначення структур даних, що дозволяють об'єднувати різні типи даних в один об'єкт.

– Матчінг (Pattern Matching): Система зразків у Rust дозволяє зручно обробляти різні випадки значень за допомогою конструкції match.

– Винятковий обробка (Error Handling): Використання Result типу даних для представлення можливості виняткової ситуації, що дозволяє елегантно обробляти помилки.

– Система властивостей (Traits): Traits дозволяють визначати спільну функціональність для різних типів даних, сприяючи використанню поліморфізму.

– Шаблони (Generics): Rust підтримує шаблони для створення функцій та структур, які можуть працювати з різними типами даних.

– Безпека та Уникнення Збоїв (Safety and Concurrency): Rust акцентує на безпеці та уникненні збоїв, що робить її особливо підходящою для системного програмування та паралельної роботи.

Ці основні концепції допомагають розробникам створювати безпечні та ефективні програми, особливо в областях, де важлива швидкість та низький рівень системи.

У мові програмування Rust власники – це концепція, яка дозволяє керувати ресурсами (наприклад, пам'яттю) забезпечуючи виключення гонок за даними та інші проблеми [35]. Замість того, щоб користуватися списками для відстеження, хто має доступ до ресурсів, Rust використовує систему власності, позбавляючи програмістів необхідності ручного введення списків або збірників.

Власники в Rust працюють наступним чином:

– Правило одного власника (Rule of One Owner): Кожен ресурс (наприклад, шматок пам'яті) може мати лише одного власника. Це означає, що коли об'єкт входить в область видимості власника, тільки він може ним володіти.

– Передача власності (Ownership Transfer): Якщо ви передаєте власність об'єкта іншій змінній або функції, власність попереднього власника автоматично втрачається [36]. Це може відбуватися через присвоєння або передачу параметрів функції.

– Видалення власності (Dropping Ownership): Коли об'єкт виходить за межі своєї області видимості або власник звільняє його, викликається функція "drop", яка відповідає за звільнення ресурсів.

Ці концепції дозволяють уникнути багатьох проблем, пов'язаних з керуванням ресурсами, та забезпечують високий рівень безпеки пам'яті в програмах, написаних на мові Rust.

Принцип "власності" та "позичання" робить Rust ефективною та безпечною мовою для програмування системного рівня, де керування пам'яттю є критичним аспектом [37]. Rust вдосконалює можливості роботи з пам'яттю, запобігаючи типовим проблемам, таким як гонки пам'яті, без пожертви швидкодії [38].

Некероване переповнення буфера (buffer overflow) – це тип атаки на програмне забезпечення, коли дані записуються за межі виділеного для них буфера пам'яті [39]. Це може призвести до перезапису важливих даних, порушення роботи програми або навіть виконання зловмисного коду. У мові програмування Rust існують вбудовані заходи безпеки, які роблять цей вид атак важким.

Однією з основних властивостей мови Rust є система власності та безпеки пам'яті. Завдяки цьому, в Rust можна писати безпечний код, який уникне багатьох типових помилок, включаючи переповнення буфера [40].

Власні типи даних: В Rust ви можете визначати свої власні типи даних, що робить код більш експресивним та зрозумілим, а також дозволяє уникнути типових помилок.

Власніцька система: Кожен об'єкт в Rust має свого "власника", існує лише один власник для кожного об'єкта [41]. Це унеможливлює некероване видалення або модифікацію даних з боку некоректного коду.

Власності та позики: Rust використовує систему властностей та позик, яка дозволяє ефективно працювати з пам'яттю, уникнути двоїстого видалення та забезпечити безпеку використання пам'яті.

Безпечні абстракції над пам'яттю: Rust надає безпечні абстракції над роботою з пам'яттю, такі як сриви, які роблять некероване переповнення буфера менш ймовірним.

Інструменти аналізу коду: Rust інтегрує інструменти аналізу коду, такі як borrow checker, які перевіряють власність та правильність використання пам'яті ще до компіляції, допомагаючи виявляти можливі помилки.

Однак, незважаючи на всі ці заходи безпеки, важливо слідкувати за правильністю та безпекою вашого коду при розробці. Застосовуючи практики безпеки та слідкуючи за рекомендаціями спільноти, ви зможете створювати надійний та безпечний код у мові програмування Rust.

Імутабельність в Rust вказує на те, що дані не можуть бути змінені після їхнього створення [42]. Це допомагає уникнути проблем з паралельним виконанням та непередбаченими змінами даних в коді.

Паралельність в Rust дозволяє виконувати багато задач одночасно, щоб використовувати багатоядерні процесори ефективно. Rust надає механізми, такі як потоки та актори, для роботи з паралельністю безпечно та ефективно [43]. Це сприяє підвищенню продуктивності програм та їхній готовності для використання в багатоядерних середовищах.

Опціональні типи (Option): Опціональний тип в Rust використовується для представлення випадків, коли значення може бути присутнім або відсутнім. Замість використання нульових покажчиків, ви використовуєте Option, щоб виразити цю відсутність. Це дозволяє уникнути деяких типових помилок, пов'язаних із нульовими покажчиками.

Система власності та запозичення: У Rust, кожен об'єкт може мати лише одного "власника" в конкретний момент часу. Це дозволяє визначити, як довго і

яким чином використовується пам'ять. Власність переходить від одного власника до іншого, і це допомагає уникнути багатьох проблем, включаючи нульові покажчики та викидання пам'яті.

Ці підходи дозволяють Rust створити безпечну та надійну систему програмування, де програміст має контроль над використанням пам'яті, при цьому уникнені небезпечні практики, пов'язані з нульовими покажчиками в інших мовах програмування.

У Rust системи контролю рівня системи є набором засобів, які дозволяють програмістам контролювати доступ до пам’яті та ресурсів системи. Ці засоби допомагають запобігти помилкам пам’яті та іншим проблемам безпеки.

Однією з найважливіших систем контролю рівня системи в Rust є типізація [44]. Типізація гарантує, що дані, що зберігаються в змінній, відповідають типу змінної. Це допомагає запобігти помилкам, таким як завантаження значення в змінну неправильного типу.

Межа безпеки пам’яті: Границі безпеки пам’яті запобігають програмістам доступу до пам’яті поза областю, на яку вони мають право [45]. Це допомагає запобігти помилкам пам’яті, таким як переповнення буфера.

Контроль доступу до ресурсів: Контроль доступу до ресурсів дозволяє програмістам контролювати доступ до ресурсів системи, таких як файли та сокети. Це допомагає запобігти помилкам, таким як закриття відкритого файлу або використання заблокованого сокета.

Мультипотікування: Rust підтримує мультипотікування, що дозволяє програмістам запускати кілька потоків виконання одночасно [46]. Rust також підтримує ряд систем контролю рівня системи для запобігання помилок, пов’язаних з мультипотікуванням, таких як мертві замки та гонка станів.

Системи контролю рівня системи в Rust є потужним інструментом, який може допомогти програмістам створювати безпечні та надійні програми.

Cargo – це пакетний менеджер і система збірки для мови програмування Rust. Він став невід'ємною частиною екосистеми Rust і грає ключову роль у встановленні, зборці та управлінні залежностями проекту.

Cargo дозволяє швидко створити новий проект з установкою основних файлів, таких як Cargo.toml (файл конфігурації проекту) та початкова структура каталогів [47]. За допомогою Cargo можна здійснювати збірку проекту, використовуючи команду cargo build. Це автоматично знаходить та встановлює залежності проекту, а також компілює його виконуваний файл. Cargo вбудовано підтримує тестування проекту за допомогою команди cargo test. Він автоматично визначає тести у проекті та виконує їх.

Файл Cargo.toml містить інформацію про залежності проекту. Cargo автоматично завантажує та встановлює ці залежності при першому запуску або під час оновлення [48]. За допомогою команди cargo update можна оновити залежності проекту до останніх версій, враховуючи обмеження версій, вказані в файлі Cargo.toml. Cargo автоматично керує версіями проекту та його залежностей. Це робить управління версіями бібліотек інтуїтивно зрозумілим та ефективним.

Cargo дозволяє легко публікувати ваш проект чи бібліотеку на Crates.io – центральний реєстр пакетів для Rust. Cargo також надає інші команди для виконання завдань, таких як відладка (cargo run), генерація документації (cargo doc), перевірка стилю коду (cargo fmt), тощо. Cargo робить процес розробки на мові програмування Rust зручним та ефективним, забезпечуючи широкі можливості управління проектами та залежностями.

2.3 Середовище для програмування Visual Studio Code

Visual Studio Code (VSCode) – це безкоштовний текстовий редактор, розроблений компанією Microsoft. Його головною метою є надання зручного та потужного інструменту для розробки програмного забезпечення на різних мовах програмування. На рисунку 2.3 наведено принцип роботи Visual Studio Code.

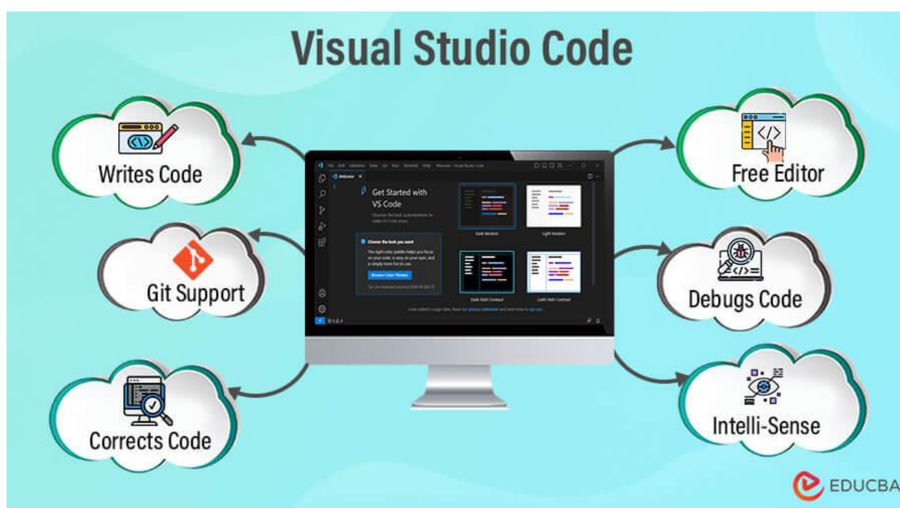

Рисунок 2.3 – Принцип роботи Visual Studio Code

Visual Studio Code є повністю безкоштовним і доступним для завантаження на офіційному веб-сайті або через репозиторій на GitHub. Безкоштовність робить його доступним для широкого кола користувачів, незалежно від їхнього рівня навичок чи фінансових можливостей [49]. Це робить VSCode популярним серед розробників, які шукають надійний і продуктивний інструмент для написання коду.

Однією з ключових особливостей VSCode є те, що він є відкритим кодом. Це означає, що весь вихідний код редактора доступний для перегляду, модифікацій та вдосконалення для будь-якого, хто бажає внести свій внесок. Репозиторій VSCode розміщений на платформі GitHub, і будь-хто може створити власний форк проекту, внести зміни та надіслати їх на розгляд для включення в основний код.

Завдяки відкритості вихідного коду, VSCode має активну та велику спільноту розробників. Це означає, що користувачі можуть з легкістю знаходити рішення для своїх проблем, спілкуватися і обмінюватися ідеями, а також приймати участь у вдосконаленні редактора [50]. Відкритий код сприяє інноваціям і прискорює розвиток VSCode, оскільки багато розробників приносять свій внесок у формі нових функцій, виправлень помилок та покращень продуктивності.

VSCode підтримує велику кількість розширень, які дозволяють користувачам налаштовувати редактор під свої потреби. Це також відкритий код

і доступний для внесення змін. Це створює потужну екосистему інструментів для розробки, яка легко розширюється завдяки активній спільноті.

Усі ці аспекти – безкоштовність, відкритий код та активна спільнота – роблять VSCode привабливим інструментом для розробників у всьому світі, незалежно від їхнього досвіду чи галузі роботи. VSCode використовує концепцію розширень, яка дозволяє розробникам створювати та встановлювати розширення для роботи з різними мовами програмування. На рисунку 2.4 наведено оновлення версії Visual Studio Code.

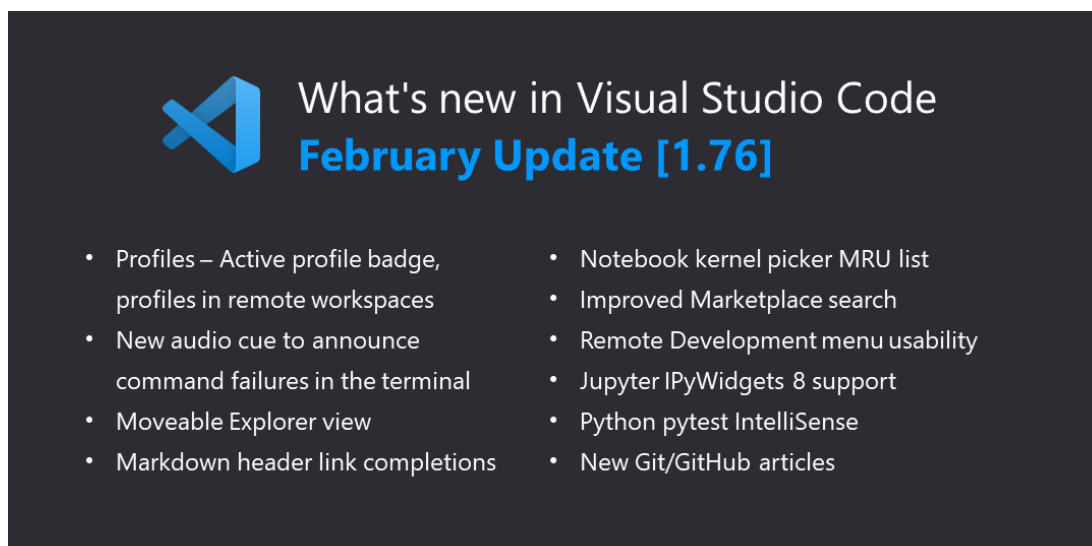

Рисунок 2.4 – Оновлення версії Visual Studio Code.

Розширення можуть включати функції, такі як підсвічування синтаксису, автодоповнення коду, підтримка відладки, інтеграція зі системами контролю версій та інші [51]. VSCode підтримує широкий спектр мов програмування, включаючи, але не обмежуючись, JavaScript, TypeScript, Python, Java, C#, C++, Ruby, PHP, HTML, CSS, Go, Kotlin, Swift, і багато інших.

Розширення для різних мов програмування можна легко знаходити та встановлювати через маркетплейс VSCode. Розширення оновлюються та розвиваються спільнотою розробників, що робить VSCode актуальним для нових технологій та мов програмування. VSCode надає можливості відлагодження для багатьох мов, що робить процес відладки більш зручним та ефективним. Розробники можуть легко налаштовувати VSCode для роботи з конкретними

проектами та мовами програмування, використовуючи файл налаштувань "settings.json".

Інтерфейс VSCode дуже інтуїтивно зрозумілий та легкий у використанні, що робить його доступним для розробників з різним рівнем досвіду.

Розширення для vов Програмування VSCode підтримує безліч мов програмування, а розширення для цих мов надають інтегровану підтримку, таку як синтаксичне виділення, автодоповнення коду, перевірка помилок і т. д.

Розширення для роботи з git інтеграція з системами контролю версій є важливою для розробників [52]. Розширення для Git дозволяють вам взаємодіяти з репозиторіями, переглядати історію змін, вносити зміни та багато іншого.

Розширення для роботи з Docke якщо ви працюєте з контейнерами Docker, розширення для VSCode допоможе вам легко керувати контейнерами, створювати Dockerfile та виконувати інші операції.

Розширення для розробки веб-додатків VSCode підтримує розробку веб-додатків і розширення для роботи з HTML, CSS, JavaScript, TypeScript, інструментами для фронтенду та бекенду.

Розширення для вирішення задач та роботи з файлами надають можливості для пошуку, вирішення проблем та роботи з файловою системою.

Visual Studio Code Marketplace це офіційний магазин розширень для VSCode, де розробники можуть знайти, встановити та оновлювати розширення. Тут розташовано тисячі розширень для різних завдань та мов програмування. Спільнота розробників VSCode має велику та активну спільноту розробників. Вони допомагають один одному, створюють та вдосконалюють розширення, діляться порадами та знаннями.

Відкритий код та розробка спільнотою дозволяє розробникам внести власні внески, створювати власні розширення та взаємодіяти з іншими розробниками [53]. Інтеграція з іншими інструментами VSCode легко інтегрується з іншими популярними інструментами розробки, що дозволяє створювати потужні комбінації для роботи з проектами.

Розширення та екосистема VSCode забезпечують високий рівень гнучкості та персоналізації для розробників, роблячи його одним з найбільш популярних інструментів для розробки програмного забезпечення.

Інтеграція з Git в Visual Studio Code (VSCode) дозволяє зручно взаємодіяти з системою контролю версій під час розробки програмного забезпечення. Ви можете використовувати команди Git безпосередньо з інтерфейсу VSCode, що спрощує відстеження змін у вашому коді. Додатково, інтеграція забезпечує можливість перегляду історії змін, розв'язання конфліктів під час злиття гілок та інші зручні функції.

При відкритті проекту в VSCode, ви можете побачити вкладку "Source Control" у бічній панелі. Тут ви знайдете список змінених файлів, можливість створювати коміти, створювати нові гілки, та переглядати стан вашого репозиторію.

Також, VSCode надає можливість перегляду історії комітів, відгалужень, та відстеження змін прямо в редакторі коду. За допомогою вбудованого терміналу ви можете виконувати Git-команди безпосередньо з VSCode, що дозволяє вам ефективно керувати вашим проектом.

Щоб почати використовувати Git в VSCode, просто відкрийте ваш проект у редакторі, та виберіть необхідні опції Source Control. Зручний інтерфейс VSCode спрощує роботу з Git та дозволяє зосередитися на розробці вашого проекту.

Автодоповнення та підказки у Visual Studio Code (VSCode) є важливими функціями для полегшення написання коду. Коли ви вводите код, редактор намагається передбачити, що ви хочете ввести, і пропонує вам варіанти автодоповнення або надає підказки для полегшення роботи.

За допомогою цих функцій ви можете швидко та ефективно писати код, зменшуючи кількість символів, які вам потрібно ввести вручну. Крім того, це допомагає уникнути помилок і швидше ознайомлює з можливими методами та властивостями об'єктів.

VSCode використовує інтегрований алгоритм автодоповнення, який враховує контекст вашого коду, типи даних, імпорти, функції та багато іншого.

Коли ви починаєте вводити ім'я функції чи змінної, редактор надає рекомендації на основі доступного контексту.

Підказки також можуть включати інформацію про параметри функцій, типи змінних, та навіть короткі пояснення для методів чи властивостей. Це полегшує вам розуміння та використання різних API та бібліотек.

У VSCode ви можете використовувати клавішу Tab для вибору підказаного варіанта або просто продовжити введення для ігнорування підказок. Можливість швидко вибирати варіанти за допомогою гарячих клавіш дозволяє вам швидко та ефективно працювати з кодом.

Отже, завдяки функціям автодоповнення та підказок у VSCode ви можете писати код швидше, зменшуючи ймовірність помилок та отримуючи необхідну інформацію про ваш код у режимі реального часу.

Вбудований термінал у Visual Studio Code – це інтегрована консоль, яка дозволяє вам виконувати команди системи безпосередньо з середовища редагування коду. Це зручно, оскільки ви можете взаємодіяти з вашим проектом та виконувати різноманітні завдання, не виходячи з редактора.

Вбудований термінал може бути налаштований та призначений для різних оболонок (наприклад, Command Prompt, PowerShell, Bash). Ви можете легко відкрити його, натиснувши клавішу Ctrl + ` . Верхня панель терміналу також має вкладки, що дозволяють вам переключатися між різними екземплярами терміналу.

Завдяки цьому функціоналу ви можете використовувати термінал для виклику команд з консолі, управління версіями, встановлення пакетів та виконання інших завдань, пов'язаних з розробкою програмного забезпечення.

Розробка веб-застосунків у середовищі Visual Studio Code (VSCode) представляє собою важливий етап в інженерному процесі, що включає в себе комплексні завдання з написання програмного коду, відлагодження та створення функціональних веб-додатків. VSCode, як легкий та високоефективний редактор коду, забезпечує підтримку різних мов програмування, включаючи, але не обмежуючись HTML, CSS і JavaScript.

Початковий етап включає ініціалізацію нового проекту, використовуючи відповідні команди в терміналі для створення структури, встановлення залежностей та ініціалізації необхідних файлів, таких як index.html, style.css і app.js.

У роботі з VSCode активно використовуються можливості автоматичного завершення коду, спрямовані на полегшення написання коду шляхом швидкого введення структурних елементів мов HTML, CSS і JavaScript. Додатково, для полегшення інтеграції з різними фреймворками, як от React, Angular чи Vue, рекомендується використовувати розширення, доступні для VSCode.

Ефективне відлагодження коду досягається за допомогою вбудованих інструментів відлагодження VSCode. Установка точок зупинки та подальший аналіз в режимі відлагодження дозволяють виявляти та виправляти помилки в програмному коді.

Нарешті, для забезпечення систематизованості та історії розробки, рекомендується використовувати систему контролю версій, таку як Git, для ефективного ведення журналу змін у кодовій базі.

В основі сучасних інформаційних технологій лежить необхідність ефективного управління та взаємодії з контейнеризованими додатками. Зокрема, інтеграція Docker у середовище Visual Studio Code (VSCode) відкриває перед користувачем можливості зручного та ефективного управління контейнерами.

Docker – це платформа для розробки, доставки та експлуатації застосунків у контейнерах. Контейнери дозволяють упаковувати програмне забезпечення та його залежності в єдиний об'єкт, що забезпечує відокремленість та переносимість застосунків. Visual Studio Code (VSCode) надає інтеграцію з Docker, що полегшує роботу з контейнерами безпосередньо в середовищі розробки.

У VSCode ви можете використовувати розширення Docker для взаємодії з Docker-контейнерами. Ви можете створювати, запускати та відлажувати контейнери, а також керувати їх конфігурацією. Розширення надає графічний інтерфейс для керування контейнерами та образами, а також забезпечує

можливість переглядати журнали, зупиняти та видаляти контейнери безпосередньо з VSCode.

Крім того, у VSCode існують розширення для роботи з Docker Compose, яке дозволяє визначати та запускати багатоконтейнерні додатки за допомогою файлу конфігурації. Ви можете визначати всі ваші контейнери та їх налаштування у файлі docker-compose.yml, а потім керувати ними безпосередньо з VSCode.

Використання Docker у сполученні з VSCode дозволяє розробникам ефективно створювати та розгортати віртуальні середовища для їхніх додатків. Цей підхід спрощує процес розробки, тестування та впровадження, забезпечуючи консистентність середовища в усіх етапах життєвого циклу програмного продукту.

До переваг роботи з Docker у VSCode відносяться можливість легкого налаштування контейнерів, швидкий доступ до їхніх логів та консолі, а також інтеграція із засобами контролю версій. Це сприяє підвищенню продуктивності розробки та забезпечує стабільність додатків під час їхнього впровадження.

Таким чином, використання Docker у поєднанні з VSCode є стратегічно обґрунтованим кроком для розробників, орієнтованих на оптимізацію процесів розробки та підвищення якості програмного забезпечення.

2.4 Операційна система Linux

Linux – це вільна та відкрита операційна система, яка є альтернативою комерційним операційним системам, таким як Windows чи macOS. Основу Linux складає ядро Linux, створене Лінусом Торвальдсом у 1991 році. Проте важливо відзначити, що саме ядро Linux називається Linux, але колективно операційну систему частіше називають GNU/Linux, оскільки багато інструментів та утиліт, що використовуються в Linux, походять з проекту GNU.

Лінукс, операційна система з відкритим вихідним кодом, зазнала значного розвитку та взаємодії спільноти з часу свого виникнення. Цей процес почався в 1991 році, коли Лінус Торвальдс випустив першу версію ядра Linux [54]. З того

часу відбувалося постійне зростання спільноти розробників та користувачів, що сприяло вдосконаленню системи. На рисунку 2.5 наведено переваги роботи Linux.

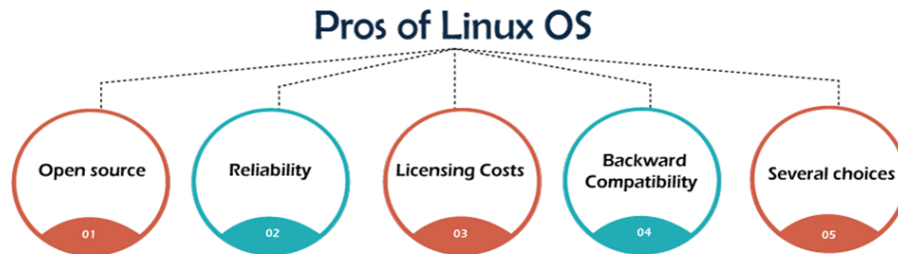

Рисунок 2.5 – Переваги роботи Linux

Розвиток Linux ґрунтується на принципах співпраці та відкритості. Розробники з усього світу приєднуються до проекту, обмінюючи ідеями та кодом через різноманітні комунікаційні канали. Цей відкритий підхід стимулює інновації та дозволяє розвивати систему на широкому фронті.

Спільнота Linux включає не тільки програмістів, а й тестувальників, дизайнерів, документаторів і простих користувачів. Кожен може внести свій внесок, навіть якщо він не є програмістом. Це робить Linux дуже доступним та гнучким для різних видів користувачів.

Комунікація в спільноті здійснюється через електронні листи, форуми, чати та інші інтернет-ресурси. Дискусії спрямовані на прийняття рішень, виправлення помилок, вдосконалення та визначення напрямку подальшого розвитку системи.

Лінукс також активно використовує систему керування версіями Git, що сприяє спільній роботі над проектом і дозволяє вносити зміни до коду, контролюючи його історію та ефективність. Цей колективний підхід дозволяє Linux швидко адаптуватися до нових технологій та вимог користувачів, забезпечуючи стабільність та безпеку операційної системи.

Linux знаходить застосування у багатьох різних областях, оскільки вона є гнучкою, стабільною та має велику спільноту користувачів і розробників. Ось докладніше про деякі з основних сфер застосування Linux:

Багато розробників використовують Linux для розробки програмного забезпечення. Однією з причин цього є наявність різноманітних інструментів для програмістів та підтримка різних мов програмування. Linux домінує в сфері серверних операцій. Від веб-серверів (таких як Apache, Nginx) до баз даних (таких як MySQL, PostgreSQL) і серверів електронної пошти, багато серверів працюють під управлінням Linux.

Багато хмарних платформ (таких як Amazon Web Services, Google Cloud Platform, Microsoft Azure) використовують Linux в якості операційної системи для своїх серверів та сервісів. Linux широко використовується в ембедованих системах, таких як маршрутизатори, телевізори, інтернет-роутери, системи безпеки та інші пристрої Інтернету речей.

Багато університетів та освітніх установ використовують Linux у своїх лабораторіях і курсах, оскільки вона є безкоштовною та надає можливість доступу до відкритого коду. Linux широко використовується в області мережевої безпеки, маючи ряд інструментів і можливостей для забезпечення безпеки мереж та даних.

Деякі операційні системи для медіацентрів, такі як Kodi, використовують ядро Linux. Linux також використовується для розробки ігор та віртуальної реальності. У великих дата-центрах Linux використовується для оптимізації ресурсів та зменшення витрат енергії, особливо на серверах, які працюють в режимі безперервної роботи.

Kodi – це відкрите програмне забезпечення для медіацентрів, яке почало свій шлях як Xbox Media Center (XBMC). Kodi розробляється командою добровольців і надає можливості для відтворення та стрімінгу різних медіа-вмісту, такого як відео, музика, фотографії та телевізійні програми.

Операційна система Kodi може працювати на різних платформах, включаючи Windows, macOS, Linux, Android та інші. Вона має зручний інтерфейс, що дозволяє користувачам легко навігувати та вибирати різні опції. Крім того, Kodi підтримує різні додатки та розширення, що дозволяє розширювати його функціональні можливості.

Важливою особливістю Kodi є його здатність працювати як централізований медіацентр, об'єднуючи в собі велику кількість різних джерел контенту. Це дозволяє користувачам зручно отримувати доступ до відео, музики та інших медіафайлів з різних джерел, включаючи локальні диски, мережеві пристрої та інтернет-ресурси.

Kodi також підтримує різні формати відео та аудіо, а також може використовуватися для перегляду зображень та слайд-шоу. Його гнучкість та розширюваність роблять його популярним вибором серед користувачів, які шукають універсальний медіацентр для розваг та розваг.

Загалом, Kodi представляє собою потужний інструмент для організації та відтворення медіа-контенту, що надає широкі можливості для налаштування та персоналізації залежно від потреб користувача.

Відкритий код в Linux є ключовим принципом, який лежить в основі філософії цієї операційної системи [55]. Загальна ідея полягає в тому, що користувачі мають право переглядати, змінювати і розповсюджувати вихідний код програмного забезпечення. Це надає багато переваг, таких як забезпечення безпеки, стабільності і швидкості розвитку.

У відкритому коді ви можете переглядати та аналізувати вихідний код програм, що дозволяє вам зрозуміти, як вони працюють. Ви можете вносити власні зміни або доповнення до програм, а також розповсюджувати власні версії. Це створює величезну спільноту розробників, яка сприяє інноваціям і вдосконаленню програм.

Завдяки відкритому коду, користувачі можуть впевнено використовувати програмне забезпечення, знаючи, що їхні дані залишаються конфіденційними і що вони мають повний контроль над тим, як працює їхня система. Це також робить можливим аудит програм для виявлення потенційних проблем безпеки та їхнього швидкого виправлення.

Однією з найважливіших філософських концепцій відкритого коду в Linux є ідея вільної дистрибуції. Це означає, що ви можете вільно розповсюджувати копії програмного забезпечення разом із їхнім вихідним кодом. Це забезпечує відкритий доступ до технологій для всіх, що сприяє вирівнюванню можливостей.

Крім того, відкритий код сприяє створенню стандартів та сумісності між програмами, що полегшує інтеграцію різноманітних рішень та розширює можливості користувачів. Загалом, відкритий код в Linux визначає спільноту і розвиток операційної системи, роблячи її більш демократичною та доступною для всіх.

Лінукс відомий своєю різноманітністю дистрибутивів, яка виникає завдяки відкритому характеру операційної системи [56]. Кожен дистрибутив має свою унікальну філософію, цільову аудиторію та спрямованість. На рисунку 2.6 наведено популярність дистрибутивів Linux.

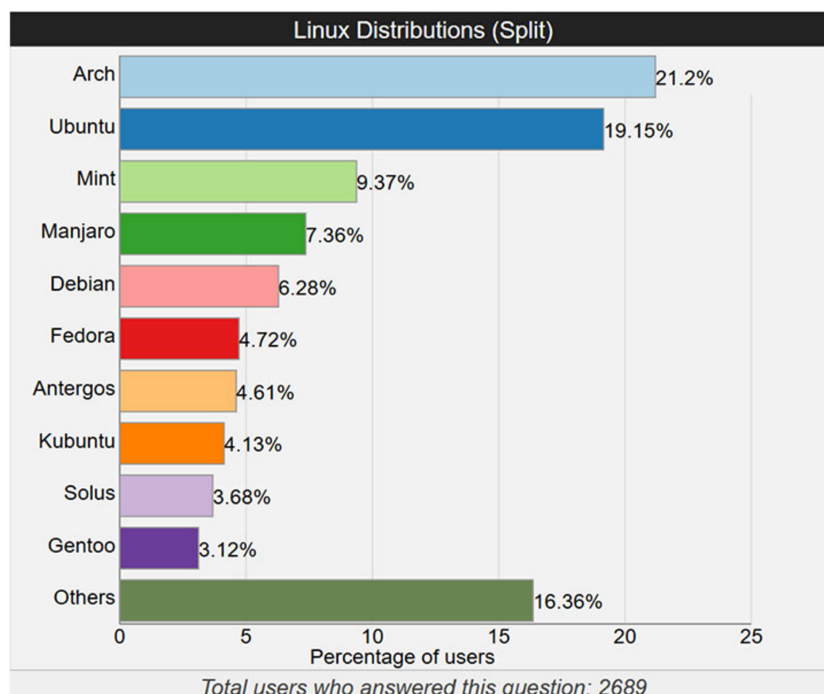

Рисунок 2.6 – Популярність дистрибутивів Linux

Дистрибутиви можуть відрізнятися за використаною базовою системою (наприклад, Debian, Arch, Red Hat), способом управління пакунками (APT, RPM, Pacman), типом середовища робочого столу (GNOME, KDE, Xfce, LXQt), а також метою використання (серверні, настільні, вбудовані системи).

Деякі дистрибутиви акцентують увагу на стабільності і надійності (наприклад, Debian), інші спрямовані на відгук від спільноти та швидку виправлення помилок (наприклад, Arch Linux). Є також спеціалізовані

дистрибутиви для використання у вбудованих системах, мережевих пристроях чи для кібербезпеки.

Один з найпопулярніших дистрибутивів Linux – Ubuntu. Ubuntu – це дистрибутив операційної системи Linux, що визначається своєю широкою популярністю та активним спільнотним співтовариством [57]. Заснований на Debian, він вигідно вирізняється своєю простотою використання та доступністю для користувачів будь-якого рівня досвіду.

Однією з ключових особливостей Ubuntu є використання середовища робочого столу GNOME, що надає зручний та інтуїтивно зрозумілий інтерфейс. Дистрибутив регулярно оновлюється, включає в себе останні версії програмного забезпечення та інструментів. На рисунку 2.7 наведено схему роботи Ubuntu.

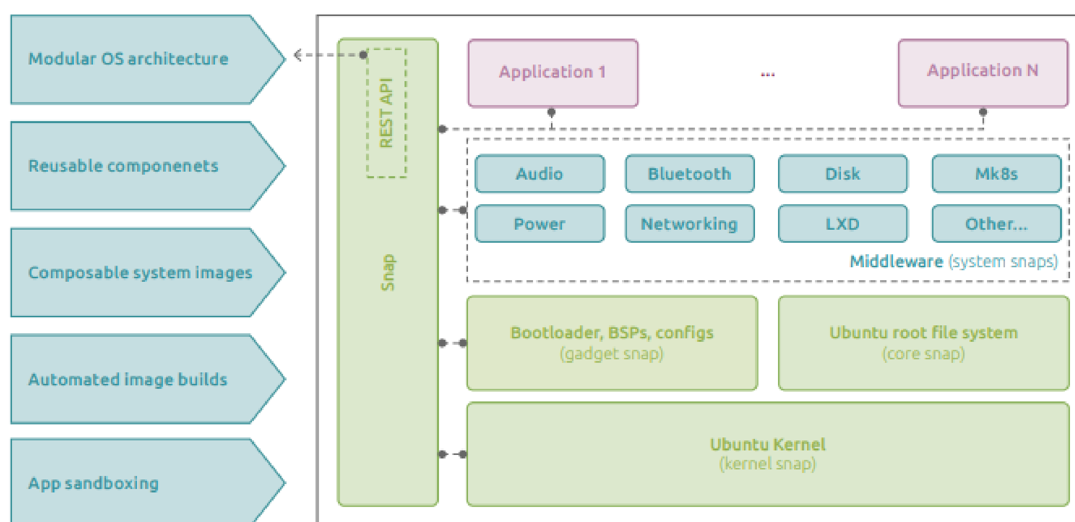

Рисунок 2.7 – Схема роботи Ubuntu

Ще однією перевагою є активна підтримка та допомога від спільноти Ubuntu, яка готова відповідати на запитання, допомагати вирішувати проблеми та надавати поради [58]. Це робить дистрибутив відмінним вибором як для початківців, так і для досвідчених користувачів Linux.

Загалом, Ubuntu визнано не лише як надійний варіант для особистих комп'ютерів, але й як дистрибутив, що знаходить широке застосування у серверному та корпоративному середовищі.

Таке різноманіття дозволяє користувачам вибирати той дистрибутив, який відповідає їхнім потребам та вимогам, що визначається як технічними характеристиками, так і філософією спільноти, яка стоїть за кожним дистрибутивом.

Мультизадачність в операційній системі Linux – це здатність обробляти одночасно декілька завдань. ОС розподіляє ресурси, такі як процесорний час і пам'ять, між різними процесами, дозволяючи їм виконуватися паралельно. Кожен процес отримує свій власний обсяг ресурсів і виконується незалежно від інших, що забезпечує швидкодіючий та ефективний режим роботи.

Стійкість в Linux вказує на здатність системи працювати безперебійно та ефективно навіть у випадку непередбачуваних ситуацій, таких як випадкові помилки апаратного забезпечення, вірусні атаки чи програмні помилки. Однією з переваг Linux є його стійкість до великої кількості ситуацій, які можуть призвести до відмови в інших операційних системах.

Linux базується на стабільному ядрі, яке активно підтримується та оновлюється спільнотою розробників. Важливою характеристикою стійкості є можливість використання системи без перерви, а також можливість відновлення роботи після виникнення проблем [59]. Linux також має ефективні засоби моніторингу та управління системою, що дозволяє швидко виявляти і виправляти проблеми

Командний рядок в Linux – це інтерфейс, який дозволяє користувачеві взаємодіяти з операційною системою шляхом введення текстових команд. Користувач може використовувати команди для виконання різноманітних завдань, таких як керування файлами, налаштування системи, мережева взаємодія та інші операції.

Введені команди передаються операційній системі для виконання. Командний рядок дозволяє вам працювати з файлами та каталогами, запускати програми, керувати процесами та налаштовувати системні параметри. Команди

можуть приймати параметри та аргументи для визначення конкретних опцій чи об'єктів, над якими ви хочете виконати операцію.

Один з важливих аспектів командного рядка – це можливість використовувати команди для автоматизації завдань, створення скриптів та виконання їх в пакетному режимі. Командний рядок також дозволяє вам взаємодіяти з системою, коли графічний інтерфейс не доступний або не ефективний.

Важливою характеристикою командного рядка є можливість швидко відповідати на зміни в системі, використовуючи команди та скрипти, що дозволяє ефективно взаємодіяти з операційною системою в текстовому режимі.

Linux побудований на принципах мультикористувацькості, що дозволяє кожному користувачеві мати свій власний обліковий запис з власними правами доступу. Це робить систему більш безпечною, оскільки обмежує доступ користувачів до чужих ресурсів. Система прав доступу в Linux гнучка та деталізована. Кожен файл і ресурс може мати визначені права на читання, запис та виконання для власника, групи і інших користувачів.

Користувачі з обмеженими правами, такими як звичайні користувачі, не мають доступу до системних налаштувань і критичних ресурсів. Привілеї адміністратора (root) використовуються обдумано та обмежено [60]. Журналізація подій і аудит безпеки дозволяє виявити аномальну активність або спроби несанкціонованого доступу до системи.

Регулярні оновлення ядра та програмного забезпечення важливі для усунення виявлених уразливостей та забезпечення актуальної захисту. Linux має вбудовані засоби для налаштування файрволу та контролю мережевого трафіку, що дозволяє ефективно захищати систему від небажаного доступу. Можливості шифрування файлових систем і комунікацій дозволяють захистити конфіденційні дані від несанкціонованого доступу.

2.5 Висновок до другого розділу

В другому розділі кваліфікаційної роботи описано особливості блокчейну Solana. Описано переваги та недоліки, а також специфіку роботи із ним. Також описано мову програмування Rust, а саме: синтаксис та систему власників, боротьбу з гонками пам'яті, безпеку, імутабельність та паралельність, боротьбу з нульовим покажчиком, боротьбу з гонками даних, системний рівень контролю, пакетний менеджер Cargo та підтримку платформ. Розглянуто VSCode і розписані головні характеристики: безкоштовність та відкритий код, підтримку багатьох мов програмування, розширення та екосистема, Інтеграція з Git. Також описана операційна система Linux і її ключові особливості: відкритий код, різноманіття дистрибутивів, мультизадачність та стійкість, командний рядок, безпеку, розвиток та спільноту, широкий спектр застосувань.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ СМАРТ-КОНТРАКТІВ У БЛОКЧЕЙНІ SOLANA

3.1 Написання смарт-контракту, який реалізує систему голосування

Створення смарт-контракту для голосування на виборах, який буде реалізований у мережі Solana, використовуючи мову програмування Rust. Solana, з її високою швидкістю транзакцій та низькими комісіями, є ідеальною платформою для розгортання децентралізованих додатків, які вимагають швидкої обробки та великої пропускної здатності [61]. У контексті голосування, це дозволяє забезпечити прозорість та безпеку виборчого процесу.

Розпочнемо розробку з визначення файлової структури. Готова структура наведена на рисунку 3.1

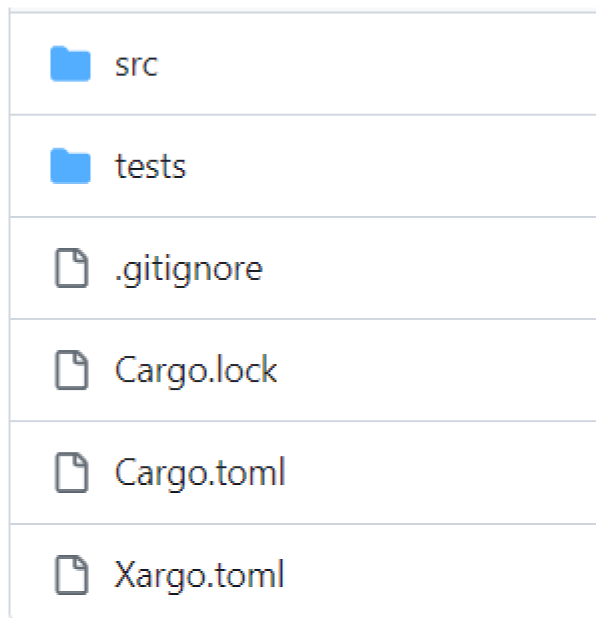

Рисунок 3.1 – Файлова структура смарт-контракту для голосування на виборах

Опишемо детально кожен елемент із файлової структури:

– src: Директорія, що містить вихідний код смарт-контракту.

– tests: Директорія з тестами для перевірки функціоналу смарт-контракту.

– .gitignore: Файл для ігнорування певних файлів системою контролю версій Git.

– Cargo.lock: Автоматично генерований файл, що відстежує залежності у Rust.

– Cargo.toml: Файл конфігурації для Cargo, що визначає проект та його залежності.

– Xargo.toml: Розширена версія Cargo.toml для складніших конфігурацій збірки.

Оскільки основний код смарт-контракту знаходиться всередині директорії src, то її внутрішня структура наведена на рисунку 3.2.

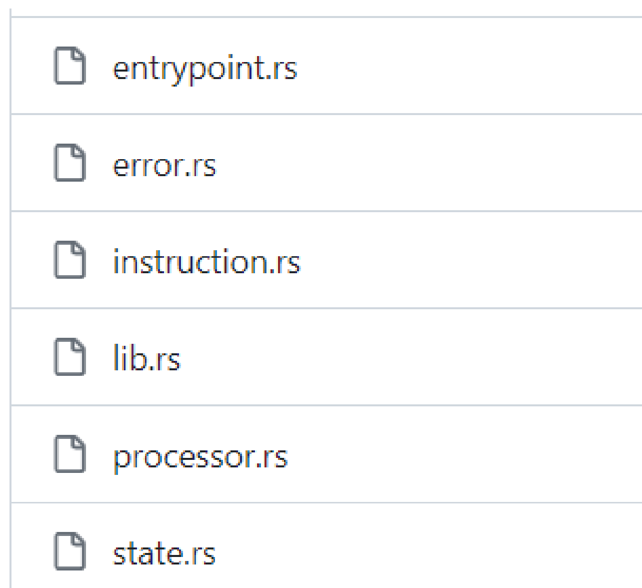

Рисунок 3.2 – Файлова структура директорії src смарт-контракту для голосування на виборах

У подальшому кожен файл буде розглянуто окремо.

3.2 Вхідна точка у смарт-контракт, який реалізує систему голосування

Написання смарт-контракту на починається з визначення вхідної точки (entrypoint). Вхідна точка є фундаментальним компонентом у розробці смарт-контрактів, оскільки вона визначає, як код контракту буде взаємодіяти з блокчейном. Код для вхідної точки розміщений у файлі entypoint.rs. На рисунку 3.3 наведено код вхідної точки для розроблюваного смарт-контракту.

```
use solana_program::{
    account_info::AccountInfo, entrypoint, entrypoint::ProgramResult, pubkey::Pubkey,
};

use crate::processor::Processor;

entrypoint!(process_instruction);

pub fn process_instruction(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    instruction_data: &[u8],
) -> ProgramResult {
    Processor::process(program_id, accounts, instruction_data)
}
```

Рисунок 3.3 – Код вхідної точки смарт-контракту для голосування на виборах

Використовуючи макрос entrypoint!, код вказує, що функція process_instruction буде використовуватися як вхідна точка. process_instruction є функцією, яка викликається при кожній транзакції або взаємодії з контрактом. Вона приймає три параметри:

– program_id (типу &Pubkey): Ідентифікатор програми, який вказує на конкретний смарт-контракт у мережі Solana.

– accounts (типу &[AccountInfo]): Масив облікових записів, які беруть участь у транзакції або взаємодії.

– instruction_data (типу &[u8]): Дані, передані у вигляді байтів, які містять інформацію про транзакцію або запит.

Функція делегує обробку даних класу Processor через виклик Processor::process(program_id, accounts, instruction_data). Це демонструє використання патерну проектування, де бізнес-логіка смарт-контракту ізольована від блокчейн-логіки, що спрощує тестування та відлагодження.

Цей підхід відображає модульність та високий рівень абстракції, характерні для сучасних практик розробки блокчейн-додатків.

3.3 Інструкції смарт-контракту, який реалізує систему голосування

У файлі instruction.rs міститься код, який визначає можливі інструкції для взаємодії із смарт-контрактом. Центральним елементом файлу є енам, наведений на рисунку 3.4.

```rust
#[derive(BorshSerialize, BorshDeserialize, Debug, Clone, PartialEq)]
pub enum VoteInstruction {
    /// Participate in vote.
    /// Accounts:
    /// 0. `[signer]` want to vote
    /// 1. `[writable]` contain info about vote that this user participate in, PDA
    /// 2. `[]` concrete vote, PDA
    /// 3. `[]` Rent sysvar
    /// 4. `[]` System program
    Vote { direction: Direction },

    /// Create a vote.
    /// Accounts:
    /// 0. `[signer]` admin
    /// 1. '[writable]' vote to create, PDA
    /// 2. '[writable]' vote counter, PDA
    /// 3. `[]` Rent sysvar, PDA
    /// 4. `[]` System program, PDA
    /// 5. '[]' Clock, PDA
    CreateVote { vote_seed: Pubkey },

    /// Delete a vote.
    /// Accounts:
    /// 0. `[signer]` admin
    /// 1. `[writable]` vote to delete, PDA
    /// 2. '[writable]' vote counter, PDA
    /// 3. '[]' Clock, PDA
    DeleteVote { admin: [u8; 32] },
```

Рисунок 3.4 – Enum VoteInstruction

Також у цьому файлі визначені конкретні функції, які перетворюють вхідні акаунти та додаткові дані у конкретні інструкції. На рисунку 3.5 наведено функції для серіалізації інструкцій.

```rust
impl VoteInstruction {
    pub fn delete(admin: &Pubkey, vote: &Pubkey) -> Instruction {
        let (vote_counter_pubkey, _) = VoteCounter::get_vote_pubkey_with_bump();
        Instruction::new_with_borsh(
            id(),
            &VoteInstruction::DeleteVote { admin: admin.to_bytes() },
            vec![
                AccountMeta::new_readonly(*admin, true),
                AccountMeta::new(*vote, false),
                AccountMeta::new(vote_counter_pubkey, false),
                AccountMeta::new_readonly(sysvar::clock::id(), false),
            ],
        )
    }

    pub fn vote(user: &Pubkey, vote: &Pubkey, direction: Direction) -> Instruction {
        let user_votes_pubkey = UserVotes::get_uservote_pubkey(user, vote);
        Instruction::new_with_borsh(
            id(),
            &VoteInstruction::Vote { direction: (direction) },
            vec![
                AccountMeta::new_readonly(*user, true),
                AccountMeta::new(user_votes_pubkey, false),
                AccountMeta::new(*vote, false),
                AccountMeta::new_readonly(sysvar::rent::id(), false),
                AccountMeta::new_readonly(system_program::id(), false),
            ],
        )
    }
```

Рисунок 3.5 – Функції серіалізації інструкцій смарт-контракту

Таким чином ці функції є одним з ключових у усьому контракті.

3.4 Помилки смарт-контракту, який реалізує систему голосування

У файлі error.rs міститься код, який визначає можливі помилки при взаємодії із смарт-контрактом. Центральним елементом файлу є енам, наведений на рисунку 3.6.

```
#[derive(Error, Debug, Copy, Clone)]
pub enum VoteError {
    #[error("User signature is required")]
    SignedRequired,

    #[error("Admin signature is required")]
    AdminRequired,

    #[error("Trying to create second vote counted")]
    DoubleCounter,

    #[error("Trying to create new vote when max count of votes already created")]
    MaxVote,

    #[error("Trying to define non-existed vote")]
    WrongVoteDefine,

    #[error("Trying to double participate in single vote")]
    DoubleParticipate,

    #[error("Trying to participate in closed vote")]
    CloseVoteParticipate,

    #[error("Wrong UserVote PDA")]
    WrongUserVotePDA,

    #[error("Wrong settings PDA")]
    WrongSettingsPDA,
}
```

Рисунок 3.6 – Enum VoteError

У цьому енамі визначені можливі помилки при взаємодії із контрактом [62]. Підхід із зазначення конкретних помилок для різних варіантів розвитку подій є дуже зручним з точки зору користувача, адже для нього стає зрозумілим що саме він зробив не так.

3.5 Структури даних смарт-контракту, який реалізує систему голосування

У файлі state.rs міститься код, у якому визначені основні структури даних, із якими взаємодіє смарт-контракт. Основні структури наведені на рисунку 3.7.

```
#[derive(BorshSerialize, BorshDeserialize, Debug)]
pub struct UserVotes {
    pub is_voted: bool,
}

impl UserVotes {
    pub fn get_uservote_pubkey_with_bump(user: &Pubkey, vote: &Pubkey) -> (Pubkey, u8) {
        Pubkey::find_program_address(
            &[&user.to_bytes(), &vote.to_bytes(), VOTE_SEED.as_bytes()],
            &id(),
        )
    }

    pub fn get_uservote_pubkey(user: &Pubkey, vote: &Pubkey) -> Pubkey {
        let (pubkey, _) = Self::get_uservote_pubkey_with_bump(user, vote);
        pubkey
    }
}

#[derive(BorshSerialize, BorshDeserialize, Debug, PartialEq)]
pub enum VoteStatus {
    Alive,
    Closed,
}

#[derive(BorshSerialize, BorshDeserialize, Debug)]
pub struct Vote {
    pub admin: [u8; 32],

    pub all_votes_for: u32,

    pub all_votes_against: u32,

    pub clock: u64,

    pub status: VoteStatus,
}
```

Рисунок 3.7 – Основні структури даних смарт-контракту для голосування на виборах

Також загальна кількість голосів зберігається у структурі VoteCounter, яка наведена на рисунку 3.8.

```
#[derive(BorshSerialize, BorshDeserialize, Debug)]
pub struct VoteCounter {
    pub counter: u8,
}

impl VoteCounter {
    pub fn get_vote_pubkey_with_bump() -> (Pubkey, u8) {
        Pubkey::find_program_address(&[SETTINGS_SEED.as_bytes()], &id())
    }

    pub fn get_vote_pubkey() -> Pubkey {
        let (pubkey, _) = Self::get_vote_pubkey_with_bump();
        pubkey
    }

    pub fn is_ok_vote_pubkey(vote_pubkey: &Pubkey) -> bool {
        let (pubkey, _) = Self::get_vote_pubkey_with_bump();
        pubkey.to_bytes() == vote_pubkey.to_bytes()
    }
}
```

Рисунок 3.8 – Структура VoteCounter

У подальшому усі структури описані у цьому файлів застосовуватимуться під час реалізації функціоналу смарт-контракту.

3.6 Основна логіка смарт-контракту, який реалізує систему голосування

У файлі processor.rs міститься код, у якому основна логіка смарт-контракту. Вона складається із багатьох функцій. Сам процес голосування реалізовано у функції process_vote. ЇЇ фрагмент наведено на рисунку 3.9.

```
fn process_vote(direction: Direction, accounts: &[AccountInfo]) -> ProgramResult {
    let acc_iter = &mut accounts.iter();
    let user_info = next_account_info(acc_iter)?;
    let participate_info = next_account_info(acc_iter)?;
    let vote_info = next_account_info(acc_iter)?;
    let rent_info = next_account_info(acc_iter)?;
    let system_program_info = next_account_info(acc_iter)?;

    if !user_info.is_signer {
        return Err(VoteError::SignedRequired.into());
    }

    let (participate_pubkey, bump_seed) =
        UserVotes::get_uservote_pubkey_with_bump(user_info.key, vote_info.key);

    if participate_pubkey != *participate_info.key {
        return Err(VoteError::WrongUserVotePDA.into());
    }

    if participate_info.data_is_empty() {
        let participate = UserVotes { is_voted: false };
        let space = participate.try_to_vec()?.len();
        let rent = &Rent::from_account_info(rent_info)?;
        let lamports = rent.minimum_balance(space);
        let signer_seeds: &[&[_]] = &[
            &user_info.key.to_bytes(),
            &vote_info.key.to_bytes(),
            VOTE_SEED.as_bytes(),
            &[bump_seed],
        ];
```

Рисунок 3.9 – Фрагмент функції process_vote

У подальшій логіці застосовується функція invoke_signed. Окрім неї також існує функція invoke. Призначення у них приблизно однакове – викликати у поточному контракті функцію з іншого. Конкретно у функції process_vote ми викликаємо system_instruction::create_account, щоб створити participate_info для конкретного користувача для участі у голосуванні. Кожен користувач може проголосувати тільки 1 раз. На рисунку 3.10 наведено описаний фрагмент коду

```
    invoke_signed(
        &system_instruction::create_account(
            user_info.key,
            &participate_pubkey,
            lamports,
            space as u64,
            &id(),
        ),
        &[user_info.clone(), participate_info.clone(), system_program_info.clone()],
        &[signer_seeds],
    )?;
    let _ = participate.serialize(&mut &mut participate_info.data.borrow_mut()[..]);
}

let mut participation = UserVotes::try_from_slice(&participate_info.data.borrow())?;
let mut vote = Vote::try_from_slice(&vote_info.data.borrow())?;

if participation.is_voted {
    return Err(VoteError::DoubleParticipate.into());
}
```

Рисунок 3.10 – Фрагмент функції із invoke_signed

Також у описуваному файлі міститься функція process_create, призначена для створення голосування. Її фрагмент наведено на рисунку 3.11.

```
fn process_create(accounts: &[AccountInfo], vote_seed: Pubkey) -> ProgramResult {
    let acc_iter = &mut accounts.iter();
    let admin_info = next_account_info(acc_iter)?;
    let vote_info = next_account_info(acc_iter)?;
    let vote_counter_info = next_account_info(acc_iter)?;
    let rent_info = next_account_info(acc_iter)?;
    let system_program_info = next_account_info(acc_iter)?;
    let clock_sysvar_info = next_account_info(acc_iter)?;

    let (vote_pubkey, bump_seed) = Vote::get_vote_pubkey_with_bump(&vote_seed);

    if vote_pubkey != *vote_info.key {
        return Err(VoteError::WrongVoteDefine.into());
    }

    if !admin_info.is_signer {
        return Err(VoteError::AdminRequired.into());
    }

    let mut vote_counter = VoteCounter::try_from_slice(&vote_counter_info.data.borrow())?;
    let time = Clock::from_account_info(clock_sysvar_info)?.slot;

    if vote_counter.counter >= MAX_VOTES {
        return Err(VoteError::MaxVote.into());
    }
```

Рисунок 3.11 – Фрагмент функції process_create

Також у описуваному файлі міститься функція process_delete, призначена для закриття активного голосування. Її фрагмент наведено на рисунку 3.12.

```
fn process_delete(accounts: &[AccountInfo], admin: [u8; 32]) -> ProgramResult {
    msg!("process_delete: admin={:?}", admin,);
    let acc_iter = &mut accounts.iter();
    let admin_info = next_account_info(acc_iter)?;
    let vote_info = next_account_info(acc_iter)?;
    let vote_counter_info = next_account_info(acc_iter)?;
    let clock_sysvar_info = next_account_info(acc_iter)?;

    if !admin_info.is_signer {
        return Err(VoteError::AdminRequired.into());
    }

    let mut vote = Vote::try_from_slice(&vote_info.data.borrow())?;
    let mut vote_counter = VoteCounter::try_from_slice(&vote_counter_info.data.borrow())?;
    let clock = Clock::from_account_info(clock_sysvar_info)?;

    if vote.admin != admin_info.key.to_bytes() {
        return Err(VoteError::AdminRequired.into());
    }
    msg!("clock.slot: {}, vote.clock: {}", clock.slot, vote.clock);
    if clock.slot - vote.clock >= TIME_TO_LIVE {
        vote.status = VoteStatus::Closed;
        vote_counter.counter -= 1;
    }

    let _ = vote.serialize(&mut &mut vote_info.data.borrow_mut()[..]);
    let _ = vote_counter.serialize(&mut &mut vote_counter_info.data.borrow_mut()[..]);

    msg!("process_delete: done");
    Ok(())
}
```

Рисунок 3.12 – Фрагмент функції process_delete

Також у описуваному файлі міститься функція process_create_counter, призначена для створення акаунта-лічильника для голосування. Вона наведена на рисунку 3.13.

```
fn process_create_counter(accounts: &[AccountInfo]) -> ProgramResult {
    let acc_iter = &mut accounts.iter();
    let admin_info = next_account_info(acc_iter)?;
    let vote_counter_info = next_account_info(acc_iter)?;
    let rent_info = next_account_info(acc_iter)?;
    let system_program_info = next_account_info(acc_iter)?;

    if !admin_info.is_signer {
        return Err(VoteError::AdminRequired.into());
    }

    let (vote_pubkey, bump_seed) = VoteCounter::get_vote_pubkey_with_bump();

    if !vote_counter_info.data_is_empty() {
        return Err(VoteError::DoubleCounter.into());
    }

    let vote_counter = VoteCounter { counter: 0 };
    let space = vote_counter.try_to_vec()?.len();
    let rent = &Rent::from_account_info(rent_info)?;
    let lamports = rent.minimum_balance(space);
    let signer_seeds: &[&[_]] = &[SETTINGS_SEED.as_bytes(), &[bump_seed]];
    invoke_signed(
        &system_instruction::create_account(
            admin_info.key,
            &vote_pubkey,
            lamports,
            space as u64,
            &id(),
        ),
        &[admin_info.clone(), vote_counter_info.clone(), system_program_info.clone()],
        &[signer_seeds],
    )?;

    let _ = vote_counter.serialize(&mut &mut vote_counter_info.data.borrow_mut()[..]);

    Ok(())
}
```

Рисунок 3.13 – Фрагмент функції process_create_counter

На початку кожної із функцій ми використовуємо ітератор акаунтів, звідки отримуємо усіх учасників транзакцій. Далі логіка кожної конкретної функції залежить від її призначення.

3.7 Тести смарт-контракту, який реалізує систему голосування

Щоб бути впевненим у тому, що контракт функціонує правильно, написано тести, які перевіряють весь можливий функціонал. Тести містяться у директорії tests. При їх написанні використовувалась асинхронна бібліотека tokio. Першим етапом при реалізації тестів для смарт-контрактів є створення тестового середовища. Фрагмент функції, яка створює це середовще, а також структура Env, яка і є цим середовищем наведені на рисунку 3.14.

```
struct Env {
    ctx: ProgramTestContext,
    admin: Keypair,
    user_01: Keypair,
    user_02: Keypair,
    user_03: Keypair,
}

impl Env {
    async fn new() -> Self {
        let program_test = ProgramTest::new("voting", id(), processor!(process_instruction));
        let mut ctx = program_test.start_with_context().await;

        let admin = Keypair::new();
        let user_01 = Keypair::new();
        let user_02 = Keypair::new();
        let user_03 = Keypair::new();

        ctx.banks_client
            .process_transaction(Transaction::new_signed_with_payer(
                &[
                    system_instruction::transfer(
                        &ctx.payer.pubkey(),
                        &admin.pubkey(),
                        1_000_000_000,
                    ),
                    system_instruction::transfer(
                        &ctx.payer.pubkey(),
                        &user_01.pubkey(),
                        1_000_000_000,
                    ),
```

Рисунок 3.14 – Фрагмент функції створення тестового середовища і декларація структури Env

Спершу було реалізовано функцію test_vote, яка перевіряє правильність голосування для простого користувача. Її фрагмент наведено на рисунку 3.15

```
// test of 1 user vote
#[tokio::test]
async fn test_vote() {
    let mut env = Env::new().await;
    let vote_seed = Pubkey::new_unique();
    let tx = Transaction::new_signed_with_payer(
        &[VoteInstruction::create_vote(&env.admin.pubkey(), &vote_seed)],
        Some(&env.admin.pubkey()),
        &[&env.admin],
        env.ctx.last_blockhash,
    );

    env.ctx.banks_client.process_transaction(tx).await.unwrap();

    let tx = Transaction::new_signed_with_payer(
        &[VoteInstruction::vote(
            &env.user_01.pubkey(),
            &Vote::get_vote_pubkey(&vote_seed),
            Direction::For,
        )],
        Some(&env.user_01.pubkey()),
        &[&env.user_01],
        env.ctx.last_blockhash,
    );
    env.ctx.banks_client.process_transaction(tx).await.unwrap();
```

Рисунок 3.15 – Фрагмент функції test_vote

Також було реалізовано функцію test_vote_third, яка перевіряє правильність голосування 3 різних користувачів, яку беруть участь в одному голосуванні. Її фрагмент наведено на рисунку 3.16.

```
// test of 3 users vote
#[tokio::test]
async fn test_vote_third() {
    let mut env = Env::new().await;
    let vote_seed = Pubkey::new_unique();
    let tx = Transaction::new_signed_with_payer(
        &[VoteInstruction::create_vote(&env.admin.pubkey(), &vote_seed)],
        Some(&env.admin.pubkey()),
        &[&env.admin],
        env.ctx.last_blockhash,
    );

    env.ctx.banks_client.process_transaction(tx).await.unwrap();

    let tx = Transaction::new_signed_with_payer(
        &[VoteInstruction::vote(
            &env.user_01.pubkey(),
            &Vote::get_vote_pubkey(&vote_seed),
            Direction::For,
        )],
        Some(&env.user_01.pubkey()),
        &[&env.user_01],
        env.ctx.last_blockhash,
    );
    env.ctx.banks_client.process_transaction(tx).await.unwrap();
```

Рисунок 3.16 – Фрагмент функції test_vote_third

Також було реалізовано функцію test_delete, яка перевіряє правильність закриття голосування після закінчення часу його роботи (кожному голосування визначений певний час існування, і коли він закінчується, голосування повинне закритись) . Її фрагмент наведено на рисунку 3.17

```
// test delete with time wait
#[tokio::test]
async fn test_delete() {
    let mut env = Env::new().await;

    let vote_seed = Pubkey::new_unique();

    let tx = Transaction::new_signed_with_payer(
        &[VoteInstruction::create_vote(&env.admin.pubkey(), &vote_seed)],
        Some(&env.admin.pubkey()),
        &[&env.admin],
        env.ctx.last_blockhash,
    );

    env.ctx.banks_client.process_transaction(tx).await.unwrap();

    env.ctx.warp_to_slot(11);

    let tx = Transaction::new_signed_with_payer(
        &[VoteInstruction::delete(&env.admin.pubkey(), &Vote::get_vote_pubkey(&vote_seed))],
        Some(&env.admin.pubkey()),
        &[&env.admin],
        env.ctx.last_blockhash,
    );
```

Рисунок 3.17 – Фрагмент функції test_delete

Також було реалізовано функцію test_delete_no_wait, яка перевіряє правильність закриття голосування після проведення транзакції із його закриття [63]. Її фрагмент наведено на рисунку 3.18

```
// test delete without time wait
#[tokio::test]
async fn test_delete_no_wait() {
    let mut env = Env::new().await;

    let vote_seed = Pubkey::new_unique();

    let tx = Transaction::new_signed_with_payer(
        &[VoteInstruction::create_vote(&env.admin.pubkey(), &vote_seed)],
        Some(&env.admin.pubkey()),
        &[&env.admin],
        env.ctx.last_blockhash,
    );

    env.ctx.banks_client.process_transaction(tx).await.unwrap();

    let tx = Transaction::new_signed_with_payer(
        &[VoteInstruction::delete(&env.admin.pubkey(), &Vote::get_vote_pubkey(&vote_seed))],
        Some(&env.admin.pubkey()),
        &[&env.admin],
        env.ctx.last_blockhash,
    );

    env.ctx.banks_client.process_transaction(tx).await.unwrap();
```

Рисунок 3.18 – Фрагмент функції test_delete_no_wait

Також було реалізовано функцію double_vote, яка перевіряє неможливість одному користувачеві двічі взяти участь у одному голосуванні. Оскільки контракт не має дозволити таку поведінку, над тестом оголошено макрос #[should_panic]. Її фрагмент наведено на рисунку 3.19.

```
// test user to double participate in 1 vote
#[should_panic]
#[tokio::test]
async fn double_vote() {
    let mut env = Env::new().await;
    let vote_seed = Pubkey::new_unique();
    let tx = Transaction::new_signed_with_payer(
        &[VoteInstruction::create_vote(&env.admin.pubkey(), &vote_seed)],
        Some(&env.admin.pubkey()),
        &[&env.admin],
        env.ctx.last_blockhash,
    );

    env.ctx.banks_client.process_transaction(tx).await.unwrap();

    let tx = Transaction::new_signed_with_payer(
        &[VoteInstruction::vote(
            &env.user_01.pubkey(),
            &Vote::get_vote_pubkey(&vote_seed),
            Direction::For,
        )],
        Some(&env.user_01.pubkey()),
        &[&env.user_01],
        env.ctx.last_blockhash,
    );
```

Рисунок 3.19 – Фрагмент функції double_vote

Також було реалізовано функцію not_admin, яка перевіряє неможливість користувачеві, який не є адміністратором, закрити голосування. Оскільки контракт не має дозволити таку поведінку, над тестом оголошено макрос #[should_panic]. Її фрагмент наведено на рисунку 3.20.

```
// test user (not admin) try to delete vote
#[should_panic]
#[tokio::test]
async fn not_admin() {
    let mut env = Env::new().await;
    let vote_seed = Pubkey::new_unique();
    let tx = Transaction::new_signed_with_payer(
        &[VoteInstruction::create_vote(&env.admin.pubkey(), &vote_seed)],
        Some(&env.admin.pubkey()),
        &[&env.admin],
        env.ctx.last_blockhash,
    );

    env.ctx.banks_client.process_transaction(tx).await.unwrap();

    let tx = Transaction::new_signed_with_payer(
        &[VoteInstruction::delete(&env.user_01.pubkey(), &Vote::get_vote_pubkey(&vote_seed))],
        Some(&env.user_01.pubkey()),
        &[&env.user_01],
        env.ctx.last_blockhash,
    );
    env.ctx.banks_client.process_transaction(tx).await.unwrap();
}
```

Рисунок 3.20 – Фрагмент функції not_admin

Також було реалізовано функцію test_create_vote, яка перевіряє правильність створення голосування. Її фрагмент наведено на рисунку 3.21.

```
// test of vote create
#[tokio::test]
async fn test_create_vote() {
    let mut env = Env::new().await;

    let vote_seed = Pubkey::new_unique();

    let tx = Transaction::new_signed_with_payer(
        &[VoteInstruction::create_vote(&env.admin.pubkey(), &vote_seed)],
        Some(&env.admin.pubkey()),
        &[&env.admin],
        env.ctx.last_blockhash,
    );

    env.ctx.banks_client.process_transaction(tx).await.unwrap();

    let acc =
        env.ctx.banks_client.get_account(Vote::get_vote_pubkey(&vote_seed)).await.unwrap().unwrap();
    let vote = Vote::try_from_slice(acc.data.as_slice()).unwrap();
    assert_eq!(vote.admin, env.admin.pubkey().to_bytes());
}
```

Рисунок 3.21 – Фрагмент функції test_create_vote

Усі описані тести відпрацювали із успішним результатом, отже смарт-контракт спроектовано та реалізовано коректно.

3.8 Написання смарт-контракту, який реалізує liquidity pool

У цьому параграфі розгляуто процес створення смарт-контракту для liquidity pool, який реалізований у мережі Solana, використовуючи мову програмування Rust.

Файлова структура цього смарт-контракту аналогічна файловій структурі смарт-контракту, який реалізує систему голосування. Окрім того опис файлу entrypoint.rs опущено, адже у всіх контрактах він виконує однакову роль.

У файлі instruction.rs міститься код, який визначає можливі інструкції для взаємодії із смарт-контрактом. Центральним елементом файлу є енам, фрагмент якого наведений на рисунку 3.22.

```
#[derive(BorshSerialize, BorshDeserialize, Debug, Clone, PartialEq)]
pub enum PoolInstruction {
    /// Provide liquidity.
    /// Accounts:
    /// 0. `[signer]` user`s account
    /// 1. `[]` user`s token x account
    /// 2. `[]` user`s token y account
    /// 3. `[]` user`s token lp account
    /// 4. `[]` pool`s token x account
    /// 5. `[]` pool`s token y account
    /// 6. `[]` mint lp token account
    /// 7. `[signer]` minter account
    /// 8. `[]` token program account, PDA
    ProvideLiquidity { x_amount: u64, y_amount: u64 },

    /// Swap tokens.
    /// Accounts:
    /// 0. `[signer]` user`s account
    /// 1. `[]` user`s token from swap account
    /// 2. `[]` user`s token to swap account
    /// 3. `[]` pool`s token from swap account
    /// 4. `[]` pool`s token to swap account
    /// 5. `[]` commision from account
    /// 6. `[signer]` minter account
    /// 7. `[]` token program account, PDA
    SwapTokens { amount: u64 },
```

Рисунок 3.22 – Фрагмент enum PoolInstruction

Також у цьому файлі визначені конкретні функції, які перетворюють вхідні акаунти та додаткові дані у конкретні інструкції. На рисунку 3.23 наведено функції для серіалізації інструкцій.

```
impl PoolInstruction {
    pub fn provide_liquidity(
        user: &Pubkey,
        admin: &Pubkey,
        x_user_token: &Pubkey,
        y_user_token: &Pubkey,
        lp_user_token: &Pubkey,
        pool_x_token: &Pubkey,
        pool_y_token: &Pubkey,
        mint_lp_token: &Pubkey,
        commision_x_token: &Pubkey,
        commision_y_token: &Pubkey,
        x_amount: u64,
        y_amount: u64,
    ) -> Instruction {
        let withdraw_pubkey = WithdrawedFee::get_withdraw_pubkey(user);
        let total_pubkey = TotalCommision::get_total_pubkey();
        Instruction::new_with_borsh(
            id(),
            &PoolInstruction::ProvideLiquidity { x_amount, y_amount },
            vec![
                AccountMeta::new_readonly(*user, true),
                AccountMeta::new(withdraw_pubkey, false),
                AccountMeta::new(*x_user_token, false),
                AccountMeta::new(*y_user_token, false),
```

Рисунок 3.23 – Функції що реалізує liquidity pool

Таким чином ці функції є одними з ключових у усьому контракті.

3.9 Структури даних смарт-контракту, який реалізує liquidity pool

У файлі state.rs міститься код, у якому визначені основні структури даних, із якими взаємодіє смарт-контракт. Основні структури наведені на рисунку 3.24.

```
#[derive(BorshSerialize, BorshDeserialize, Debug)]
pub struct WithdrawedFee {
    pub user_x_withdraw: u64,
    pub user_y_withdraw: u64,
}

impl WithdrawedFee {
    pub fn get_withdraw_pubkey_with_bump(user: &Pubkey) -> (Pubkey, u8) {
        Pubkey::find_program_address(&[&user.to_bytes(), POOL_SEED.as_bytes()], &id())
    }

    pub fn get_withdraw_pubkey(user: &Pubkey) -> Pubkey {
        let (pubkey, _) = Self::get_withdraw_pubkey_with_bump(user);
        pubkey
    }
}

#[derive(BorshSerialize, BorshDeserialize, Debug)]
pub struct TotalCommision {
    pub total_x_commision: u64,
    pub total_y_commision: u64,
}

impl TotalCommision {
    pub fn get_total_pubkey_with_bump() -> (Pubkey, u8) {
        Pubkey::find_program_address(&[&id().to_bytes(), POOL_SEED.as_bytes()], &id())
    }

    pub fn get_total_pubkey() -> Pubkey {
        let (pubkey, _) = Self::get_total_pubkey_with_bump();
        pubkey
    }
}
```

Рисунок 3.24 – Основні структури даних смарт-контракту, який реалізує liquidity pool

У подальшому усі структури описані у цьому файлів застосовуватимуться під час реалізації функціоналу смарт-контракту.

3.10 Помилки смарт-контракту, який реалізує liquidity pool

У файлі error.rs міститься код, який визначає можливі помилки при взаємодії із смарт-контрактом. Центральним елементом файлу є енам, наведений на рисунку 3.24.

```
#[derive(Error, Debug, Copy, Clone)]
pub enum PoolError {
    #[error("Trying to provide liquidity without offer both tokens")]
    ZeroProvide,

    #[error("Trying to provide more tokens than possessed")]
    OverProvide,

    #[error("Wrong ratio of providing tokens")]
    SlippageFail,

    #[error("Trying to buy more tokens than present in the pool")]
    OverBuy,

    #[error("Trying to buy more tokens than can to pay")]
    TooMuchBuy,

    #[error("Trying to withdraw more liquidity than possessed")]
    OverWithdraw,

    #[error("User signature is required")]
    SignedRequired,

    #[error("Wrong withdraw account")]
    WrongWithdraw,
}
```

Рисунок 3.24 – Enum PoolError

У цьому енамі визначені можливі помилки при взаємодії із контрактом. Підхід із зазначення конкретних помилок для різних варіантів розвитку подій є дуже зручним з точки зору користувача, адже для нього стає зрозумілим що саме він зробив не так.

3.11 Основна логіка смарт-контракту, який реалізує liquidity pool

У файлі processor.rs міститься код, у якому основна логіка смарт-контракту [64]. Вона складається із багатьох функцій. Процес надання ліквідності реалізовано у функції provide_liquidity. Її фрагмент наведено на рисунку 3.25.

```
fn provide_liquidity(accounts: &[AccountInfo], x_amount: u64, y_amount: u64) -> ProgramResult {
    msg!("Providing liquidity");

    let acc_iter = &mut accounts.iter();
    let user_info = next_account_info(acc_iter)?;
    let withdraw_info = next_account_info(acc_iter)?;
    let x_user_token_info = next_account_info(acc_iter)?;
    let y_user_token_info = next_account_info(acc_iter)?;
    let xy_lp_user_info = next_account_info(acc_iter)?;
    let pool_x_token_info = next_account_info(acc_iter)?;
    let pool_y_token_info = next_account_info(acc_iter)?;
    let mint_lp_token_info = next_account_info(acc_iter)?;
    let current_comission_x_tokem_info = next_account_info(acc_iter)?;
    let current_comission_y_tokem_info = next_account_info(acc_iter)?;
    let total_commision_info = next_account_info(acc_iter)?;
    let admin_info = next_account_info(acc_iter)?;
    let token_info = next_account_info(acc_iter)?;
    let _ = next_account_info(acc_iter)?;
    let _ = next_account_info(acc_iter)?;

    if !user_info.is_signer {
        return Err(PoolError::SignedRequired.into());
    }

    if x_amount == 0 || y_amount == 0 {
        return Err(PoolError::ZeroProvide.into());
    }
```

Рисунок 3.25 – Фрагмент функції provide_liquidity

Також у описуваному файлі міститься функція slippage_tolerance_check, призначена для перевірки того, чи обмін вказаної юзером кількості токенів не призведе до занадто значних змін у пулі. Її фрагмент наведено на рисунку 3.26.

```
pub fn slippage_tolerance_check(
    pool_x_token: Account,
    pool_y_token: Account,
    x_amount: u64,
    y_amount: u64,
) -> bool {
    let start_ratio: f64 =
        (pool_x_token.amount - x_amount) as f64 / (pool_y_token.amount - y_amount) as f64;
    let new_ratio: f64 = pool_x_token.amount as f64 / pool_y_token.amount as f64;
    let slippage = 1.0 - new_ratio / start_ratio;
    if slippage.abs() > SLIPPAGE_TOLERANCE as f64 / 100.0 {
        false
    } else {
        true
    }
}
```

Рисунок 3.26 – Фрагмент функції slippage_tolerance_check

Також у описуваному файлі міститься функція swap_tokens, призначена для реалізації інструкції обміну токенів. Її фрагмент наведено на рисунку 3.27.

```
pub fn swap_tokens(accounts: &[AccountInfo], amount: u64) -> ProgramResult {
    msg!("Swap tokens");

    let acc_iter = &mut accounts.iter();
    let user_info = next_account_info(acc_iter)?;
    let user_from_token_info = next_account_info(acc_iter)?;
    let user_to_token_info = next_account_info(acc_iter)?;
    let pool_from_token_info = next_account_info(acc_iter)?;
    let pool_to_token_info = next_account_info(acc_iter)?;
    let commision_info = next_account_info(acc_iter)?;
    let admin_info = next_account_info(acc_iter)?;
    let token_info = next_account_info(acc_iter)?;

    if !user_info.is_signer {
        return Err(PoolError::SignedRequired.into());
    }

    let pool_from_token = Account::unpack_from_slice(&pool_from_token_info.data.borrow())?;
    let pool_to_token = Account::unpack_from_slice(&pool_to_token_info.data.borrow())?;

    if amount >= pool_to_token.amount {
        return Err(PoolError::OverBuy.into());
    }
```

Рисунок 3.27 – Фрагмент функції swap_tokens

Також у описуваному файлі міститься функція swap_price_define, призначена для визначення курсу, за яким буде проведений обмін токенів. Її реалізація наведена на рисунку 3.28.

```
pub fn swap_price_define(amount: u64, pool_from_token: Account, pool_to_token: Account) -> u64 {
    ((amount as f64 / (pool_to_token.amount - amount) as f64) * pool_from_token.amount as f64)
        as u64
}
```

Рисунок 3.28 – Реалізація функції swap_price_define

Також у описуваному файлі міститься функція withdraw_liquidity, призначена для реалізації інструкції виведення ліквідності. Її фрагмент наведено на рисунку 3.29.

```
pub fn withdraw_liquidity(accounts: &[AccountInfo], amount: u64) -> ProgramResult {
    msg!("Withdraw liquidity");

    let acc_iter = &mut accounts.iter();
    let user_info = next_account_info(acc_iter)?;
    let _ = next_account_info(acc_iter)?;
    let x_user_token_info = next_account_info(acc_iter)?;
    let y_user_token_info = next_account_info(acc_iter)?;
    let xy_lp_user_info = next_account_info(acc_iter)?;
    let pool_x_token_info = next_account_info(acc_iter)?;
    let pool_y_token_info = next_account_info(acc_iter)?;
    let mint_lp_token_info = next_account_info(acc_iter)?;
    let _ = next_account_info(acc_iter)?;
    let _ = next_account_info(acc_iter)?;
    let _ = next_account_info(acc_iter)?;
    let admin_info = next_account_info(acc_iter)?;
    let token_info = next_account_info(acc_iter)?;
    let _ = next_account_info(acc_iter)?;
    let _ = next_account_info(acc_iter)?;

    if !user_info.is_signer {
        return Err(PoolError::SignedRequired.into());
    }
```

Рисунок 3.29 – Фрагмент функції withdraw_liquidity

Також у описуваному файлі міститься функція liquidity_profit, призначена для підрахунку прибутку, отриманого за рахунок надання ліквідності. Її реалізація наведена на рисунку 3.30.

```
pub fn liquidity_profit(
    amount: u64,
    total_lp: u64,
    token_x_in_pool: u64,
    token_y_in_pool: u64,
) -> [u64; 2] {
    if total_lp == 0 {
        return [0, 0];
    }
    let ratio = amount as f64 / total_lp as f64;
    [
        (token_x_in_pool as f64 * ratio) as u64,
        (token_y_in_pool as f64 * ratio) as u64,
    ]
}
```

Рисунок 3.30 – Реалізація функції liquidity_profit

Також у описуваному файлі міститься функція withdraw_fee, призначена для реалізації інструкції виведення комісії. Її фрагмент наведено на рисунку 3.31.

```
pub fn withdraw_fee(accounts: &[AccountInfo]) -> ProgramResult {
    msg!("Withdraw commision");

    let acc_iter = &mut accounts.iter();
    let user_info = next_account_info(acc_iter)?;
    let withdraw_info = next_account_info(acc_iter)?;
    let x_user_token_info = next_account_info(acc_iter)?;
    let y_user_token_info = next_account_info(acc_iter)?;
    let xy_lp_user_info = next_account_info(acc_iter)?;
    let _ = next_account_info(acc_iter)?;
    let _ = next_account_info(acc_iter)?;
    let mint_lp_token_info = next_account_info(acc_iter)?;
    let current_comission_x_tokem_info = next_account_info(acc_iter)?;
    let current_comission_y_tokem_info = next_account_info(acc_iter)?;
    let total_commision_info = next_account_info(acc_iter)?;
    let admin_info = next_account_info(acc_iter)?;
    let token_info = next_account_info(acc_iter)?;
    let rent_info = next_account_info(acc_iter)?;
    let system_program_info = next_account_info(acc_iter)?;

    if !user_info.is_signer {
        return Err(PoolError::SignedRequired.into());
    }
```

Рисунок 3.31 – Фрагмент функції withdraw_fee

3.12 Тести смарт-контракту, який реалізує liquidity pool

Для перевірки правильності роботи контракту написано тести. Вони містяться у директорії tests. Фрагмент функції, яка створює тестове середовище, а також структура Env, яка і є цим середовищем наведені на рисунку 3.32.

```
struct Env {
    ctx: ProgramTestContext,
    admin: Keypair,
    user_01: Keypair,
    user_02: Keypair,
    mint_lp_account: Keypair,
    user_01_x_token_account: Keypair,
    user_01_y_token_account: Keypair,
    user_01_lp_token_account: Keypair,
    user_02_x_token_account: Keypair,
    user_02_y_token_account: Keypair,
    user_02_lp_token_account: Keypair,
    pool_x_token_account: Keypair,
    pool_y_token_account: Keypair,
    commision_x_token_account: Keypair,
    commision_y_token_account: Keypair,
}

impl Env {
    async fn new() -> Self {
        let program_test = ProgramTest::new("pool", id(), processor!(process_instruction));
        let mut ctx = program_test.start_with_context().await;

        let admin = Keypair::new();
        let user_01 = Keypair::new();
        let user_02 = Keypair::new();

        ctx.banks_client
            .process_transaction(Transaction::new_signed_with_payer(
                &[
                    system_instruction::transfer(
                        &ctx.payer.pubkey(),
                        &admin.pubkey(),
                        1_000_000_000,
                    ),
```

Рисунок 3.32 – Фрагмент функції створення тестового середовища і декларація структури Env

Спершу було реалізовано функцію provide_liquidity_first, яка перевіряє поведінку пула, коли перший користувач надає ліквідність. Її фрагмент наведено на рисунку 3.33.

```
// test of first user providing liquidity
#[tokio::test]
async fn provide_liquidity_first() {
    let mut env = Env::new().await;

    let tx = Transaction::new_signed_with_payer(
        &[PoolInstruction::provide_liquidity(
            &env.user_01.pubkey(),
            &env.admin.pubkey(),
            &env.user_01_x_token_account.pubkey(),
            &env.user_01_y_token_account.pubkey(),
            &env.user_01_lp_token_account.pubkey(),
            &env.pool_x_token_account.pubkey(),
            &env.pool_y_token_account.pubkey(),
            &env.mint_lp_account.pubkey(),
            &env.commision_x_token_account.pubkey(),
            &env.commision_y_token_account.pubkey(),
            5,
            15,
        )],
        Some(&env.user_01.pubkey()),
        &[&env.user_01, &env.admin],
        env.ctx.last_blockhash,
    );
```

Рисунок 3.33 – Фрагмент функції provide_liquidity_first

Також було реалізовано функцію part_liquidity_withdraw, яка перевіряє поведінку пула, коли користувач хоче вивести частину своєї ліквідності. Її фрагмент наведено на рисунку 3.34.

```
// withdraw part of user`s liquidity
#[tokio::test]
async fn part_liquidity_withdraw() {
    let mut env = Env::new().await;

    let tx = Transaction::new_signed_with_payer(
        &[PoolInstruction::provide_liquidity(
            &env.user_01.pubkey(),
            &env.admin.pubkey(),
            &env.user_01_x_token_account.pubkey(),
            &env.user_01_y_token_account.pubkey(),
            &env.user_01_lp_token_account.pubkey(),
            &env.pool_x_token_account.pubkey(),
            &env.pool_y_token_account.pubkey(),
            &env.mint_lp_account.pubkey(),
            &env.commision_x_token_account.pubkey(),
            &env.commision_y_token_account.pubkey(),
            5,
            15,
        )],
        Some(&env.user_01.pubkey()),
        &[&env.user_01, &env.admin],
        env.ctx.last_blockhash,
    );

    env.ctx.banks_client.process_transaction(tx).await.unwrap();
```

Рисунок 3.34 – Фрагмент функції part_liquidity_withdraw

Також було реалізовано функцію swap, яка перевіряє поведінку пула, коли користувач хоче провести обмін токенів. Її фрагмент наведено на рисунку 3.35.

```
// test of tokens swap
#[tokio::test]
async fn swap() {
    let mut env = Env::new().await;

    let tx = Transaction::new_signed_with_payer(
        &[PoolInstruction::provide_liquidity(
            &env.user_01.pubkey(),
            &env.admin.pubkey(),
            &env.user_01_x_token_account.pubkey(),
            &env.user_01_y_token_account.pubkey(),
            &env.user_01_lp_token_account.pubkey(),
            &env.pool_x_token_account.pubkey(),
            &env.pool_y_token_account.pubkey(),
            &env.mint_lp_account.pubkey(),
            &env.commision_x_token_account.pubkey(),
            &env.commision_y_token_account.pubkey(),
            5,
            15,
        )],
        Some(&env.user_01.pubkey()),
        &[&env.user_01, &env.admin],
        env.ctx.last_blockhash,
    );

    env.ctx.banks_client.process_transaction(tx).await.unwrap();
```

Рисунок 3.35 – Фрагмент функції swap

Також було реалізовано функцію withdraw_fee_first, яка перевіряє поведінку пула, коли користувач хоче вперше вивести свій прибуток (комісію інших користувачів). Її фрагмент наведено на рисунку 3.36.

```
// user first time withdraw commision
#[tokio::test]
async fn withdraw_fee_first() {
    let mut env = Env::new().await;

    let tx = Transaction::new_signed_with_payer(
        &[PoolInstruction::provide_liquidity(
            &env.user_01.pubkey(),
            &env.admin.pubkey(),
            &env.user_01_x_token_account.pubkey(),
            &env.user_01_y_token_account.pubkey(),
            &env.user_01_lp_token_account.pubkey(),
            &env.pool_x_token_account.pubkey(),
            &env.pool_y_token_account.pubkey(),
            &env.mint_lp_account.pubkey(),
            &env.commision_x_token_account.pubkey(),
            &env.commision_y_token_account.pubkey(),
            500000,
            750000,
        )],
        Some(&env.user_01.pubkey()),
        &[&env.user_01, &env.admin],
        env.ctx.last_blockhash,
    );

    env.ctx.banks_client.process_transaction(tx).await.unwrap();
```

Рисунок 3.36 – Фрагмент функції withdraw_fee_first

Також було реалізовано функцію withdraw_fee_second_time, яка перевіряє поведінку пула, коли користувач хоче вдруге вивести свій прибуток (комісію інших користувачів). Її фрагмент наведено на рисунку 3.37.

```
// user withdraw commision second time, but new commision don`t arrived yet
#[tokio::test]
async fn withdraw_fee_second_time() {
    let mut env = Env::new().await;

    let tx = Transaction::new_signed_with_payer(
        &[PoolInstruction::provide_liquidity(
            &env.user_01.pubkey(),
            &env.admin.pubkey(),
            &env.user_01_x_token_account.pubkey(),
            &env.user_01_y_token_account.pubkey(),
            &env.user_01_lp_token_account.pubkey(),
            &env.pool_x_token_account.pubkey(),
            &env.pool_y_token_account.pubkey(),
            &env.mint_lp_account.pubkey(),
            &env.commision_x_token_account.pubkey(),
            &env.commision_y_token_account.pubkey(),
            500000,
            750000,
        )],
        Some(&env.user_01.pubkey()),
        &[&env.user_01, &env.admin],
        env.ctx.last_blockhash,
    );

    env.ctx.banks_client.process_transaction(tx).await.unwrap();
```

Рисунок 3.37 – Фрагмент функції withdraw_fee_second_time

Також було реалізовано функцію provide_new_liquidity, яка перевіряє поведінку пула, коли користувач хоче вдруге внести ліквідність [65]. Її фрагмент наведено на рисунку 3.38.

```
// test for user not immediately get new commision fees after providing liquidity
#[tokio::test]
async fn provide_new_liquidity() {
    let mut env = Env::new().await;

    let tx = Transaction::new_signed_with_payer(
        &[PoolInstruction::provide_liquidity(
            &env.user_01.pubkey(),
            &env.admin.pubkey(),
            &env.user_01_x_token_account.pubkey(),
            &env.user_01_y_token_account.pubkey(),
            &env.user_01_lp_token_account.pubkey(),
            &env.pool_x_token_account.pubkey(),
            &env.pool_y_token_account.pubkey(),
            &env.mint_lp_account.pubkey(),
            &env.commision_x_token_account.pubkey(),
            &env.commision_y_token_account.pubkey(),
            50000,
            75000,
        )],
        Some(&env.user_01.pubkey()),
        &[&env.user_01, &env.admin],
        env.ctx.last_blockhash,
    );

    env.ctx.banks_client.process_transaction(tx).await.unwrap();
```

Рисунок 3.38 – Фрагмент функції provide_new_liquidity

Усі описані тести відпрацювали із успішним результатом, отже смарт-контракт спроектовано та реалізовано коректно.

3.13 Висновок до третього розділу

В третьому розділі кваліфікаційної роботи описано створення смарт-контрактів для liquidity pool та системи голосування на виборах на блокчейні Solana за допомогою мови програмування Rust. Спроектовано основну логіку, логіку обробки помилок, написано інтеграційні тести для усього функціоналу. Наведено опис основних інструкцій для кожного із контрактів, розглянуто структури даних, необхідні для реалізації вказаних смарт-контрактів.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Вимоги щодо охорони праці при роботі з комп’ютерами.

Через масовий характер робіт, що виконуються працівниками за допомогою комп’ютера, законодавством України чітко врегульовано норми та вимоги до використання комп’ютерної техніки на підприємстві, безпосередньо й охорона праці при роботі з комп’ютером.

Приміщення, в яких планується установка та подальша робота з комп’ютером, повинні відповідати проектній документації будинку, погодженій з уповноваженими державними органами. Крім того, роботодавець повинен враховувати санітарні нормативи освітлення, вимоги до параметрів мікроклімату (температура, відносна вологість), ступеня і сили вібрації, звукового шуму і вогнестійкості приміщення, а також характеристики електромагнітного, ультрафіолетового та інфрачервоного полів. Конкретні показники зазначених санітарних норм див. в Державних санітарих правилах і нормах роботи з візуальними дисплейними терміналами електроннообчислювальних машин ДСанПІН 3.3.2.007-98, затверджених Постановою Головного державного санітарного лікаря України №7 від 10 грудня 1998 року.

Правила поширюються на умови й організацію праці при роботі з візуальними дисплейними терміналами (ВДТ) усіх типів вітчизняного та зарубіжного виробництва на основі електронно-променевих трубок (ЕПТ), що використовуються в електронно-обчислювальних машинах (ЕОМ) колективного використання та персональних ЕОМ (ПЕОМ). Так, наприклад, роботодавцю заборонено установлювати комп’ютери в приміщеннях, розташованих у підвалах будинків.

Для уникнення можливих аварій та замикань, поряд з приміщеннями, де вестиметься робота з комп’ютером (над чи під ними), також не дозволяється проведення робіт, що потребують здійснення надмірно вологих технологічних процесів [66]. Відповідне приміщення повинно бути укомплектоване системами центрального або індивідуального опалення, кондиціонування чи вентиляції

повітря. Але при установці зазначених систем, необхідно переконатись, що батареї опалення, водопровідні труби, вентиляційні кабелі тощо, надійно сховані під захисними щитками, які перешкоджатимуть можливому потраплянню робітника під напругу.

У кожній кімнаті, де обладнуватимуться робочі місця співробітників, що працюватимуть на комп’ютері, повинні бути наявні елементи природного та штучного освітлення. При цьому, на вікнах слід встановити легко регульовані жалюзі чи штори, які дозволять працівникам коригувати рівень освітлення в приміщенні. Бажано розмістити комп’ютери в кімнаті таким чином, щоб світло потрапляло на екрани моніторів з півдня чи північного сходу. З метою досягнення максимального рівня безпечності і охорони праці при роботі з комп’ютером, виробничі приміщення необхідно обладнати аптечками першої медичної допомоги, системами автоматичної пожежної сигналізації і вогнегасниками. В приміщенні, в якому разом працюють 5 або більше комп’ютерів, на видимому місці установлюється службовий вимикач, який у разі потреби дозволить повністю відключити електричне живлення кімнати.

Роботодавець, який використовує найману працю робітників, повинен забезпечити відповідність їхніх робочих місць комфортним та безпечним умовам. Розмір одного робочого місця має становити не менше 6 квадратних метрів. При необхідності, суміжні робочі місця співробітників, що працюють з комп’ютером, слід розділити перегородками висотою до 2 метрів. При визначенні достатнього розміру приміщення і робочого місця на одну особу необхідно додатково враховувати шафи, сейфи, тумби або інші предмети меблів чи обладнання, які знаходяться в кімнаті. На столі працівника можливо розмістити допоміжні для роботи пристрої (принтери, колонки, сканери), а також місця для зберігання документів, за умови, що це не обмежуватиме видимість екрану і не заважатиме працівнику. У разі надмірного шуму чи вібрації технічного обладнання, роботодавець повинен забезпечити працівників антивібраційними килимками. Робочий стілець співробітника має бути підйомно-поворотним, легко регульованим за висотою та забезпечувати належну підтримку та зручне положення спини і хребта особи. Щодня необхідно

проводити вологе прибирання приміщення, та очищати робоче місце та безпосередньо монітор комп’ютера від запиленості.

На підприємстві забороняється: проводити ремонт та технічне обслуговування комп’ютера за робочим місцем працівника; самочинно ремонтувати або намагатись здійснити технічне налагодження комп’ютера без залучення компетентних спеціалістів; складувати на робочому місці зайві документи, деталі та предмети, що не потрібні для роботи; використовувати монітори з нечітким зображенням та монітори, у яких наявні поламки екрану; працювати з матричним принтером без антивібраційного покриття та зі знятою кришкою [67]. Допускати до роботи осіб, які не пройшли затверджений на підприємстві курс охорони праці для роботи з комп’ютером, не дозволяється.

4.2 Організація охорони праці працівників на підприємстві

З метою забезпечення сприятливих для здоров'я умов праці, високого рівня працездатності, профілактики травматизму і професійних захворювань, отруєнь та відвернення іншої можливої шкоди для здоров'я на підприємствах, в установах і організаціях різних форм власності повинні встановлюватися єдині санітарногігієнічні вимоги до організації виробничих процесів, пов'язаних з діяльністю людей, а також до якості машин, обладнання, будівель та інших об'єктів, які можуть мати шкідливий вплив на здоров'я. Всі державні стандарти, технічні умови і промислові зразки обов'язково погоджуються з органами охорони здоров'я в порядку, встановленому законодавством. Власники і керівники підприємств, установ та організацій зобов'язані забезпечити в їхній діяльності виконання правил техніки безпеки, виробничої санітарії та інших вимог щодо охорони здоров'я, передбачених законодавством, не допускати шкідливого впливу на здоров'я людей (ст. 28 Основ законодавства України про охорону здоров'я).

Власник зобов'язаний створити в кожному структурному підрозділі й на робочому місці умови праці відповідно до вимог нормативних актів, а також

забезпечити дотримання прав працівників, гарантованих чинним законодавством.

З цією метою власник забезпечує функціонування системи управління охороною здоров'я, для чого створює на підприємстві підрозділи, які традиційно іменуються службою охорони праці. Типове положення про службу охорони праці затверджене наказом Державного комітету України з нагляду за охороною праці від 15 листопада 2004 р. № 255. Служба охорони праці створюється на підприємствах з кількістю працюючих 50 і більше осіб. На підприємстві з кількістю працюючих менше 50 осіб функції служби охорони праці можуть виконувати у порядку сумісництва (суміщення) особи, які мають відповідну підготовку. На підприємстві з кількістю працюючих менше 20 осіб для виконання функцій служби охорони праці можуть залучатися сторонні спеціалісти на договірних засадах, які мають виробничий стаж роботи не менше трьох років і пройшли навчання з охорони праці. Служба охорони праці підпорядковується безпосередньо роботодавцю [68]. Ліквідація служби охорони праці допускається тільки у разі ліквідації підприємства чи припинення використання найманої праці фізичною особою.

На службу охорони праці покладено виконання таких завдань. У разі відсутності впровадженої системи якості відповідно до ISO 9001, опрацювання ефективної системи управління охороною праці на підприємстві та сприяння удосконаленню діяльності у цьому напрямку кожного структурного підрозділу і кожного працівника; забезпечення фахової підтримки рішень роботодавця з цих питань; організація проведення профілактичних заходів, спрямованих на усунення шкідливих і небезпечних виробничих факторів, запобігання нещасним випадкам на виробництві, професійним захворюванням та іншим випадкам загрози життю або здоров'ю працівників; вивчення та сприяння впровадженню у виробництво досягнень науки і техніки, прогресивних і безпечних технологій, сучасних засобів колективного та індивідуального захисту працівників; контроль за дотриманням працівниками вимог законів та інших нормативно-правових актів з охорони праці, положень (у разі наявності) галузевої угоди, розділу "Охорона праці", колективного договору та актів з охорони праці, що діють у

межах підприємства; інформування та надання роз'яснень працівникам підприємства з питань охорони праці.

4.3 Інженерний захист персоналу об’єкту та населення

Функціонування на території нашої країни численних об’єктів підвищеної небезпеки, переважно в зонах з підвищеною концентрацією населення, різко посилює небезпеку великих техногенних катастроф, провокує та збільшує негативну дію особливо небезпечних стихійних явищ. Щороку втрати від таких надзвичайних ситуацій вимірюються тисячами людських життів, мільярдними збитками та невиправною шкодою для природного середовища.

Масштабність і багатогранність завдань щодо протидії сучасним природним і техногенним загрозам вимагають висококваліфікованої, технічно оснащеної, мобільної державної системи цивільного захисту.

Така система визнана складовою національної безпеки, а виконання її завдань – важливим обов’язком органів виконавчої влади всіх рівнів. Відповідно, з метою наближення до світових стандартів, від назви основного інструмента державної політики у сфері протидії наслідкам катастроф – цивільна оборона ми переходимо до назви – цивільний захист. І це не випадково [69]. Сукупність завдань, що стоять перед службами цивільної оборони багатьох країн, більше пов’язані сьогодні з проблемами мирного часу, що дозволяє говорити про цивільний захист населення і територій, а не про цивільну оборону у воєнний час.

До захисних споруд цивільного захисту належать:

– Сховище – герметична споруда для захисту людей, в якій протягом певного часу створюються умови, що виключають вплив на них небезпечних факторів, які виникають внаслідок надзвичайної ситуації, воєнних (бойових) дій та терористичних актів.

– Протирадіаційне укриття – негерметична споруда для захисту людей, в якій створюються умови, що виключають вплив на них іонізуючого опромінення у разі радіоактивного забруднення місцевості.

– Швидкоспоруджувана захисна споруда цивільного захисту – захисна споруда, що зводиться із спеціальних конструкцій за короткий час для захисту людей від дії засобів ураження в особливий період.

Для захисту людей від деяких факторів небезпеки, що виникають внаслідок надзвичайних ситуацій у мирний час, та дії засобів ураження в особливий період також використовуються споруди подвійного призначення та найпростіші укриття.

Споруда подвійного призначення – це наземна або підземна споруда, що може бути використана за основним функціональним призначенням і для захисту населення.

Найпростіше укриття – це фортифікаційна споруда, цокольне або підвальне приміщення, що знижує комбіноване ураження людей від небезпечних наслідків надзвичайних ситуацій, а також від дії засобів ураження в особливий період.

За місцем розташування сховища можуть бути вбудовані і окремо розташовані. До вбудованих відносяться сховища, які розташовані в підвальних приміщеннях будинків, а до окремо розташованих – сховища, які розташовані за межами будинків і споруд.

Для вирішення питань щодо укриття населення в захисних спорудах цивільного захисту центральні органи виконавчої влади, місцеві державні адміністрації, органи місцевого самоврядування та суб’єкти господарювання завчасно створюють фонд таких споруд.

Порядок створення, утримання фонду захисних споруд цивільного захисту та ведення його обліку визначається Кабінетом Міністрів України. Проектування, будівництво, пристосування і розміщення захисних споруд та об’єктів подвійного призначення здійснюються згідно з нормами, які розробляються відповідно до Закону України "Про будівельні норми".

Вимоги щодо утримання та експлуатації захисних споруд визначаються центральним органом виконавчої влади, який забезпечує формування та реалізує державну політику у сфері цивільного захисту.

Утримання захисних споруд цивільного захисту у готовності до використання за призначенням здійснюється суб’єктами господарювання, на балансі яких вони перебувають (у тому числі споруд, що не увійшли до їх статутних капіталів у процесі приватизації (корпоратизації), за рахунок власних коштів.

У разі використання однієї захисної споруди кількома суб’єктами господарювання вони беруть участь в утриманні споруди відповідно до укладених між ними договорів.

4.4 Критичні стани людини

Нещасні випадки і професійні захворювання є наслідком незадовільних умов праці, які виникають в процесі виробництва в результаті дії небезпечних і шкідливих факторів [70]. Виділяють такі групи факторів, які визначають стан безпеки праці: організаційні, технічні, санітарно -гігієнічні і психофізіологічні. При аналізі виробничого травматизму і професійних захворювань, враховує весь комплекс факторів, які впливають на безпеку праці.

Виробнича травма – це наслідок дії на організм різних зовнішніх, небезпечних виробничих факторів. Найчастіше виробнича травма – це результат механічного впливу при наїздах або контакті з механічним обладнанням.

Успішна профілактика виробничого травматизму та професійної захворюваності можлива лише за умови ретельного вивчення причин їх виникнення.

Організаційні причини: незадовільне функціонування, недосконалість або відсутність систем управління охороною праці, недоліки під час навчання безпечним прийомам праці, відсутність або неякісне проведення інструктажу, допуск до роботи без навчання та перевірки знань з охорони праці, неякісне розроблення, недосконалість інструкцій з охорони праці або їх відсутність, відсутність у посадових інструкціях визначення функціональних обов'язків з питань охорони праці, порушення режиму праці та відпочинку, невикористання засобів індивідуального захисту через незабезпеченість ними, виконання робіт з

відмоченими, несправними засобами колективного захисту, системи сигналізації, вентиляції, освітлення, залучення до роботи працівників не за спеціальністю, порушення технологічного процесу, порушення вимог безпеки під час експлуатації обладнання, устаткування, машин, механізмів, порушення вимог безпеки під час експлуатації транспортних засобів, порушення трудової дисципліни, невиконання посадових обов'язків; невиконання вимог інструкцій з охорони праці.

Технічні причини представленні такими чинниками, як конструктивні недоліки, недосконалість, недостатня надійність засобів виробництва, транспортних засобів, неякісне розроблення або відсутність проектної документації на будівництво, реконструкцію виробничих об'єктів, будівель, споруд, обладнання, устаткування, недосконалість технологічного процесу, його невідповідність вимогам безпеки, незадовільний технічний стан виробничих об'єктів, будинків, споруд, території, засобів виробництва, транспортних засобів; незадовільний стан виробничого середовища.

Санітарно-гігієнічні причини: незадовільні метеорологічні умови на робочому місці, перевищення запиленості та загазованості повітря робочої зони, відсутність або недостатнє природне освітлення, підвищена пульсація світлового потоку штучного освітлення; підвищений рівень шуму та вібрації, інфразвукових та ультразвукових коливань на робочому місці, підвищений рівень ультразвукової та інфрачервоної радіації.

Психофізіологічні причини: алкогольне, наркотичне сп'яніння, токсилогічне отруєння, низька нервово-психологічна стійкість, помилкові дії внаслідок втоми працівника через надмірну важкість і напруженість роботи, монотонність праці, незадовільні фізичні дані або стан здоров'я, незадовільний психологічний клімат у колективі, травмування внаслідок протиправних дій інших осіб [71].

В загальному технічні причини складають близько 20%, організаційні – 50%, санітарно-гігієнічні – 15%, психофізіологічні – 10%, інші – 5-10%.

Травматизм – це раптове ушкодження організму людини, внаслідок чого потерпілий тимчасово або назавжди втрачає працездатність

Професійне захворювання – захворювання, яке виникло внаслідок професійної діяльності застрахованого та зумовлено виключно або переважно тривалим впливом шкідливих речовин, певних видів робіт та інших факторів, пов'язаних з роботою.

Список професійних захворювань ділиться на сім основних груп. Існують захворювання, що зумовлені гострим впливом хімічних факторів. До цього пункту належать хронічні отруєння та їх наслідки, самостійні чи в поєднанні з іншими ураженнями: анемією, нефропатією, гепатитом, ураженням очей, кісток, нервової системи, органів дихання токсичного характеру. Сюди ж відносять хвороби шкіри, металеву лихоманку тощо.

Також бувають захворювання, що виникли через вплив промислових аерозолів. Це різні пневмоконіози, професійні бронхіти, бісиноз, емфізема легенів, дистрофічні зміни верхніх дихальних шляхів. Деколи стаються хвороби, що виникли в результаті впливу фізичних факторів. Очолює цей список променева хвороба і променеві ураження в гострих і хронічних стадіях, розлади вегетосудинної системи, ангіоневроз. Сюди ж належать електроофтальмія, вібраційна хвороба, нейросенсорна приглухуватість, катаракта, кесонна хвороба, перегріви, механічні епідермози, опіки і поразки лазерним випромінюванням.

Часто буваютя захворювання, що виникли в результаті фізичних перевантажень та окремих перенапружень систем і органів тіла. У цьому списку – координаторні неврози, полі- і мононевропатії, радикулопатії шийно-плечової та поперековокрижової частин, хронічні міофібрози плеча та передпліччя, тендовагініти, периартроз, варикозне розширення вен, неврози і багато інших хвороб, у тому числі деякі розлади статевої сфери [72].

Деколи є хвороби, зумовлені впливом біологічних факторів. Це – інфекційні та паразитарні хвороби, набуті в процесі професійної діяльності в результаті контакту з хворими, дисбактеріози і кандидози, обумовлені контактом із зараженими речовинами, мікози відкритих ділянок шкіри. Також алергічні захворювання: риніти, бронхіти й інші прояви алергії, що виникли в результаті необхідного контакту з речовинами та сполуками, які містять алергени.

Окрім того існують новоутворення злоякісного характеру (рак). Це пухлини печінки, шкіри, сечового міхура, лейкоз, ракові захворювання шлунка, пухлини рота та органів дихання, кісток, спричинені впливом шкідливих речовин, присутніх на робочому місці.

Порядок проведення атестації робочих місць за умовами праці від 1 серпня 1992 року. Атестація робочих місць проводиться 1 раз в 5 років

4.5 Підвищення стійкості роботи об'єктів приладобудівної галузі в воєнний час

У сучасних умовах розвитку технологій та геополітичних конфліктів особливо важливою стає проблема забезпечення стійкості функціонування об'єктів приладобудівної галузі під час воєнних дій. Це завдання вимагає комплексного підходу та розробки ефективних стратегій, спрямованих на забезпечення безперебійності виробничих процесів та збереження важливих ресурсів.

Однією з ключових складових успішної стійкості об'єктів приладобудівної галузі є впровадження сучасних систем автоматизації та контролю, які дозволяють в режимі реального часу виявляти потенційні загрози та швидко реагувати на них. Важливою є також інтеграція заходів кібербезпеки для захисту від електронних атак та витоків інформації, які можуть призвести до паралізування робочих процесів.

Підвищення стійкості об'єктів приладобудівної галузі в воєнний час також включає в себе розробку імовірних сценаріїв дій та планів евакуації для персоналу та обладнання. Надійні системи зберігання та захисту даних, а також регулярне навчання персоналу з питань безпеки і взаємодії в умовах надзвичайних ситуацій, грають ключову роль у забезпеченні ефективності заходів захисту.

Для вирішення проблем стійкості об'єктів приладобудівної галузі також важливо враховувати аспекти логістики та забезпечення, забезпечуючи

необхідні резерви матеріалів та запасів, а також створюючи механізми швидкого відновлення виробничих ліній після можливих руйнувань [73].

Додатковим аспектом, який слід враховувати при підвищенні стійкості роботи об'єктів приладобудівної галузі в умовах воєнних конфліктів, є розробка та впровадження інноваційних технологій. Використання передових науково-дослідницьких розробок у сфері матеріалознавства, енергетики та інженерії може сприяти створенню більш стійких та ефективних конструкцій об'єктів, які відповідають вимогам воєнної безпеки.

Питання енергозабезпечення також має велике значення в контексті підвищення стійкості. Розробка та використання альтернативних джерел енергії, в тому числі сонячних батарей, вітроенергетичних установок та ефективних систем зберігання енергії, може забезпечити незалежність від централізованих енергетичних мереж та збільшити стійкість об'єктів під час можливих перерв у постачанні електроенергії.

Значна увага повинна також приділятися питанням технічного обслуговування та ремонту. Розробка програм планування та управління технічним обслуговуванням, а також створення ефективних систем дистанційного моніторингу та діагностики можуть значно покращити швидкість відновлення роботи об'єктів після можливих пошкоджень чи вторгнень.

Не менш важливим є впровадження систем попередження та реагування на хімічні, біологічні та радіаційні загрози. Розробка та впровадження автоматизованих систем моніторингу середовища, а також систем для негайного реагування на можливі небезпеки, дозволяє забезпечити захист як для персоналу, так і для інфраструктури об'єктів.

Ще однією важливою аспектом є взаємодія з органами влади та військовими структурами. Розробка планів евакуації та координація дій з військовими службами можуть значно покращити реагування на можливі загрози. Також важливо створити ефективні системи зв'язку та обміну інформацією між цивільними та військовими структурами для оперативного реагування на ситуації надзвичайних подій.

Особливу увагу слід звертати на резервування та дублювання важливих систем. Створення резервних центрів у безпечних місцях та використання технологій автоматичного перемикання дозволяє підтримувати неперервну роботу об'єктів у воєнний період.

Важливою є і робота з персоналом, яка включає в себе проведення тренінгів та симуляцій в умовах воєнних дій. Підготовка персоналу до екстремальних ситуацій, а також навчання ефективним методам захисту власного життя та об'єктів промисловості є ключовим елементом стратегії підвищення стійкості.

Важливо також враховувати соціальний аспект. Створення систем підтримки для працівників та їх сімей, а також врахування психологічних аспектів в умовах воєнних конфліктів може значно полегшити адаптацію та підтримку працівників.

Додатковим важливим аспектом є розвиток мережі інфраструктури та транспорту для забезпечення надійного постачання ресурсів і матеріалів у воєнний період. Це може включати побудову альтернативних шляхів постачання, створення надійних систем транспортування, а також резервування складських приміщень для зберігання необхідних запасів.

Науково-дослідницька робота в галузі розробки нових технологій і матеріалів для об'єктів приладобудівної галузі є важливим компонентом стратегії підвищення стійкості. Використання новітніх технологій, які забезпечують більшу витривалість та надійність обладнання, може відігравати ключову роль у забезпеченні функціонування об'єктів в умовах воєнних конфліктів.

Створення партнерств та об'єднань між підприємствами галузі, державними органами, а також місцевими громадами може сприяти забезпеченню взаємодії у вирішенні загальних проблем стійкості об'єктів під час воєнних дій.

Необхідно також активно працювати над впровадженням принципів сталого розвитку та зелених технологій. Розробка екологічно чистих та

енергоефективних рішень може не лише знижувати вплив на довкілля, але і робити об'єкти більш стійкими в умовах екстремальних ситуацій.

Ще однією важливою складовою є врахування географічних та територіальних особливостей об'єктів приладобудівної галузі. Аналіз і врахування місцевих умов може допомогти визначити оптимальні місця для розташування об'єктів, а також розробляти стратегії, спрямовані на мінімізацію вразливості до воєнних загроз.

Важливою є також здатність до швидкого відновлення функціонування після можливих атак чи руйнувань. Системи резервування та швидкого відновлення інфраструктури можуть забезпечити мінімізацію перерв у виробництві та постачанні.

Окрім того, важливо враховувати аспекти відкритого доступу до інформації. Заходи з обмеження доступу до критичної інформації та використання захисту від кібератак можуть допомогти у забезпеченні конфіденційності та недоступності критичних даних для потенційних загроз.

Забезпечення сталого забезпечення об'єктів важливе не тільки в умовах воєнних дій, але і для забезпечення роботи під час кризових ситуацій та природних лих, тому розробка планів невідкладних ситуацій та управління ризиками є невід'ємною частиною стратегії стійкості об'єктів приладобудівної галузі.

Додатковим важливим аспектом є створення мережі співпраці та обміну інформацією між об'єктами приладобудівної галузі, а також між різними галузями економіки. Це дозволяє об'єктам ефективно обмінюватися досвідом, ресурсами та інформацією в умовах воєнних дій, що сприяє спільному розв'язанню проблем та підвищенню стійкості системи в цілому.

Важливим елементом є також врахування аспектів екологічної стійкості. Розвиток технологій та методів, спрямованих на мінімізацію негативного впливу виробництва на довкілля, може не лише зробити об'єкти стійкими до можливих екологічних загроз, але й забезпечити позитивний вклад у вирішення екологічних проблем.

Приділення уваги соціальним аспектам, таким як забезпечення безпеки працівників та їхніх сімей, створення програм соціальної підтримки в умовах воєнних дій, також важливо для підтримання стабільності та ефективності функціонування об'єктів.

Ще однією важливою складовою стратегії підвищення стійкості об'єктів приладобудівної галузі в умовах воєнних конфліктів є розвиток систем управління кризовими ситуаціями. Це включає в себе створення ефективних команд кризового управління, розробку планів евакуації та реагування на надзвичайні ситуації, а також проведення регулярних навчань і симуляцій.

Для підвищення рівня стійкості важливо також вдосконалювати системи моніторингу та діагностики. Використання передових технологій для постійного контролю за технічним станом обладнання дозволяє вчасно виявляти можливі поломки чи відхилення в роботі, що сприяє швидкому їх виправленню.

Активне залучення до розробки та впровадження стандартів безпеки є важливою частиною стратегії. Стандартизація процедур, технічних вимог та заходів безпеки дозволяє не лише забезпечити високий рівень стійкості об'єктів, але й полегшити взаємодію з іншими підприємствами та владними органами.

Важливим елементом є також постійне вдосконалення систем зберігання та захисту даних. У воєнний період, інформація може стати об'єктом атак, і тому розвиток криптографічних методів та систем захисту даних визначається як критичний елемент стратегії.

Ще однією важливою складовою стратегії є розробка інтегрованих планів енергозабезпечення. Створення незалежних та ефективних систем енергозабезпечення може значно підвищити стійкість об'єктів приладобудівної галузі. Використання альтернативних джерел енергії, таких як сонячні батареї, вітрогенератори та системи зберігання енергії, може зменшити залежність від традиційних джерел та забезпечити незалежність у виробництві.

Розробка та впровадження гнучких виробничих процесів також важлива для підвищення стійкості об'єктів. Можливість швидко адаптувати виробництво до змінних умов, переключатися на альтернативні види виробництва та

реагувати на виклики може допомогти у підтриманні ефективності у воєнний період.

Зберігання та раціональне використання водних ресурсів є також важливою аспектом стратегії. Врахування можливостей забезпечення водою виробничих процесів та розробка ефективних систем очищення та повторного використання води може допомогти у вирішенні питань водозабезпечення в умовах обмежених ресурсів.

Національна та міжнародна координація є ключовим елементом стратегії стійкості об'єктів приладобудівної галузі в умовах воєнних дій. Взаємодія з владними та міжнародними організаціями, обмін досвідом та ресурсами може значно зміцнити обороноздатність та резистентність об'єктів.

Додатковим аспектом в стратегії є акцент на розвиток інноваційних технологій та наукових досліджень. Застосування передових наукових розробок у сфері матеріалознавства, інженерії та інформаційних технологій може підвищити технічний рівень об'єктів і зробити їх менш вразливими до можливих атак.

Поліпшення систем безпеки та контролю над доступом – ще один ключовий аспект. Використання біометричних технологій, систем відеоспостереження та ефективних систем шифрування може забезпечити додатковий рівень захисту об'єктів у воєнний період.

Створення резервів кадрів та планів заміщення персоналу є важливою складовою стратегії [74]. В умовах воєнних дій може виникнути необхідність швидко мобілізувати та замінити працівників, тому розробка та тренування персоналу на ефективність в умовах кризи є невід'ємною частиною стратегії стійкості.

Розробка систем відновлення після кризи також важлива. Це включає в себе не тільки відновлення виробничих процесів, але і відновлення партнерських відносин, логістичних мереж та постачання.

Ще однією важливою складовою стратегії є активна участь в розвитку стандартів та нормативів з безпеки в галузі приладобудування. Створення та впровадження обов'язкових стандартів щодо конструкції, виробництва та

експлуатації може покращити загальний рівень безпеки та стійкості об'єктів, а також сприяти їхній відповідності міжнародним стандартам.

Особливу увагу слід звертати на розробку і впровадження ефективних систем виявлення та протидії кібератакам. Захист інформаційно-комунікаційної інфраструктури важливий для забезпечення безпеки та стабільності об'єктів, адже кіберзагрози можуть впливати на роботу технологічних систем, призводячи до серйозних наслідків.

Розвиток систем взаємодії та обміну інформацією з іншими об'єктами галузі та агентствами з безпеки також грає важливу роль. Активна участь у великих об'єднаннях та об'єднаннях галузі може сприяти обміну досвідом та вирішенню спільних завдань з підвищення стійкості.

Створення та вдосконалення систем внутрішнього та зовнішнього моніторингу довкілля є ще однією важливою ініціативою. Такі системи дозволяють вчасно виявляти зміни в екологічних умовах, що може бути важливим в умовах можливих загроз або кризових ситуацій.

Узагальнюючи, активна участь у розвитку стандартів безпеки, захист від кібератак, співпраця з іншими об'єктами галузі та агентствами, а також вдосконалення систем моніторингу довкілля є додатковими елементами стратегії підвищення стійкості об'єктів приладобудівної галузі [75].

4.6 Висновок до четвертого розділу

В четвертому розділі кваліфікаційної роботи описано вимоги щодо охорони праці при роботі з комп’ютерами. Наведено нормативні документи, у яких подані ці вимоги, вказано поради для роботодавців та робітників. Розглянуто питання організації охорони праці працівників на підприємстві. Описано сучасні засоби колективного та індивідуального захисту працівників. Розглянуто питання інженерного захисту персоналу об’єкту та населення. Описано захисні споруди цивільного захисту. Наведено інформацію, що стосується критичних станів людини. Особливу увагу приділено профілактиці виробничих травм. Також наведено засоби для профілактики та попередження

можливому виникненню цих травм. Описано основні сім груп можливих професійних захворювань. Розглянуто питання підвищення стійкості роботи об'єктів приладобудівної галузі в воєнний час. Описано сучасні системи автоматизації та контролю. Особливу увагу звернено на резервування та дублювання важливих систем.

ВИСНОВКИ

Для кваліфікаційної роботи на тему: “Дослідження застосунків блокчейну та смарт-контрактів засобами мови програмування Rust для оптимізації різногалузевих задач” була використана мова програмування Rust у програмному середовищі VScode на операційній системі Linux.

В першому розділі кваліфікаційної роботи:

– Подано інформацію про блокчейн, а також вказано його переваги на недоліки.

– Розглянуто поняття ноди та метод досягнення консенсусу для роботи смарт-контрактів.

– Описано інноваційні технології для виборів та ліквідні пули.

– Проаналізовано смарт-контракти та їхні переваги і недоліки.

– Подано та проаналізовано інформацію про існуючі рішення проблеми.

В другому розділі кваліфікаційної роботи:

– Подано інформацію про блокчейн Solana, а також вказано його переваги на недоліки.

– Розглянуто мову програмування Rust, її недоліки та переваги, а також обґрунтовано її вибір.

– Проаналізоване програмне середовище Visual Studio Code.

– Розглянуто операційну систему Linux та її переваги і недоліки.

В третьому розділі кваліфікаційної роботи:

– Описано створення смарт-контракту для liquidity pool.

– Спроектовано смарт-контракт для системи голосування на виборах на блокчейні Solana за допомогою мови програмування Rust.

– Спроектовано основну логіку та логіку обробки помилок для розроблювальних смарт-контрактів.

– Написано інтеграційні тести для усього функціоналу.

– Наведено опис основних інструкцій для кожного із контрактів.

– Розглянуто структури даних, необхідні для реалізації смарт-контрактів.

У розділі «Безпека життєдіяльності, основи хорони праці» розглянуто вимоги щодо охорони праці при роботі з комп’ютером, описано організацію охорони праці працівника на підприємстві, проаналізовано критичні стани людини, а також інженерний захист персоналу об’єкту та населення, розглянуто підвищення стійкості роботи об'єктів приладобудівної галузі в воєнний час.

ПЕРЕЛІК ДЖЕРЕЛ

1 Raj, Koshik. Foundations of blockchain: the pathway to cryptocurrencies and decentralized blockchain applications. Packt Publishing Ltd, 2019.

2 Song, Jimmy. Programming bitcoin: Learn how to program bitcoin from scratch. O'Reilly Media, 2019.

3 Klabnik, Steve, and Carol Nichols. The Rust programming language. No Starch Press, 2023.

4 McNamara, Tim. Rust in Action. Simon and Schuster, 2021.

5 Klabnik, Steve, and Carol Nichols. The Rust programming language. No Starch Press, 2023.

6 Wu, Xun Brian, Zhihong Zou, and Dongying Song. Learn ethereum: build your own decentralized applications with ethereum and smart contracts. Packt Publishing Ltd, 2019.

7 Comert, Oguz. "Blockchain Revolution: How the Technology behind Bitcoin and Other Cryptocurrencies Is Changing the World." (2020): 272.

8 Zheng, Zibin, et al. "An overview on smart contracts: Challenges, advances and platforms." Future Generation Computer Systems 105 (2020): 475-491.

9 Bugden, William, and Ayman Alahmar. "Rust: The programming language for safety and performance." arXiv preprint arXiv:2206.05503 (2022).

10 Tapscott, Don, and Alex Kaplan. "Blockchain revolution in education and lifelong learning." Blockchain Research Institute-IBM Institute for Business Value (2019).

11 Herian, Robert. "Smart contracts: a remedial analysis." Information & Communications Technology Law 30.1 (2021): 17-34.

12 Burger, Annetta, William G. Kennedy, and Andrew Crooks. "Organizing theories for disasters into a complex adaptive system framework." Urban Science 5.3 (2021): 61.

13 Labouseur, Alan G., Matthew Johnson, and Thomas Magnusson. "Demystifying blockchain by teaching it in computer science: adventures in essence,

accidents, and data structures." Journal of Computing Sciences in Colleges 34.6 (2019): 43-56.

14 Fandl, Kevin J. "Can smart contracts enhance firm efficiency in emerging markets?." Nw. J. Int'l L. & Bus. 40 (2019): 333.

15 Jung, Ralf. "Understanding and evolving the Rust programming language." (2020).

16 Tapscott, Don, and Alex Tapscott. Blockchain revolution: how the technology behind bitcoin is changing money, business, and the world. Penguin, 2019.

17 Bomprezzi, Chantal. Implications of blockchain-based smart contracts on contract law. Vol. 23. Nomos Verlag, 2021.

18 Cogo, Filipe Roseiro, Xin Xia, and Ahmed E. Hassan. "Assessing the alignment between the information needs of developers and the documentation of programming languages: a case study on Rust." ACM Transactions on Software Engineering and Methodology 32.2 (2023): 1-48.

19 Antonopoulos, Andreas M., Olaoluwa Osuntokun, and René Pickhardt. Mastering the Lightning Network. " O'Reilly Media, Inc.", 2021.

20 Zhu, Shuofei, et al. "Learning and programming challenges of rust: A mixed-methods study." Proceedings of the 44th International Conference on Software Engineering. 2022.

21 Moringiello, Juliet M., and Christopher K. Odinet. "The property law of tokens." Fla. L. Rev. 74 (2022): 607.

22 Jean-Louis, Nerla, et al. "Sgxonerated: Finding (and partially fixing) privacy flaws in tee-based smart contract platforms without breaking the tee." Cryptology ePrint Archive (2023).

23 Klabnik, Steve, and Carol Nichols. The Rust programming language. No Starch Press, 2023.

24 Singh, Manpreet. Is blockchain a paradigmatic shift in accounting technology?. Diss. RMIT University, 2023.

25 Singh, Manpreet. Is blockchain a paradigmatic shift in accounting technology?. Diss. RMIT University, 2023.

26 Klabnik, Steve, and Carol Nichols. The Rust programming language. No Starch Press, 2023.

27 Chisnall, David, et al. "Secure Compilation (Dagstuhl Seminar 21481)." Dagstuhl Reports. Vol. 11. No. 10. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

28 Strutynska, I., Dmytrotsa, L., Kozbur, H., Ermolayev, V., Mallet, F., Yakovyna, V., ... & Spivakovsky, A. (2019, June). The Main Barriers and Drivers of the Digital Transformation of Ukraine Business Structures. In ICTERI (pp. 50-64).

29 Marinova, Svetla T., and Marin A. Marinov. "RECONFIGURATION OF BUSINESS MODELS AND ECOSYSTEMS."

30 Craib, Raymond. Adventure capitalism: A history of libertarian exit, from the era of decolonization to the digital age. PM Press, 2022.

31 Wright, Robert E., and Aleksandra Przegalińska. Debating Universal Basic Income: Pros, Cons, and Alternatives. Springer Nature, 2022.

32 Gopu, Arunkumar, Neelanarayanan Venkataraman, and M. Nalini. "Toward the Internet of Things and Its Applications: A Review on Recent Innovations and Challenges." Cognitive Computing for Internet of Medical Things (2022): 1-21.

33 CHIAVAZZO, ELIODORO, et al. "Critical overview and feasibility study of two-phase immersion cooling systems for power electronics." (2022).

34 Acemoglu, Daron, Capital Deepening, and Non-Balanced Economic Growth. "VeronicaGuerrieri." Capital Deepening andNonbalanced Economic Growth Journal ofPolitical Economy 116 (2018).

35 Evans, Ana Nora, Bradford Campbell, and Mary Lou Soffa. "Is Rust used safely by software developers?." Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. 2020.

36 Borgsmüller, Nico. The Rust programming language for embedded software development. Diss. Technische Hochschule Ingolstadt, 2021.

37 Bae, Yechan, et al. "Rudra: finding memory safety bugs in rust at the ecosystem scale." Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. 2021.

38 Sharma, Rahul, Vesa Kaihlavirta, and Claus Matzinger. The The Complete Rust Programming Reference Guide: Design, develop, and deploy effective software systems using the advanced constructs of Rust. Packt Publishing Ltd, 2019.

39 Qin, Boqin, et al. "Understanding memory and thread safety practices and issues in real-world Rust programs." Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. 2020.

40 Rivera, Elijah, et al. "Keeping safe rust safe with galeed." Annual Computer Security Applications Conference. 2021.

41 Bugden, William, and Ayman Alahmar. "The Safety and Performance of Prominent Programming Languages." International Journal of Software Engineering and Knowledge Engineering 32.05 (2022): 713-744.

42 Ardito, Luca, et al. "rust-code-analysis: A Rust library to analyze and extract maintainability information from source codes." SoftwareX 12 (2020): 100635.

43 Tillemann, Tomicah, et al. "The blueprint for blockchain and social innovation." New America (2019).

44 Tapscott, Don, and Alex Tapscott. Blockchain revolution: how the technology behind bitcoin is changing money, business, and the world. Penguin, 2019.

45 Korneychuk, Boris. "The Political Economy of the Blockchain Society." Digital Transformation and Global Society: Third International Conference, DTGS 2018, St. Petersburg, Russia, May 30–June 2, 2018, Revised Selected Papers, Part I 3. Springer International Publishing, 2018.

46 Novotný, Tomáš. Překážky uplatnění blockchainu v obchodním modelu založeném na sdílené ekonomice. Diss. Masaryk University, Faculty of Economics and Administration, 2022.

47 Herian, Robert. "Taking blockchain seriously." Law and Critique 29 (2018): 163-171.

48 O'Dair, Marcus, and Marcus O’Dair. "Blockchain: the internet of value." Distributed Creativity: How Blockchain Technology will Transform the Creative Economy (2019): 15-30.

49 Hart, Cara Confer. Exploring the Experiences of Primary Care Staff Due to the Lack of Interoperability between Electronic Health Records. Diss. Colorado Technical University, 2022.

50 Dwamena, Elizabeth. Smart contracts: the Fourth Industrial Revolution and private international law. University of Johannesburg (South Africa), 2021.

51 Ghodoosi, Farshad. "Contracting in the age of smart contracts." Wash. L. Rev. 96 (2021): 51.

52 Bayern, Shawn. Autonomous organizations. Cambridge University Press, 2021.

53 Fandl, Kevin J. "Can smart contracts enhance firm efficiency in emerging markets?." Nw. J. Int'l L. & Bus. 40 (2019): 333.

54 Reyes, Carla L. "If Rockefeller were a coder." Geo. Wash. L. Rev. 87 (2019): 373.

55 Alonzo, Carolina. "Decentralized autonomous organization: is it the corporate future?." (2022).

56 Rust, Chelsea Lynn. "Implementation of a Clinical Practice Guideline for the Prevention and Management of Adult Obesity." (2019).

57 Chatley, Robert, Alastair Donaldson, and Alan Mycroft. "The next 7000 programming languages." Computing and software science: State of the art and perspectives (2019): 250-282.

58 Kennedy, Fred George William. Impact of nuclear thermal propulsion on the NASA 90-day study's baseline missions for the space exploration initiative. Diss. Massachusetts Institute of Technology, 2020.

59 Angst, Ueli M. "Challenges and opportunities in corrosion of steel in concrete." Materials and Structures 51 (2018): 1-20.

60 Crichton, Will, et al. "Modular information flow through ownership." Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation. 2022.

61 McColl-Kennedy, Janet R., et al. "Gaining customer experience insights that matter." Journal of service research 22.1 (2019): 8-26.

62 VanHattum, Alexa. Lightweight Formal Methods for Correct, Efficient Systems Programming. Diss. Cornell University, 2023.

63 Obot, I. B., et al. "Progress in the development of sour corrosion inhibitors: Past, present, and future perspectives." Journal of Industrial and Engineering Chemistry 79 (2019): 1-18.

64 Obot, I. B., et al. "Progress in the development of sour corrosion inhibitors: Past, present, and future perspectives." Journal of Industrial and Engineering Chemistry 79 (2019): 1-18.

65 Howell, Bronwyn E., and Petrus H. Potgieter. "Governance of smart contracts in blockchain institutions." Available at SSRN 3423190 (2019).

66 Міщенко, С. М. "Управління персоналом підприємства." (2023).

67 Міщенко, Юлія Павлівна. "Аналіз і оцінка підприємницької діяльності." (2023).

68 Слухай, Олександра Олександрівна. "Діагностика фінансового стану підприємства та шляхи його покращення." (2020).

69 Петрованов, Андрій Сергійович. "Вдосконалення управління грошовими розрахунками сільськогосподарського підприємства." (2021).

70 Кравченко, Олександр Олександрович. "Формування системи стратегічного управління кадровим складом підприємства." (2023).

71 Захарчук, Світлана Павлівна. "Інноваційні напрями стратегічного розвитку підприємства (на прикладі ТОВ «ОСП Корпорація Ватра»)." (2019).

72 Руднєва, Олена Юріївна. "РЕГУЛЮВАННЯ ЕКОЛОГІЧНИХ НАСЛІДКІВ ДІЯЛЬНОСТІ ПРОМИСЛОВОГО ПІДПРИЄМСТВА." (2020).

73 Міщенко, Валерія Олександрівна. "Інноваційне забезпечення надання послуг на ТОВ «Нова пошта»." (2023).

74 Орлатий, Михайло Кузьмович. "25.00. 02–механізми державного управління Дисертація на здобуття наукового ступеня."

75 Аулін, В. В., et al. "Теоретичні і методологічні основи логістики транспортних і виробничих систем." (2021).

ДОДАТКИ

Додаток А

Публікація 1

Матеріали XII Міжнародної науково-практичної конференції молодих учених та студентів «АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ» – Тернопіль, 6-7 грудня 2023 року

УДК 004.43
С.-З. Ю. Хома, А. В. Семак, к.т.н. Г. В. Козбур
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

LIQUIDITY POOL ЯК ЗАМІНА ЗВИЧНИХ РИНКІВ ДЛЯ ТОРГІВЛІ ВАЛЮТАМИ

S.-Z. Y. Khoma, A. V. Semak, H. V. Kozbur, Ph.D., Assoc. Prof.

LIQUIDITY POOL AS A SUBSTITUTE FOR TRADITIONAL MARKETS FOR CURRENCY TRADING

Традиційні ринки валют стикаються з проблемами, такими як нестабільність ліквідності, високі комісії за транзакції та потреба в посередниках, що ускладнює торгівлю та знижує ефективність. Ліквідні пули (liquidity pool), як альтернатива, можуть пропонувати рішення цих проблем за рахунок інноваційних технологій.

Ліквідні пули – це децентралізовані платформи, що дозволяють криптовалютну торгівлю без необхідності традиційного посередника. Основною перевагою ліквідних пулів є їхня спроможність забезпечувати постійну ліквідність, використовуючи алгоритмічне ціноутворення, що значно знижує розрив між купівлею та продажем та забезпечує більш стабільні ціни. Ліквідні пули використовують спеціальні алгоритми для визначення ціни активів, що дозволяє уникнути проблем, пов'язаних з недостатньою ліквідністю на традиційних ринках.

Ще однією перевагою є зниження транзакційних витрат. Оскільки ліквідні пули виключають необхідність у брокерських комісіях та інших посередницьких витратах, вони дозволяють здійснювати торгівлю за нижчими цінами, що робить ринок доступнішим для ширшого кола інвесторів.

Водночас, ліквідні пули використовують механізм автоматичного маркет-мейкінгу, що дозволяє здійснювати обміни без прямої участі інших трейдерів, що значно спрощує процес торгівлі.

Також, ліквідні пули можуть стати інструментом для залучення нових інвесторів, які цікавляться децентралізованими фінансами (DeFi) та шукають альтернативні способи інвестування та зберігання коштів. Децентралізована фінансова екосистема значно знижує бар'єри для входу нових учасників. Ця модель не тільки сприяє підвищенню ліквідності на ринках, але й робить процес більш інклюзивним, дозволяючи будь-якому індивіду з мінімальними інвестиціями стати ліквідним постачальником.

Децентралізація, яка є ключовим принципом ліквідних пулів, також відіграє важливу роль у зменшенні системного ризику. В традиційних фінансових системах, централізовані інститути, такі як банки та біржі, можуть становити точки відмови. Втім, у системах, побудованих на ліквідних пулах, ризик розподіляється серед всієї мережі, що робить кожного учасника не тільки інвестором, але й частиною стійкої інфраструктури.

Технологія блокчейн, яка лежить в основі ліквідних пулів, забезпечує високий ступінь безпеки завдяки своїй незмінності та прозорості. Кожна транзакція є публічно записаною і перевіреною, що знижує шанси на маніпуляції на ринку або шахрайські дії.

Крім того, ліквідні пули можуть сприяти глобалізації торгівлі, оскільки вони не мають географічних обмежень і дозволяють трейдерам з будь-якої точки світу взаємодіяти з пулом без обмежень. Це створює додаткові можливості для арбітражу та інших торговельних стратегій, забезпечуючи більшу ефективність ринку.

Розвиток ліквідних пулів також відкриває двері для створення спеціалізованих фінансових інструментів, які можуть пропонувати хеджування, левередж, та інші

складні фінансові операції без необхідності звертання до традиційних посередників. Це не тільки спрощує процеси, але й знижує витрати для кінцевих користувачів, дозволяючи їм використовувати більш складні фінансові стратегії.

Серед успішних прикладів застосування ліквідіті пулів у реальному світі можна навести Uniswap і Aave. Обидві платформи призначені для децентралізованого обміну криптовалют, де використовуються ліквідні пули для забезпечення торгівлі. Користувачі можуть вносити свої активи в пули ліквідності і отримувати винагороду у вигляді транзакційних зборів, зароблених пулом.

Концепція ліквідності як послуги (Liquidity as a Service - LaaS) стає все більш актуальною у світі DeFi, де ліквідні пули можуть надавати свої ресурси іншим децентралізованим платформам або традиційним фінансовим інституціям. Ця модель дозволяє платформам забезпечувати неперервну торгівлю, мінімізувати вартість залучення ліквідності та пропонувати користувачам кращі умови обміну. Такий симбіоз може створити новітній міст між DeFi та традиційними фінансами, підвищуючи ефективність та інклюзивність фінансових систем на глобальному рівні.

Однак, слід визнати певні ризики, які супроводжують ліквідні пули. Імперманентні втрати – це один із ризиків для постачальників ліквідності, що виникають, коли тимчасові різниці в цінах можуть призвести до втрат у порівнянні зі зберіганням активів поза пулом. Попри це, інноваційні механізми компенсації, такі як виплата відсотків або розподіл транзакційних зборів, можуть пом'якшити ці втрати.

Незважаючи на зростаючу популярність та очевидні переваги, ліквідні пули все ще знаходяться на ранніх етапах розвитку, і регуляторна невизначеність залишається проблемою. Законодавче регулювання може вплинути на розвиток та прийняття ліквідних пулів, особливо в регіонах з жорсткими фінансовими вимогами. Водночас, зростання інтересу до DeFi та ліквідних пулів спонукає регуляторів розробляти нові рамки, що можуть сприяти інноваціям та захисту інвесторів.

Враховуючи успішні приклади використання ліквідних пулів на криптовалютних платформах, можна стверджувати, що вони мають потенціал стати важливим елементом у сфері торгівлі валютами. Їх інноваційний підхід може забезпечити більш ефективну, безпечну та доступну торгівлю для всіх учасників ринку.

Висновок: Ліквідні пули являють собою перспективну заміну традиційним ринкам для торгівлі валютами, пропонуючи вдосконалені технологічні рішення, які підвищують ліквідність, знижують витрати та сприяють демократизації фінансових ринків.

Література

1. Zhang, L., Ma, X., & Liu, Y. (2022). SoK: Blockchain Decentralization. arXiv preprint arXiv:2205.04256.
2. Andreas, M. A. (2014). Mastering Bitcoin: unlocking digital cryptocurrencies.
3. Lewis, A. (2018). The basics of bitcoins and blockchains: an introduction to cryptocurrencies and the technology that powers them. Mango Media Inc.

Публікація 2

Матеріали XII Міжнародної науково-практичної конференції молодих учених та студентів «АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ» – Тернопіль, 6-7 грудня 2023 року

УДК 004.43
А. В. Семак, С.-З. Ю. Хома, к.т.н. Г. В. Козбур
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ВИКОРИСТАННЯ СМАРТ-КОНТРАКТІВ ДЛЯ ОПТИМІЗАЦІЇ ПРОЦЕСУ ГОЛОСУВАННЯ НА ВИБОРАХ

A. V. Semak, S.-Z. Y. Khoma, H. V. Kozbur, Ph.D., Assoc. Prof.

USING SMART CONTRACTS TO OPTIMIZE THE VOTING PROCESS IN ELECTIONS

Сучасні виборчі системи стикаються з численними викликами, такими як недостатня прозорість, можливість шахрайства та труднощі в організації та підрахунку голосів. Застосування смарт-контрактів видається перспективним напрямком для вирішення цих проблем. Смарт-контракти – це програмні коди, які автоматизовано виконують, контролюють та верифікують угоди. Основні характеристики смарт-контрактів наведено в таблиці 1 [1].

Таблиця 1. Основні характеристики смарт-контрактів

Характеристики смарт-контрактів	– неможливість внесення змін після ініціалізації – захищеність від несанкціонованого доступу – можливість самоперевірки – недоцільність залучення посередників
Можливості смарт-контрактів	– автоматизація типових процесів – підтримка облікових записів з мультипідписом для розподілу управління – зменшення впливу довіри при виборі контрагента
Застосування смарт-контрактів	– автоматизація типових бізнес-процесів – фінансові операції – демократичне децентралізоване управління – управління ланцюгами поставок – реєстрація нерухомості, авторських прав
Недоліки смарт-контрактів	– складність кодування контрактів зі спірними умовами – відкритість інформації, яка зберігається на блокчейн – складність використання для великих обсягів даних

Смарт контракти допомагають позбутись багатьох соціальних проблем при голосуванні. Ціна голосу на виборах може коливатися в залежності від численних факторів. Економічний статус, рівень корупції, соціальний тиск і політична система — усе це може впливати на те, яка вартість приділяється голосу. У країнах з високим рівнем бідності та нерівності може виникати схильність продавати голоси чи піддаватися впливу заради особистих вигод. Корупція може робити голоси доступними для купівлі, знижуючи їхню вартість. Соціальний тиск або вплив від родини, спільноти чи роботодавця також може впливати на свободу висловлення виборців. У країнах з високим рівнем демократії та політичною стабільністю голос може бути більш цінним, в той час як у менш розвинених політично системах його вартість може бути піддаватися сумнівам. Освіта також відіграє важливу роль: рівень освіченості може визначати, наскільки ефективно виборці можуть виражати свою волю на основі інформації та аналізу політичних питань.

Поряд із цим, важливо відзначити, що смарт-контракти можуть розширити можливості голосування, зокрема для громадян, які перебувають за межами свого

резидентського району або навіть країни. Впровадження смарт-контрактів може стати не лише засобом оптимізації процесів, а й демократизації голосування, роблячи його більш доступним та універсальним для всіх шарів населення.

До прикладів застосування блокчейн-голосування у реальному світі можна віднести вибори 2021 року в Гренландії. Країна, населення якої становить 56 000, використовувала блокчейн у своїх виборах 2021 року. Цей випадок продемонстрував потенціал для використання блокчейну у виборах у менших електоральних системах, а також висвітлив сфери, що потребують поліпшення. Також можна зазначити використання ZCoin Тайською Демократичною Партією. У листопаді 2018 року вона провела праймеріз для вибору свого нового лідера, використовуючи цифровий токен ZCoin. Це були перші великомасштабні політичні вибори, проведені з використанням технології блокчейну, і вони успішно зібрали 127 479 голосів з усього Таїланду.

Оскільки блокчейн-технології можуть мати обмеження щодо швидкості та обсягу транзакцій, оптимізація смарт-контрактів є критичною для їх ефективного використання в масштабних виборчих процесах. Розробники повинні враховувати витрати на комісію транзакцій і час виконання транзакцій, оптимізуючи код для мінімізації ресурсів. Крім того, важливим є використання архітектурних підходів, які дозволяють масштабувати систему без втрати продуктивності, наприклад, шардування даних або використання легких блокчейн-протоколів.

Висновок. Розглянуто потенціал використання смарт-контрактів як інструменту для оптимізації виборчих систем. Проаналізовано основні переваги застосування блокчейн-технологій, зокрема підвищення рівня безпеки, прозорості та автоматизації виборчих процесів. Зазначено, що смарт-контракти сприяють зменшенню ризиків шахрайства та підвищують довіру громадян до чесності та легітимності виборів. Впровадження цієї технології може сприяти демократизації голосування та зробити його більш доступним для глобального електорату. Наведено успішні приклади впровадження смарт-контрактів у реальних виборчих кампаніях, що демонструють їх ефективність та перспективність. Враховуючи отримані дані, можна зробити висновок, що смарт-контракти мають значний потенціал для реформування сучасних виборчих систем та забезпечення сталого розвитку демократичних процесів.

Література

1. Кулинич Віталій. Штучний інтелект для смарт-контрактів. https://yur-gazeta.com/publications/practice/informaciyne-pravo-telekomunikaciyi/shtuchniy-intelekt-dlya-smartkontraktiv.html

2. ANITHA, V., et al. Transparent voting system using blockchain. Measurement: Sensors, 2023, 25: 100620.

3. PAWLAK, Michał; PONISZEWSKA-MARAŃDA, Aneta. Trends in blockchain-based electronic voting systems. Information Processing & Management, 2021, 58.4: 102595.

4. MACRINICI, Daniel; CARTOFEANU, Cristian; GAO, Shang. Smart contract applications within blockchain technology: A systematic mapping study. Telematics and Informatics, 2018, 35.8: 2337-2354.

Публікація 3

УДК 004.43

А. Семак, С.-З. Хома, Г. Козбур, к.т.н., доц.
(Тернопільський національний технічний університет імені Івана Пулюя)

ІНФОРМАЦІЙНІ СИСТЕМИ СМАРТ-КОНТРАКТІВ ДЛЯ ВИБОРЧИХ ПРОЦЕСІВ

A. Semak, S.-Z. Khoma, H. Kozbur, Ph.D., Assoc. Prof.

INFORMATION SYSTEMS OF SMART CONTRACTS FOR ELECTION PROCESSES

Традиційні виборчі системи стикаються з численними викликами, такими як недостатня прозорість, можливість шахрайства та труднощі в організації та підрахунку голосів. Застосування смарт-контрактів, які є угодами між вузлами мережі блокчейн, видається перспективним напрямком для вирішення цих проблем. Смарт-контракти – це комп'ютерні протоколи, який використовуються для цифрового спрощення, перевірки або забезпечення виконання переговорів щодо контракту. Основні характеристики смарт-контрактів наведено в таблиці 1.

Таблиця 1. Основні характеристики смарт-контрактів

Характеристики смарт-контрактів	Можливості смарт-контрактів	Застосування смарт-контрактів	Недоліки смарт-контрактів
• неможливість внесення змін після ініціалізації • захищеність від несанкціонованого доступу • можливість самоперевірки • недоцільність залучення посередників	• автоматизація типових процесів • підтримка облікових записів з мультипідписом для розподілу управління • зменшення впливу довіри при виборі контрагента	• автоматизація типових бізнес-процесів • фінансові операції • демократичне децентралізоване управління • управління ланцюгами поставок • реєстрація нерухомості, авторських прав	• складність кодування контрактів зі спірними умовами • відкритість інформації, яка зберігається на блокчейн • складність використання для великих обсягів даних

Впровадження смарт-контрактів є засобом оптимізації й демократизації голосування, роблячи його більш доступним та універсальним для всіх шарів населення. Смарт контракти завдяки своїм можливостям допомагають позбутись багатьох соціальних проблем при голосуванні – підкуп виборця, соціальний тиск, корупція, тощо. Смарт-контракти можуть розширити можливості голосування, зокрема для громадян, які перебувають за межами свого резидентського району або навіть країни.

До прикладів успішного застосування блокчейн-голосування у реальному світі можна віднести вибори 2021р. в Гренландії. Цей випадок продемонстрував потенціал для використання блокчейну у виборах електоральних системах, а також висвітлив сфери, що потребують поліпшення. У листопаді 2018 року Тайська Демократична Партія провела праймеріз для вибору свого нового лідера, використовуючи цифровий токен ZCoin. Це були перші великомасштабні політичні вибори, проведені з використанням технології блокчейну, і вони успішно зібрали майже 130 тисяч голосів з усього Таїланду.

Оскільки блокчейн-технології можуть мати обмеження щодо швидкості та обсягу транзакцій, оптимізація смарт-контрактів є критичною для їх ефективного використання в масштабних виборчих процесах. Розробники повинні враховувати витрати на комісію транзакцій і час виконання транзакцій, оптимізуючи код для мінімізації ресурсів. Крім того, важливим є використання архітектурних підходів, які дозволяють масштабувати систему без втрати продуктивності, наприклад, шардування даних або використання легких блокчейн-протоколів.

Публікація 4

VI Міжнародна студентська науково - технічна конференція
"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

УДК 004.43

Хома С.-З. – ст. гр. САм-51
Тернопільський національний технічний університет імені Івана Пулюя

СМАРТ-КОНТРАКТИ НА ОСНОВІ RUST ДЛЯ РОЗВИТКУ ЦИФРОВОЇ ЕКОНОМІКИ

Науковий керівник: старший викладач Шимчук Г.

Khoma S.-Z.
Ternopil Ivan Puluj National Technical University

SMART-CONTRACTS BASED ON RUST FOR DIGITAL ECONOMY DEVELOPMENT

Supervisor: Senior Lecturer Shymchuk G.

Ключові слова: rust, смарт-контракт, економіка, блокчейн.
Key words: rust, smart-contract, economy, blockchain.

Смарт-контракти це програмні коди, які дозволяють автоматизувати виконання угод та забезпечити їх безпеку за допомогою технології блокчейн. Вони можуть використовуватись у різних галузях, таких як фінанси, медицина, логістика та інші, де необхідно забезпечити точність виконання угод та уникнути ризику шахрайства або помилок.

Мова програмування Rust зазвичай використовується для розробки смарт-контрактів через свої властивості безпеки, надійності та швидкодії. Ця мова програмування дозволяє розробникам створювати надійні програми, які важко підробити або зламати. Крім того, Rust може працювати на різних архітектурах, що робить його універсальним інструментом для розробки смарт-контрактів[1].

Застосування смарт-контрактів на основі мови програмування Rust може мати значний вплив на розвиток суспільства і поліпшення якості життя людей. Зокрема, вони можуть допомогти зменшити бюрократію та спростити процеси укладання угод, забезпечити точність виконання угод, знизити витрати та покращити ефективність бізнесу. Крім того, вони можуть допомогти у забезпеченні доступу до фінансових послуг та захисту прав громадян, зокрема у контексті мікрокредитування та соціальних програм. Смарт-контракти також можуть забезпечити відкритість та прозорість у відносинах між сторонами, що сприятиме побудові довіри в бізнес-та соціальних відносинах[1].

Однак, використання смарт-контрактів також потребує уваги до кількох проблем, зокрема, забезпечення безпеки коду, вирішення проблеми зміни правил угоди після її укладення та забезпечення достатнього рівня технічної готовності користувачів для використання нових технологій[2].

У майбутньому, застосування смарт-контрактів на основі мови програмування Rust може мати значний вплив на розвиток цифрової економіки та допомогти вирішити багато сучасних проблем суспільства[3]. Однією з ключових переваг використання мови програмування Rust для розробки смарт-контрактів її високий рівень безпеки та надійності. Rust володіє системою відслідковування пам'яті та механізмами контролю доступу, що дозволяють попереджувати багато типів помилок, що можуть призвести до критичних безпекових проблем[2]. Це особливо важливо в контексті смарт-контрактів, які часто містять фінансову та особисту інформацію користувачів.

Крім того, Rust володіє високим рівнем продуктивності та швидкості виконання коду, що важливим для розробки складних смарт-контрактів з великою кількістю операцій та обчислень[1].

Застосування смарт-контрактів на основі мови програмування Rust може мати значний вплив на розвиток різних сфер, включаючи фінанси, логістику, охорону здоров'я та багато інших. Вони можуть допомогти побудувати більш прозорі та безпечні відносини між сторонами та знизити витрати на транзакції та посередництво. Відкриваються нові можливості для реалізації соціальних програм та проектів, зокрема у сфері екології та боротьби зі зміною клімату[2].

Хоча смарт-контракти можуть забезпечувати безпеку та ефективність у розробці нових бізнес-моделей, їх вплив на світову економіку може бути негативним. Наприклад, необхідність застосування смарт-контрактів може призвести до зменшення кількості робочих місць у багатьох галузях, а також до появи нових форм економічної нерівності. Крім того, наявність великої кількості смарт-контрактів на блокчейні може спричинити проблеми з масштабованістю та швидкодією системи, що може вплинути на ефективність її роботи. Тому, перед тим як впроваджувати смарт-контракти в широкому масштабі, потрібно ретельно оцінювати їхній вплив на різні аспекти суспільства та господарства.

Також варто зазначити, що смарт-контракти можуть стати об'єктом кібератак, що може призвести до крадіжки цифрових активів та порушення довіри до системи в цілому. Крім того, несправедлива дистрибуція цифрових активів за допомогою смарт-контрактів може призвести до зростання економічної нерівності, оскільки доступ до деяких ресурсів може бути обмеженим[3]. Таким чином, для успішного впровадження смарт-контрактів в суспільство потрібно вирішувати важливі економічні, соціальні та правові проблеми та забезпечувати відповідну регуляторну базу для розвитку цієї технології.

Проте, повний перехід економіки на використання смарт-контрактів, без використання старих технологій є доволі важким. Однією з причин є те, що на даний момент є деякі обмеження в їхній функціональності порівняно з традиційними контрактами. Наприклад, смарт-контракти можуть бути обмежені в тому, які умови можуть бути включені в контракт та які події можуть спричинити автоматичне виконання контракту. Також, існує ще одна причина, чому повний перехід до смарт-контрактів може бути складним - це велика кількість технологій та стандартів, які вже застосовуються у різних економічних системах та які використовують традиційні контракти. Це може призвести до того, що розробка інфраструктури та забезпечення сумісності між різними системами може бути витратною та складною задачею.

Отже, використання смарт-контрактів на основі мови програмування Rust може стати ключовим фактором у розвитку цифрової економіки та досягненні більш стійкого та просунутого суспільства в майбутньому.

Література:

1. Andreas M. Antonopoulos. The Internet of Money: A collection of talks / M. Antonopoulos Andreas / CreateSpace Independent Publishing Platform; 1st edition. 2016, 152 p. ISBN 1537000454

2. Melanie Swan. Blockchain: Blueprint for a New Economy. / Swan Melanie. – O'Reilly, 2015, 130 p. ISBN 9781491920497

3. Alex Tapscott, Don,Tapscott. Blockchain Revolution: How the Technology Behind Bitcoin and Other Cryptocurrencies is Changing the World / Tapscott Alex, ,Tapscott Don / Portfolio Penguin, 2016, 384 p. ISBN 0241237858

Додаток Б

Вміст файлу entrypoint.rs для смарт-контакту, який реалізовує систему голосування

```
use solana_program::{
    account_info::AccountInfo, entrypoint,
entrypoint::ProgramResult, pubkey::Pubkey,
};

use crate::processor::Processor;

entrypoint!(process_instruction);

pub fn process_instruction(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    instruction_data: &[u8],
) -> ProgramResult {
    Processor::process(program_id, accounts,
instruction_data)
}
```

Вміст файлу error.rs для смарт-контакту, який реалізовує систему голосування

```
use solana_program::program_error::ProgramError;
use thiserror::Error;

#[derive(Error, Debug, Copy, Clone)]
pub enum VoteError {
    #[error("User signature is required")]
    SignedRequired,

    #[error("Admin signature is required")]
    AdminRequired,

    #[error("Trying to create second vote counted")]
    DoubleCounter,

    #[error("Trying to create new vote when max count of
votes already created")]
    MaxVote,

    #[error("Trying to define non-existed vote")]
    WrongVoteDefine,

    #[error("Trying to double participate in single
vote")]
    DoubleParticipate,
```

```
    #[error("Trying to participate in closed vote")]
    CloseVoteParticipate,

    #[error("Wrong UserVote PDA")]
    WrongUserVotePDA,

    #[error("Wrong settings PDA")]
    WrongSettingsPDA,
}

impl From<VoteError> for ProgramError {
    fn from(e: VoteError) -> Self {
        ProgramError::Custom(e as u32)
    }
}
```

Вміст файлу instruction.rs для смарт-контакту, який реалізовує систему голосування

```
use borsh::{BorshDeserialize, BorshSerialize};
use solana_program::{
    instruction::{AccountMeta, Instruction},
    pubkey::Pubkey,
    system_program, sysvar,
};

use crate::{
    id,
    state::{UserVotes, Vote, VoteCounter},
};

#[derive(BorshSerialize, BorshDeserialize, Debug, Clone,
PartialEq)]
pub enum VoteInstruction {
    /// Participate in vote.
    /// Accounts:
    /// 0. `[signer]` want to vote
    /// 1. `[writable]` contain info about vote that this
user participate in, PDA
    /// 2. `[]` concrete vote, PDA
    /// 3. `[]` Rent sysvar
    /// 4. `[]` System program
    Vote { direction: Direction },

    /// Create a vote.
    /// Accounts:
    /// 0. `[signer]` admin
    /// 1. '[writable]' vote to create, PDA
    /// 2. '[writable]' vote counter, PDA
    /// 3. `[]` Rent sysvar, PDA
    /// 4. `[]` System program, PDA
```

```
        /// 5. '[]' Clock, PDA
        CreateVote { vote_seed: Pubkey },

        /// Delete a vote.
        /// Accounts:
        /// 0. `[signer]` admin
        /// 1. `[writable]` vote to delete, PDA
        /// 2. '[writable]' vote counter, PDA
        /// 3. '[]' Clock, PDA
        DeleteVote { admin: [u8; 32] },

        /// Create vote counter.
        /// Accounts:
        /// 0. `[signer]` admin
        /// 2. '[writable]' vote counter, PDA
        /// 3. `[]` Rent sysvar, PDA
        /// 4. `[]` System program, PDA
        CreateVoteCounter,
    }

    #[derive(BorshSerialize, BorshDeserialize, Debug, Clone,
PartialEq)]
    pub enum Direction {
        For,
        Against,
    }

    impl VoteInstruction {
        pub fn delete(admin: &Pubkey, vote: &Pubkey) ->
Instruction {
            let (vote_counter_pubkey, _) =
VoteCounter::get_vote_pubkey_with_bump();
            Instruction::new_with_borsh(
                id(),
                &VoteInstruction::DeleteVote { admin:
admin.to_bytes() },
                vec![
                    AccountMeta::new_readonly(*admin, true),
                    AccountMeta::new(*vote, false),
                    AccountMeta::new(vote_counter_pubkey,
false),

AccountMeta::new_readonly(sysvar::clock::id(), false),
                ],
            )
        }

        pub fn vote(user: &Pubkey, vote: &Pubkey, direction:
Direction) -> Instruction {
            let user_votes_pubkey =
UserVotes::get_uservote_pubkey(user, vote);
            Instruction::new_with_borsh(
                id(),
                &VoteInstruction::Vote { direction:
(direction) },
```

```
                vec![
                    AccountMeta::new_readonly(*user, true),
                    AccountMeta::new(user_votes_pubkey,
false),
                    AccountMeta::new(*vote, false),

AccountMeta::new_readonly(sysvar::rent::id(), false),

AccountMeta::new_readonly(system_program::id(), false),
                ],
            )
        }

        pub   fn   create_vote_counter(admin:   &Pubkey)   ->
Instruction {
            let       (vote_counter_pubkey,       _)       =
VoteCounter::get_vote_pubkey_with_bump();
            Instruction::new_with_borsh(
                id(),
                &VoteInstruction::CreateVoteCounter,
                vec![
                    AccountMeta::new(*admin, true),
                    AccountMeta::new(vote_counter_pubkey,
false),

AccountMeta::new_readonly(sysvar::rent::id(), false),

AccountMeta::new_readonly(system_program::id(), false),
                ],
            )
        }

        pub fn create_vote(admin: &Pubkey, vote_seed: &Pubkey)
-> Instruction {
            let          (vote_pubkey,          _)          =
Vote::get_vote_pubkey_with_bump(vote_seed);
            let       (vote_counter_pubkey,       _)       =
VoteCounter::get_vote_pubkey_with_bump();
            Instruction::new_with_borsh(
                id(),
                &VoteInstruction::CreateVote   {   vote_seed:
*vote_seed },
                vec![
                    AccountMeta::new(*admin, true),
                    AccountMeta::new(vote_pubkey, false),
                    AccountMeta::new(vote_counter_pubkey,
false),

AccountMeta::new_readonly(sysvar::rent::id(), false),

AccountMeta::new_readonly(system_program::id(), false),

AccountMeta::new_readonly(sysvar::clock::id(), false),
                ],
            )
```

```
        }
    }
```

Вміст файлу lib.rs для смарт-контакту, який реалізовує систему голосування

```
pub mod error;
pub mod instruction;
pub mod processor;
pub mod state;

#[cfg(not(feature = "no-entrypoint"))]
pub mod entrypoint;

pub const VOTE_SEED: &str = "vote";
pub const SETTINGS_SEED: &str = "settings";
solana_program::declare_id!("78yZvMzqAFzSHJrLNVWfqLRFFQ5Z
CGzNXB4PBxmp6z5Y");
```

Вміст файлу processor.rs для смарт-контакту, який реалізовує систему голосування

```
use borsh::{BorshDeserialize, BorshSerialize};
use     solana_program::account_info::{next_account_info,
AccountInfo};
use solana_program::entrypoint::ProgramResult;
use solana_program::program::invoke_signed;
use solana_program::pubkey::Pubkey;
use   solana_program::sysvar::{clock::Clock,   rent::Rent,
Sysvar};
use solana_program::{msg, system_instruction};

use crate::error::VoteError;
use crate::instruction::{Direction, VoteInstruction};
use    crate::state::{UserVotes,    Vote,    VoteCounter,
VoteStatus};
use crate::{id, SETTINGS_SEED, VOTE_SEED};

pub struct Processor;

pub const MAX_VOTES: u8 = 10;
pub const TIME_TO_LIVE: u64 = 10;

impl Processor {
    pub   fn   process(_program_id:   &Pubkey,   accounts:
&[AccountInfo], input: &[u8]) -> ProgramResult {
        let                instruction                    =
VoteInstruction::try_from_slice(input)?;
        match instruction {
```

```
                VoteInstruction::Vote { direction } =>
Self::process_vote(direction, accounts),
                VoteInstruction::CreateVote { vote_seed } =>
Self::process_create(accounts, vote_seed),
                VoteInstruction::DeleteVote { admin } =>
Self::process_delete(accounts, admin),
                VoteInstruction::CreateVoteCounter =>
Self::process_create_counter(accounts),
            }
        }

        fn process_vote(direction: Direction, accounts:
&[AccountInfo]) -> ProgramResult {
            let acc_iter = &mut accounts.iter();
            let user_info = next_account_info(acc_iter)?;
            let participate_info =
next_account_info(acc_iter)?;
            let vote_info = next_account_info(acc_iter)?;
            let rent_info = next_account_info(acc_iter)?;
            let system_program_info =
next_account_info(acc_iter)?;

            if !user_info.is_signer {
                return Err(VoteError::SignedRequired.into());
            }

            let (participate_pubkey, bump_seed) =

UserVotes::get_uservote_pubkey_with_bump(user_info.key,
vote_info.key);

            if participate_pubkey != *participate_info.key {
                return
Err(VoteError::WrongUserVotePDA.into());
            }

            if participate_info.data_is_empty() {
                let participate = UserVotes { is_voted: false
};
                let space = participate.try_to_vec()?.len();
                let rent =
&Rent::from_account_info(rent_info)?;
                let lamports = rent.minimum_balance(space);
                let signer_seeds: &[&[_]] = &[
                    &user_info.key.to_bytes(),
                    &vote_info.key.to_bytes(),
                    VOTE_SEED.as_bytes(),
                    &[bump_seed],
                ];
                invoke_signed(
                    &system_instruction::create_account(
                        user_info.key,
                        &participate_pubkey,
                        lamports,
                        space as u64,
```

```
                        &id(),
                    ),
                    &[user_info.clone(),
participate_info.clone(), system_program_info.clone()],
                    &[signer_seeds],
                )?;
                let _ = participate.serialize(&mut &mut
participate_info.data.borrow_mut()[..]);
            }

            let mut participation =
UserVotes::try_from_slice(&participate_info.data.borrow())?;
            let mut vote =
Vote::try_from_slice(&vote_info.data.borrow())?;

            if participation.is_voted {
                return
Err(VoteError::DoubleParticipate.into());
            }

            if vote.status != VoteStatus::Alive {
                return
Err(VoteError::CloseVoteParticipate.into());
            }

            participation.is_voted = true;

            match direction {
                Direction::For => vote.all_votes_for += 1,
                Direction::Against => vote.all_votes_against
+= 1,
            }

            let _ = participation.serialize(&mut &mut
participate_info.data.borrow_mut()[..]);
            let _ = vote.serialize(&mut &mut
vote_info.data.borrow_mut()[..]);

            Ok(())
        }

        fn process_create(accounts: &[AccountInfo],
vote_seed: Pubkey) -> ProgramResult {
            let acc_iter = &mut accounts.iter();
            let admin_info = next_account_info(acc_iter)?;
            let vote_info = next_account_info(acc_iter)?;
            let vote_counter_info =
next_account_info(acc_iter)?;
            let rent_info = next_account_info(acc_iter)?;
            let system_program_info =
next_account_info(acc_iter)?;
            let clock_sysvar_info =
next_account_info(acc_iter)?;
```

```
            let       (vote_pubkey,       bump_seed)       =
Vote::get_vote_pubkey_with_bump(&vote_seed);

            if vote_pubkey != *vote_info.key {
                return
Err(VoteError::WrongVoteDefine.into());
            }

            if !admin_info.is_signer {
                return Err(VoteError::AdminRequired.into());
            }

            let         mut          vote_counter          =
VoteCounter::try_from_slice(&vote_counter_info.data.borrow())?
;
            let                    time                    =
Clock::from_account_info(clock_sysvar_info)?.slot;

            if vote_counter.counter >= MAX_VOTES {
                return Err(VoteError::MaxVote.into());
            }

            let  vote  =  Vote::new(admin_info.key.to_bytes(),
time);
            let space = vote.try_to_vec()?.len();
            let rent = &Rent::from_account_info(rent_info)?;
            let lamports = rent.minimum_balance(space);
            let        signer_seeds:        &[&[_]]        =
&[&vote_seed.to_bytes(), &[bump_seed]];
            invoke_signed(
                &system_instruction::create_account(
                    admin_info.key,
                    &vote_pubkey,
                    lamports,
                    space as u64,
                    &id(),
                ),
                &[admin_info.clone(),      vote_info.clone(),
system_program_info.clone()],
                &[signer_seeds],
            )?;
            vote_counter.counter += 1;
            let   _    =   vote_counter.serialize(&mut   &mut
vote_counter_info.data.borrow_mut()[..]);

            if  vote.admin  !=  admin_info.key.to_bytes()  &&
vote.admin != [0; 32] {
                return Err(VoteError::AdminRequired.into());
            }

            let      _      =     vote.serialize(&mut     &mut
vote_info.data.borrow_mut()[..]);

            Ok(())
        }
```

```
fn process_delete(accounts: &[AccountInfo], admin: [u8; 32]) -> ProgramResult {
    msg!("process_delete: admin={:?}", admin,);
    let acc_iter = &mut accounts.iter();
    let admin_info = next_account_info(acc_iter)?;
    let vote_info = next_account_info(acc_iter)?;
    let vote_counter_info = next_account_info(acc_iter)?;
    let clock_sysvar_info = next_account_info(acc_iter)?;

    if !admin_info.is_signer {
        return Err(VoteError::AdminRequired.into());
    }

    let mut vote = Vote::try_from_slice(&vote_info.data.borrow())?;
    let mut vote_counter = VoteCounter::try_from_slice(&vote_counter_info.data.borrow())?;
    let clock = Clock::from_account_info(clock_sysvar_info)?;

    if vote.admin != admin_info.key.to_bytes() {
        return Err(VoteError::AdminRequired.into());
    }
    msg!("clock.slot: {}, vote.clock: {}", clock.slot, vote.clock);
    if clock.slot - vote.clock >= TIME_TO_LIVE {
        vote.status = VoteStatus::Closed;
        vote_counter.counter -= 1;
    }

    let _ = vote.serialize(&mut &mut vote_info.data.borrow_mut()[..]);
    let _ = vote_counter.serialize(&mut &mut vote_counter_info.data.borrow_mut()[..]);

    msg!("process_delete: done");
    Ok(())
}

fn process_create_counter(accounts: &[AccountInfo]) -> ProgramResult {
    let acc_iter = &mut accounts.iter();
    let admin_info = next_account_info(acc_iter)?;
    let vote_counter_info = next_account_info(acc_iter)?;
    let rent_info = next_account_info(acc_iter)?;
    let system_program_info = next_account_info(acc_iter)?;

    if !admin_info.is_signer {
        return Err(VoteError::AdminRequired.into());
```

```
            }

            let       (vote_pubkey,       bump_seed)       =
VoteCounter::get_vote_pubkey_with_bump();

            if !vote_counter_info.data_is_empty() {
                return Err(VoteError::DoubleCounter.into());
            }

            let vote_counter = VoteCounter { counter: 0 };
            let space = vote_counter.try_to_vec()?.len();
            let rent = &Rent::from_account_info(rent_info)?;
            let lamports = rent.minimum_balance(space);
            let       signer_seeds:       &[&[_]]       =
&[SETTINGS_SEED.as_bytes(), &[bump_seed]];
            invoke_signed(
                &system_instruction::create_account(
                    admin_info.key,
                    &vote_pubkey,
                    lamports,
                    space as u64,
                    &id(),
                ),
                &[admin_info.clone(),
vote_counter_info.clone(), system_program_info.clone()],
                &[signer_seeds],
            )?;

            let   _   =   vote_counter.serialize(&mut   &mut
vote_counter_info.data.borrow_mut()[..]);

            Ok(())
        }
    }
```

Вміст файлу state.rs для смарт-контакту, який реалізовує систему голосування

```
    use borsh::{BorshDeserialize, BorshSerialize};
    use solana_program::pubkey::Pubkey;

    use crate::{id, SETTINGS_SEED, VOTE_SEED};

    #[derive(BorshSerialize, BorshDeserialize, Debug)]
    pub struct UserVotes {
        pub is_voted: bool,
    }

    impl UserVotes {
        pub  fn  get_uservote_pubkey_with_bump(user:  &Pubkey,
vote: &Pubkey) -> (Pubkey, u8) {
            Pubkey::find_program_address(
```

```
                &[&user.to_bytes(),        &vote.to_bytes(),
VOTE_SEED.as_bytes()],
                &id(),
            )
        }

        pub   fn   get_uservote_pubkey(user:   &Pubkey,   vote:
&Pubkey) -> Pubkey {
            let           (pubkey,           _)            =
Self::get_uservote_pubkey_with_bump(user, vote);
            pubkey
        }
    }

    #[derive(BorshSerialize,     BorshDeserialize,     Debug,
PartialEq)]
    pub enum VoteStatus {
        Alive,
        Closed,
    }

    #[derive(BorshSerialize, BorshDeserialize, Debug)]
    pub struct Vote {
        pub admin: [u8; 32],

        pub all_votes_for: u32,

        pub all_votes_against: u32,

        pub clock: u64,

        pub status: VoteStatus,
    }

    impl Vote {
        pub  fn  get_vote_pubkey_with_bump(vote_seed:  &Pubkey)
-> (Pubkey, u8) {

Pubkey::find_program_address(&[&vote_seed.to_bytes()], &id())
        }

        pub fn get_vote_pubkey(vote_seed: &Pubkey) -> Pubkey
{
            let           (pubkey,           _)            =
Self::get_vote_pubkey_with_bump(vote_seed);
            pubkey
        }

        pub fn new(admin: [u8; 32], clock: u64) -> Self {
            Self { admin, all_votes_for: 0, all_votes_against:
0, clock, status: VoteStatus::Alive }
        }
    }

    #[derive(BorshSerialize, BorshDeserialize, Debug)]
```

```
pub struct VoteCounter {
    pub counter: u8,
}

impl VoteCounter {
    pub fn get_vote_pubkey_with_bump() -> (Pubkey, u8) {

Pubkey::find_program_address(&[SETTINGS_SEED.as_bytes()],
&id())
    }

    pub fn get_vote_pubkey() -> Pubkey {
        let          (pubkey,          _)          =
Self::get_vote_pubkey_with_bump();
        pubkey
    }

    pub fn is_ok_vote_pubkey(vote_pubkey: &Pubkey) -> bool
{
        let          (pubkey,          _)          =
Self::get_vote_pubkey_with_bump();
        pubkey.to_bytes() == vote_pubkey.to_bytes()
    }
}
```

Вміст файлу functional.rs для смарт-контакту, який реалізовує систему голосування

```
#![cfg(feature = "test-bpf")]
use std::assert_eq;

use borsh::{BorshDeserialize};
use solana_program::{
    pubkey::Pubkey,
    system_instruction,
};
use solana_program_test::{
    processor,
    tokio::{
        self,
    },
    ProgramTest, ProgramTestContext,
};

use solana_sdk::signature::{Keypair, Signer};
use solana_sdk::transaction::Transaction;
use   voting::state::{UserVotes,   Vote,   VoteCounter,
VoteStatus};
use voting::{
    entrypoint::process_instruction,
    id,
    instruction::{Direction, VoteInstruction},
```

```
    };

    struct Env {
        ctx: ProgramTestContext,
        admin: Keypair,
        user_01: Keypair,
        user_02: Keypair,
        user_03: Keypair,
    }

    impl Env {
        async fn new() -> Self {
            let program_test = ProgramTest::new("voting",
id(), processor!(process_instruction));
            let mut ctx =
program_test.start_with_context().await;

            let admin = Keypair::new();
            let user_01 = Keypair::new();
            let user_02 = Keypair::new();
            let user_03 = Keypair::new();

            ctx.banks_client

.process_transaction(Transaction::new_signed_with_payer(
                    &[
                        system_instruction::transfer(
                            &ctx.payer.pubkey(),
                            &admin.pubkey(),
                            1_000_000_000,
                        ),
                        system_instruction::transfer(
                            &ctx.payer.pubkey(),
                            &user_01.pubkey(),
                            1_000_000_000,
                        ),
                        system_instruction::transfer(
                            &ctx.payer.pubkey(),
                            &user_02.pubkey(),
                            1_000_000_000,
                        ),
                        system_instruction::transfer(
                            &ctx.payer.pubkey(),
                            &user_03.pubkey(),
                            1_000_000_000,
                        ),
                    ],
                    Some(&ctx.payer.pubkey()),
                    &[&ctx.payer],
                    ctx.last_blockhash,
                ))
                .await
                .unwrap();

            let tx = Transaction::new_signed_with_payer(
```

```
&[VoteInstruction::create_vote_counter(&admin.pubkey())],
                 Some(&admin.pubkey()),
                 &[&admin],
                 ctx.last_blockhash,
             );

ctx.banks_client.process_transaction(tx).await.unwrap();

             let acc =

ctx.banks_client.get_account(VoteCounter::get_vote_pubkey()).a
wait.unwrap().unwrap();
             let                vote_counter                  =
VoteCounter::try_from_slice(acc.data.as_slice()).unwrap();
             assert_eq!(vote_counter.counter, 0);

             Env { ctx, admin, user_01, user_02, user_03 }
         }
     }

     // test of 1 user vote
     #[tokio::test]
     async fn test_vote() {
         let mut env = Env::new().await;
         let vote_seed = Pubkey::new_unique();
         let tx = Transaction::new_signed_with_payer(

&[VoteInstruction::create_vote(&env.admin.pubkey(),
&vote_seed)],
             Some(&env.admin.pubkey()),
             &[&env.admin],
             env.ctx.last_blockhash,
         );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

         let tx = Transaction::new_signed_with_payer(
             &[VoteInstruction::vote(
                 &env.user_01.pubkey(),
                 &Vote::get_vote_pubkey(&vote_seed),
                 Direction::For,
             )],
             Some(&env.user_01.pubkey()),
             &[&env.user_01],
             env.ctx.last_blockhash,
         );

env.ctx.banks_client.process_transaction(tx).await.unwrap();

         let acc = env
             .ctx
             .banks_client
             .get_account(UserVotes::get_uservote_pubkey(
```

```
                &env.user_01.pubkey(),
                &Vote::get_vote_pubkey(&vote_seed),
            ))
            .await
            .unwrap()
            .unwrap();
        let                      user_votes                     =
UserVotes::try_from_slice(&acc.data).unwrap();

        let acc =

env.ctx.banks_client.get_account(Vote::get_vote_pubkey(&vote_s
eed)).await.unwrap().unwrap();

        let                         vote                        =
Vote::try_from_slice(acc.data.as_slice()).unwrap();
        assert_eq!(vote.all_votes_for, 1);
        assert_eq!(user_votes.is_voted, true);
    }

    // test of 3 users vote
    #[tokio::test]
    async fn test_vote_third() {
        let mut env = Env::new().await;
        let vote_seed = Pubkey::new_unique();
        let tx = Transaction::new_signed_with_payer(

&[VoteInstruction::create_vote(&env.admin.pubkey(),
&vote_seed)],
            Some(&env.admin.pubkey()),
            &[&env.admin],
            env.ctx.last_blockhash,
        );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let tx = Transaction::new_signed_with_payer(
            &[VoteInstruction::vote(
                &env.user_01.pubkey(),
                &Vote::get_vote_pubkey(&vote_seed),
                Direction::For,
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01],
            env.ctx.last_blockhash,
        );

env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let tx = Transaction::new_signed_with_payer(
            &[VoteInstruction::vote(
                &env.user_02.pubkey(),
                &Vote::get_vote_pubkey(&vote_seed),
                Direction::For,
```

```
                )],
                Some(&env.user_02.pubkey()),
                &[&env.user_02],
                env.ctx.last_blockhash,
            );

env.ctx.banks_client.process_transaction(tx).await.unwrap();

            let tx = Transaction::new_signed_with_payer(
                &[VoteInstruction::vote(
                    &env.user_03.pubkey(),
                    &Vote::get_vote_pubkey(&vote_seed),
                    Direction::Against,
                )],
                Some(&env.user_03.pubkey()),
                &[&env.user_03],
                env.ctx.last_blockhash,
            );

env.ctx.banks_client.process_transaction(tx).await.unwrap();

            let acc =

env.ctx.banks_client.get_account(Vote::get_vote_pubkey(&vote_s
eed)).await.unwrap().unwrap();

            let                         vote                         =
Vote::try_from_slice(acc.data.as_slice()).unwrap();
            assert_eq!(vote.all_votes_for, 2);
            assert_eq!(vote.all_votes_against, 1);
        }

        // test delete with time wait
        #[tokio::test]
        async fn test_delete() {
            let mut env = Env::new().await;

            let vote_seed = Pubkey::new_unique();

            let tx = Transaction::new_signed_with_payer(

&[VoteInstruction::create_vote(&env.admin.pubkey(),
&vote_seed)],
                Some(&env.admin.pubkey()),
                &[&env.admin],
                env.ctx.last_blockhash,
            );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

            env.ctx.warp_to_slot(11);

            let tx = Transaction::new_signed_with_payer(
```

```
            &[VoteInstruction::delete(&env.admin.pubkey(),
&Vote::get_vote_pubkey(&vote_seed))],
            Some(&env.admin.pubkey()),
            &[&env.admin],
            env.ctx.last_blockhash,
        );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let acc =

env.ctx.banks_client.get_account(Vote::get_vote_pubkey(&vote_s
eed)).await.unwrap().unwrap();
        let                    vote                    =
Vote::try_from_slice(acc.data.as_slice()).unwrap();

        assert_eq!(vote.status, VoteStatus::Closed);
    }

    // test delete without time wait
    #[tokio::test]
    async fn test_delete_no_wait() {
        let mut env = Env::new().await;

        let vote_seed = Pubkey::new_unique();

        let tx = Transaction::new_signed_with_payer(

&[VoteInstruction::create_vote(&env.admin.pubkey(),
&vote_seed)],
            Some(&env.admin.pubkey()),
            &[&env.admin],
            env.ctx.last_blockhash,
        );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let tx = Transaction::new_signed_with_payer(
            &[VoteInstruction::delete(&env.admin.pubkey(),
&Vote::get_vote_pubkey(&vote_seed))],
            Some(&env.admin.pubkey()),
            &[&env.admin],
            env.ctx.last_blockhash,
        );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let acc =

env.ctx.banks_client.get_account(Vote::get_vote_pubkey(&vote_s
eed)).await.unwrap().unwrap();
```

```
        let                         vote                         =
Vote::try_from_slice(acc.data.as_slice()).unwrap();

        assert_eq!(vote.status, VoteStatus::Alive);
    }

    // test user to double participate in 1 vote
    #[should_panic]
    #[tokio::test]
    async fn double_vote() {
        let mut env = Env::new().await;
        let vote_seed = Pubkey::new_unique();
        let tx = Transaction::new_signed_with_payer(

&[VoteInstruction::create_vote(&env.admin.pubkey(),
&vote_seed)],
            Some(&env.admin.pubkey()),
            &[&env.admin],
            env.ctx.last_blockhash,
        );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let tx = Transaction::new_signed_with_payer(
            &[VoteInstruction::vote(
                &env.user_01.pubkey(),
                &Vote::get_vote_pubkey(&vote_seed),
                Direction::For,
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01],
            env.ctx.last_blockhash,
        );

env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let tx = Transaction::new_signed_with_payer(
            &[VoteInstruction::vote(
                &env.user_01.pubkey(),
                &Vote::get_vote_pubkey(&vote_seed),
                Direction::Against,
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01],
            env.ctx.last_blockhash,
        );

env.ctx.banks_client.process_transaction(tx).await.unwrap();
    }

    // test user (not admin) try to delete vote
    #[should_panic]
    #[tokio::test]
    async fn not_admin() {
```

```
        let mut env = Env::new().await;
        let vote_seed = Pubkey::new_unique();
        let tx = Transaction::new_signed_with_payer(

&[VoteInstruction::create_vote(&env.admin.pubkey(),
&vote_seed)],
            Some(&env.admin.pubkey()),
            &[&env.admin],
            env.ctx.last_blockhash,
        );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let tx = Transaction::new_signed_with_payer(
            &[VoteInstruction::delete(&env.user_01.pubkey(),
&Vote::get_vote_pubkey(&vote_seed))],
            Some(&env.user_01.pubkey()),
            &[&env.user_01],
            env.ctx.last_blockhash,
        );

env.ctx.banks_client.process_transaction(tx).await.unwrap();
    }

    // test of vote create
    #[tokio::test]
    async fn test_create_vote() {
        let mut env = Env::new().await;

        let vote_seed = Pubkey::new_unique();

        let tx = Transaction::new_signed_with_payer(

&[VoteInstruction::create_vote(&env.admin.pubkey(),
&vote_seed)],
            Some(&env.admin.pubkey()),
            &[&env.admin],
            env.ctx.last_blockhash,
        );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let acc =

env.ctx.banks_client.get_account(Vote::get_vote_pubkey(&vote_s
eed)).await.unwrap().unwrap();
        let                      vote                       =
Vote::try_from_slice(acc.data.as_slice()).unwrap();
        assert_eq!(vote.admin,
env.admin.pubkey().to_bytes());
    }
```

Вміст файлу Cargo.toml для смарт-контакту, який реалізовує систему голосування

```
[package]
name = "voting"
version = "0.1.0"
authors = ["RequescoS"]
edition = "2021"

[features]
no-entrypoint = []
test-bpf = []

[dependencies]
borsh = "0.9.3"
thiserror = "1.0.30"
solana-program = "1.9.9"


[dev-dependencies]
solana-program-test = "1.9.9"
solana-sdk = "1.9.9"


[lib]
crate-type = ["cdylib", "lib"]
```

Додаток В

Вміст файлу entrypoint.rs для смарт-контакту, який реалізовує liqiudity pool

```
use solana_program::{
    account_info::AccountInfo,                    entrypoint,
entrypoint::ProgramResult, pubkey::Pubkey,
};

use crate::processor::Processor;

entrypoint!(process_instruction);

pub fn process_instruction(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    instruction_data: &[u8],
) -> ProgramResult {
    Processor::process(program_id, accounts, instruction_data)
}
```

Вміст файлу error.rs для смарт-контакту, який реалізовує liqiudity pool

```
use solana_program::program_error::ProgramError;
use thiserror::Error;

#[derive(Error, Debug, Copy, Clone)]
pub enum PoolError {
    #[error("Trying to provide liquidity without offer both
tokens")]
    ZeroProvide,

    #[error("Trying to provide more tokens than possessed")]
    OverProvide,

    #[error("Wrong ratio of providing tokens")]
    SlippageFail,

    #[error("Trying to buy more tokens than present in the
pool")]
    OverBuy,

    #[error("Trying to buy more tokens than can to pay")]
    TooMuchBuy,

    #[error("Trying   to   withdraw   more   liquidity   than
possessed")]
    OverWithdraw,
```

```
    #[error("User signature is required")]
    SignedRequired,

    #[error("Wrong withdraw account")]
    WrongWithdraw,
}

impl From<PoolError> for ProgramError {
    fn from(e: PoolError) -> Self {
        ProgramError::Custom(e as u32)
    }
}
```

Вміст файлу instruction.rs для смарт-контакту, який реалізовує liqiudity pool

```
use borsh::{BorshDeserialize, BorshSerialize};
use solana_program::{
    instruction::{AccountMeta, Instruction},
    pubkey::Pubkey,
    system_program, sysvar,
};

use spl_token;

use crate::{
    id,
    state::{TotalCommision, WithdrawedFee},
};

#[derive(BorshSerialize, BorshDeserialize, Debug, Clone,
PartialEq)]
pub enum PoolInstruction {
    /// Provide liquidity.
    /// Accounts:
    /// 0. `[signer]` user`s account
    /// 1. `[]` user`s token x account
    /// 2. `[]` user`s token y account
    /// 3. `[]` user`s token lp account
    /// 4. `[]` pool`s token x account
    /// 5. `[]` pool`s token y account
    /// 6. `[]` mint lp token account
    /// 7. `[signer]` minter account
    /// 8. `[]` token program account, PDA
    ProvideLiquidity { x_amount: u64, y_amount: u64 },

    /// Swap tokens.
    /// Accounts:
    /// 0. `[signer]` user`s account
```

```
    /// 1. `[]` user`s token from swap account
    /// 2. `[]` user`s token to swap account
    /// 3. `[]` pool`s token from swap account
    /// 4. `[]` pool`s token to swap account
    /// 5. `[]` commision from account
    /// 6. `[signer]` minter account
    /// 7. `[]` token program account, PDA
    SwapTokens { amount: u64 },

    /// Withdraw liquidity.
    /// Accounts:
    /// 0. `[signer]` user`s account
    /// 1. `[]` user`s token x account
    /// 2. `[]` user`s token y account
    /// 3. `[]` user`s token lp account
    /// 4. `[]` pool`s token x account
    /// 5. `[]` pool`s token y account
    /// 6. `[]` mint lp token account
    /// 7. `[signer]` minter account
    /// 8. `[]` token program account, PDA
    WithdrawLiquidity { amount: u64 },

    /// Withdraw fee.
    /// Accounts:
    /// 0. `[signer]` user`s account
    /// 1. '[]' user`s withdraw info account
    /// 2. `[]` user`s token x account
    /// 3. `[]` user`s token y account
    /// 4. `[]` user`s token lp account
    /// 5. `[]` mint lp token account
    /// 6. '[]' commision token x account
    /// 7. '[]' commision token y account
    /// 8. `[signer]` minter account
    /// 9. `[]` token program account, PDA
    /// 10. `[]` Rent sysvar, PDA
    /// 11. `[]` System program, PDA
    WithdrawFee,
}

impl PoolInstruction {
    pub fn provide_liquidity(
        user: &Pubkey,
        admin: &Pubkey,
        x_user_token: &Pubkey,
        y_user_token: &Pubkey,
        lp_user_token: &Pubkey,
        pool_x_token: &Pubkey,
        pool_y_token: &Pubkey,
        mint_lp_token: &Pubkey,
        commision_x_token: &Pubkey,
        commision_y_token: &Pubkey,
        x_amount: u64,
        y_amount: u64,
    ) -> Instruction {
```

```
            let              withdraw_pubkey            =
WithdrawedFee::get_withdraw_pubkey(user);
            let               total_pubkey              =
TotalCommision::get_total_pubkey();
            Instruction::new_with_borsh(
                id(),
                &PoolInstruction::ProvideLiquidity          {
x_amount, y_amount },
                vec![
                    AccountMeta::new_readonly(*user, true),
                    AccountMeta::new(withdraw_pubkey, false),
                    AccountMeta::new(*x_user_token, false),
                    AccountMeta::new(*y_user_token, false),
                    AccountMeta::new(*lp_user_token, false),
                    AccountMeta::new(*pool_x_token, false),
                    AccountMeta::new(*pool_y_token, false),
                    AccountMeta::new(*mint_lp_token, false),
                    AccountMeta::new(*commision_x_token,
false),
                    AccountMeta::new(*commision_y_token,
false),
                    AccountMeta::new(total_pubkey, false),
                    AccountMeta::new_readonly(*admin, true),

AccountMeta::new_readonly(spl_token::id(), false),

AccountMeta::new_readonly(sysvar::rent::id(), false),

AccountMeta::new_readonly(system_program::id(), false),
                ],
            )
        }

        pub fn withdraw_liquidity(
            user: &Pubkey,
            admin: &Pubkey,
            x_user_token: &Pubkey,
            y_user_token: &Pubkey,
            lp_user_token: &Pubkey,
            pool_x_token: &Pubkey,
            pool_y_token: &Pubkey,
            mint_lp_token: &Pubkey,
            commision_x_token: &Pubkey,
            commision_y_token: &Pubkey,
            amount: u64,
        ) -> Instruction {
            let              withdraw_pubkey            =
WithdrawedFee::get_withdraw_pubkey(user);
            let               total_pubkey              =
TotalCommision::get_total_pubkey();
            Instruction::new_with_borsh(
                id(),
                &PoolInstruction::WithdrawLiquidity { amount
},
                vec![
```

```
                    AccountMeta::new_readonly(*user, true),
                    AccountMeta::new(withdraw_pubkey, false),
                    AccountMeta::new(*x_user_token, false),
                    AccountMeta::new(*y_user_token, false),
                    AccountMeta::new(*lp_user_token, false),
                    AccountMeta::new(*pool_x_token, false),
                    AccountMeta::new(*pool_y_token, false),
                    AccountMeta::new(*mint_lp_token, false),
                    AccountMeta::new(*commision_x_token,
false),
                    AccountMeta::new(*commision_y_token,
false),
                    AccountMeta::new(total_pubkey, false),
                    AccountMeta::new_readonly(*admin, true),

AccountMeta::new_readonly(spl_token::id(), false),

AccountMeta::new_readonly(sysvar::rent::id(), false),

AccountMeta::new_readonly(system_program::id(), false),
                ],
            )
        }

        pub fn swap_tokens(
            user: &Pubkey,
            admin: &Pubkey,
            from_user_token: &Pubkey,
            to_user_token: &Pubkey,
            pool_from_token: &Pubkey,
            pool_to_token: &Pubkey,
            commision_from_token: &Pubkey,
            amount: u64,
        ) -> Instruction {
            Instruction::new_with_borsh(
                id(),
                &PoolInstruction::SwapTokens { amount },
                vec![
                    AccountMeta::new_readonly(*user, true),
                    AccountMeta::new(*from_user_token,
false),
                    AccountMeta::new(*to_user_token, false),
                    AccountMeta::new(*pool_from_token,
false),
                    AccountMeta::new(*pool_to_token, false),
                    AccountMeta::new(*commision_from_token,
false),
                    AccountMeta::new_readonly(*admin, true),

AccountMeta::new_readonly(spl_token::id(), false),
                ],
            )
        }

        pub fn withdraw_fee(
```

```
            user: &Pubkey,
            admin: &Pubkey,
            x_user_token: &Pubkey,
            y_user_token: &Pubkey,
            lp_user_token: &Pubkey,
            pool_x_token: &Pubkey,
            pool_y_token: &Pubkey,
            mint_lp_token: &Pubkey,
            commision_x_token: &Pubkey,
            commision_y_token: &Pubkey,
        ) -> Instruction {
            let              withdraw_pubkey              =
WithdrawedFee::get_withdraw_pubkey(user);
            let               total_pubkey              =
TotalCommision::get_total_pubkey();
            Instruction::new_with_borsh(
                id(),
                &PoolInstruction::WithdrawFee,
                vec![
                    AccountMeta::new_readonly(*user, true),
                    AccountMeta::new(withdraw_pubkey, false),
                    AccountMeta::new(*x_user_token, false),
                    AccountMeta::new(*y_user_token, false),
                    AccountMeta::new(*lp_user_token, false),
                    AccountMeta::new(*pool_x_token, false),
                    AccountMeta::new(*pool_y_token, false),
                    AccountMeta::new(*mint_lp_token, false),
                    AccountMeta::new(*commision_x_token,
false),
                    AccountMeta::new(*commision_y_token,
false),
                    AccountMeta::new(total_pubkey, false),
                    AccountMeta::new_readonly(*admin, true),

AccountMeta::new_readonly(spl_token::id(), false),

AccountMeta::new_readonly(sysvar::rent::id(), false),

AccountMeta::new_readonly(system_program::id(), false),
                ],
            )
        }
    }
```

Вміст файлу lib.rs для смарт-контакту, який реалізовує liqiudity pool

```
pub mod error;
pub mod instruction;
pub mod processor;
pub mod state;

#[cfg(not(feature = "no-entrypoint"))]
pub mod entrypoint;
```

```
pub const POOL_SEED: &str = "liquidity pool";
solana_program::declare_id!("78yZvMzqAFzSHJrLNVWfqLRFFQ5ZCGzNXB4PBxmp6z5Y");
```

Вміст файлу processor.rs для смарт-контакту, який реалізовує liqiudity pool

```
use borsh::{BorshDeserialize, BorshSerialize};
use solana_program::account_info::{next_account_info, AccountInfo};
use solana_program::entrypoint::ProgramResult;
use solana_program::program::invoke_signed;
use solana_program::pubkey::Pubkey;
use solana_program::sysvar::{rent::Rent, Sysvar};
use solana_program::{msg, program::invoke, program_pack::Pack, system_instruction};

use crate::error::PoolError;
use crate::instruction::PoolInstruction;
use crate::state::{TotalCommision, WithdrawedFee};
use crate::{id, POOL_SEED};

use spl_token::state::{Account, Mint};

pub const COMMISION_PERCENT: u64 = 3;
pub const SLIPPAGE_TOLERANCE: u64 = 1;

pub struct Processor;

impl Processor {
    pub fn process(_program_id: &Pubkey, accounts: &[AccountInfo], input: &[u8]) -> ProgramResult {
        let instruction = PoolInstruction::try_from_slice(input)?;
        match instruction {
            PoolInstruction::ProvideLiquidity { x_amount, y_amount } => {
                Self::provide_liquidity(accounts, x_amount, y_amount)
            }
            PoolInstruction::SwapTokens { amount } => Self::swap_tokens(accounts, amount),
            PoolInstruction::WithdrawLiquidity { amount } => {
                Self::withdraw_liquidity(accounts, amount)
            }
            PoolInstruction::WithdrawFee => Self::withdraw_fee(accounts),
        }
    }
```

```
         fn    provide_liquidity(accounts:    &[AccountInfo],
x_amount: u64, y_amount: u64) -> ProgramResult {
             msg!("Providing liquidity");

             let acc_iter = &mut accounts.iter();
             let user_info = next_account_info(acc_iter)?;
             let withdraw_info = next_account_info(acc_iter)?;
             let                x_user_token_info                =
next_account_info(acc_iter)?;
             let                y_user_token_info                =
next_account_info(acc_iter)?;
             let                 xy_lp_user_info                 =
next_account_info(acc_iter)?;
             let                pool_x_token_info                =
next_account_info(acc_iter)?;
             let                pool_y_token_info                =
next_account_info(acc_iter)?;
             let               mint_lp_token_info                =
next_account_info(acc_iter)?;
             let       current_comission_x_tokem_info       =
next_account_info(acc_iter)?;
             let       current_comission_y_tokem_info       =
next_account_info(acc_iter)?;
             let              total_commision_info              =
next_account_info(acc_iter)?;
             let admin_info = next_account_info(acc_iter)?;
             let token_info = next_account_info(acc_iter)?;
             let _ = next_account_info(acc_iter)?;
             let _ = next_account_info(acc_iter)?;

             if !user_info.is_signer {
                 return Err(PoolError::SignedRequired.into());
             }

             if x_amount == 0 || y_amount == 0 {
                 return Err(PoolError::ZeroProvide.into());
             }

             let                  x_user_token                  =
Account::unpack_from_slice(&x_user_token_info.data.borrow())?.
amount;
             let                  y_user_token                  =
Account::unpack_from_slice(&y_user_token_info.data.borrow())?.
amount;

             if   x_amount   >   x_user_token   ||   y_amount   >
y_user_token {
                 return Err(PoolError::OverProvide.into());
             }

             Self::withdraw_fee(accounts)?;

             let ix = spl_token::instruction::transfer(
                 token_info.key,
                 x_user_token_info.key,
```

```
                    pool_x_token_info.key,
                    user_info.key,
                    &[user_info.key],
                    x_amount,
                )?;
                invoke(
                    &ix,
                    &[
                        x_user_token_info.clone(),
                        pool_x_token_info.clone(),
                        user_info.clone(),
                        token_info.clone(),
                    ],
                )?;

                let iy = spl_token::instruction::transfer(
                    token_info.key,
                    y_user_token_info.key,
                    pool_y_token_info.key,
                    user_info.key,
                    &[user_info.key],
                    y_amount,
                )?;
                invoke(
                    &iy,
                    &[
                        y_user_token_info.clone(),
                        pool_y_token_info.clone(),
                        user_info.clone(),
                        token_info.clone(),
                    ],
                )?;

                let                  total_lp                  =
Mint::unpack_from_slice(&mint_lp_token_info.data.borrow())?;
                let                pool_x_token                =
Account::unpack_from_slice(&pool_x_token_info.data.borrow())?;
                let                pool_y_token                =
Account::unpack_from_slice(&pool_y_token_info.data.borrow())?;

                let new_lp: u64 = if total_lp.supply == 0 {
                    ((x_amount as f64) * (y_amount as f64)).sqrt()
as u64
                } else {
                    if
!Self::slippage_tolerance_check(pool_x_token,    pool_y_token,
x_amount, y_amount) {
                        return
Err(PoolError::SlippageFail.into());
                    }
                    std::cmp::min(
                        x_amount     *     total_lp.supply     /
(pool_x_token.amount - x_amount),
                        y_amount     *     total_lp.supply     /
(pool_y_token.amount - y_amount),
```

```
            )
        };

        msg!("mint!!!!!!!!!!!!!!!!!!!!!");

        let ilp = spl_token::instruction::mint_to(
            token_info.key,
            mint_lp_token_info.key,
            xy_lp_user_info.key,
            admin_info.key,
            &[admin_info.key],
            new_lp,
        )?;
        invoke(
            &ilp,
            &[
                mint_lp_token_info.clone(),
                xy_lp_user_info.clone(),
                admin_info.clone(),
                token_info.clone(),
            ],
        )?;

        let               total_commision                =
TotalCommision::try_from_slice(&total_commision_info.data.borr
ow())?;
        let token_x_commision =

Account::unpack_from_slice(&current_comission_x_tokem_info.dat
a.borrow())?.amount;
        let token_y_commision =

Account::unpack_from_slice(&current_comission_y_tokem_info.dat
a.borrow())?.amount;
        let                 total_lp                     =
Mint::unpack_from_slice(&mint_lp_token_info.data.borrow())?.su
pply;
        let                 user_lp                      =
Account::unpack_from_slice(&xy_lp_user_info.data.borrow())?.am
ount;

        let [x_amount, y_amount] = Self::liquidity_profit(
            user_lp,
            total_lp,
            token_x_commision                            +
total_commision.total_x_commision,
            token_y_commision                            +
total_commision.total_y_commision,
        );

        let          mut          withdraw               =
WithdrawedFee::try_from_slice(&withdraw_info.data.borrow())?;

        withdraw.user_x_withdraw = x_amount;
        withdraw.user_y_withdraw = y_amount;
```

```
            let _ = withdraw.serialize(&mut &mut
withdraw_info.data.borrow_mut()[..]);

            Ok(())
        }

        pub fn slippage_tolerance_check(
            pool_x_token: Account,
            pool_y_token: Account,
            x_amount: u64,
            y_amount: u64,
        ) -> bool {
            let start_ratio: f64 =
                (pool_x_token.amount - x_amount) as f64 /
(pool_y_token.amount - y_amount) as f64;
            let new_ratio: f64 = pool_x_token.amount as f64 /
pool_y_token.amount as f64;
            let slippage = 1.0 - new_ratio / start_ratio;
            if slippage.abs() > SLIPPAGE_TOLERANCE as f64 /
100.0 {
                false
            } else {
                true
            }
        }

        pub fn swap_tokens(accounts: &[AccountInfo], amount:
u64) -> ProgramResult {
            msg!("Swap tokens");

            let acc_iter = &mut accounts.iter();
            let user_info = next_account_info(acc_iter)?;
            let user_from_token_info =
next_account_info(acc_iter)?;
            let user_to_token_info =
next_account_info(acc_iter)?;
            let pool_from_token_info =
next_account_info(acc_iter)?;
            let pool_to_token_info =
next_account_info(acc_iter)?;
            let commision_info =
next_account_info(acc_iter)?;
            let admin_info = next_account_info(acc_iter)?;
            let token_info = next_account_info(acc_iter)?;

            if !user_info.is_signer {
                return Err(PoolError::SignedRequired.into());
            }

            let pool_from_token =
Account::unpack_from_slice(&pool_from_token_info.data.borrow()
)?;
```

```
            let                pool_to_token                =
Account::unpack_from_slice(&pool_to_token_info.data.borrow())?
;

            if amount >= pool_to_token.amount {
                return Err(PoolError::OverBuy.into());
            }

            let swap_price = Self::swap_price_define(amount,
pool_from_token, pool_to_token);
            let  commision_amount:  u64  =  swap_price  *
COMMISION_PERCENT / 1000;
            let                user_from_token                =
Account::unpack_from_slice(&user_from_token_info.data.borrow()
)?;

            if swap_price > user_from_token.amount {
                return Err(PoolError::TooMuchBuy.into());
            }

            let buy = spl_token::instruction::transfer(
                token_info.key,
                pool_to_token_info.key,
                user_to_token_info.key,
                admin_info.key,
                &[admin_info.key],
                amount,
            )?;
            let pay = spl_token::instruction::transfer(
                token_info.key,
                user_from_token_info.key,
                pool_from_token_info.key,
                user_info.key,
                &[user_info.key],
                swap_price,
            )?;
            let comm = spl_token::instruction::transfer(
                token_info.key,
                user_from_token_info.key,
                commision_info.key,
                user_info.key,
                &[user_info.key],
                commision_amount,
            )?;
            invoke(
                &buy,
                &[
                    pool_to_token_info.clone(),
                    user_to_token_info.clone(),
                    admin_info.clone(),
                    token_info.clone(),
                ],
            )?;
            invoke(
                &pay,
```

```
                &[
                    user_from_token_info.clone(),
                    pool_from_token_info.clone(),
                    user_info.clone(),
                    token_info.clone(),
                ],
            )?;
            invoke(
                &comm,
                &[
                    user_from_token_info.clone(),
                    commision_info.clone(),
                    user_info.clone(),
                    token_info.clone(),
                ],
            )?;

            Ok(())
        }

        pub fn swap_price_define(amount: u64, pool_from_token:
Account, pool_to_token: Account) -> u64 {
            ((amount as f64 / (pool_to_token.amount - amount)
as f64) * pool_from_token.amount as f64)
                as u64
        }

        pub  fn  withdraw_liquidity(accounts:  &[AccountInfo],
amount: u64) -> ProgramResult {
            msg!("Withdraw liquidity");

            let acc_iter = &mut accounts.iter();
            let user_info = next_account_info(acc_iter)?;
            let _ = next_account_info(acc_iter)?;
            let                x_user_token_info              =
next_account_info(acc_iter)?;
            let                y_user_token_info              =
next_account_info(acc_iter)?;
            let                 xy_lp_user_info               =
next_account_info(acc_iter)?;
            let                pool_x_token_info              =
next_account_info(acc_iter)?;
            let                pool_y_token_info              =
next_account_info(acc_iter)?;
            let               mint_lp_token_info              =
next_account_info(acc_iter)?;
            let _ = next_account_info(acc_iter)?;
            let _ = next_account_info(acc_iter)?;
            let _ = next_account_info(acc_iter)?;
            let admin_info = next_account_info(acc_iter)?;
            let token_info = next_account_info(acc_iter)?;
            let _ = next_account_info(acc_iter)?;
            let _ = next_account_info(acc_iter)?;

            if !user_info.is_signer {
```

```
            return Err(PoolError::SignedRequired.into());
        }

        let                    xy_lp_user                    =
Account::unpack_from_slice(&xy_lp_user_info.data.borrow())?.am
ount;

        if amount > xy_lp_user {
            return Err(PoolError::OverWithdraw.into());
        }

        let                 token_x_in_pool                  =
Account::unpack_from_slice(&pool_x_token_info.data.borrow())?.
amount;
        let                 token_y_in_pool                  =
Account::unpack_from_slice(&pool_y_token_info.data.borrow())?.
amount;
        let                    total_lp                      =
Mint::unpack_from_slice(&mint_lp_token_info.data.borrow())?.su
pply;
        let [x_amount, y_amount] =
            Self::liquidity_profit(amount,      total_lp,
token_x_in_pool, token_y_in_pool);

        Self::withdraw_fee(accounts)?;

        let user_withdraw = spl_token::instruction::burn(
            token_info.key,
            xy_lp_user_info.key,
            mint_lp_token_info.key,
            user_info.key,
            &[user_info.key],
            amount,
        )?;

        invoke(
            &user_withdraw,
            &[
                mint_lp_token_info.clone(),
                xy_lp_user_info.clone(),
                user_info.clone(),
                token_info.clone(),
            ],
        )?;

        let ix = spl_token::instruction::transfer(
            token_info.key,
            pool_x_token_info.key,
            x_user_token_info.key,
            admin_info.key,
            &[admin_info.key],
            x_amount,
        )?;
        invoke(
            &ix,
```

```
                &[
                    x_user_token_info.clone(),
                    pool_x_token_info.clone(),
                    admin_info.clone(),
                    token_info.clone(),
                ],
            )?;

            let iy = spl_token::instruction::transfer(
                token_info.key,
                pool_y_token_info.key,
                y_user_token_info.key,
                admin_info.key,
                &[admin_info.key],
                y_amount,
            )?;
            invoke(
                &iy,
                &[
                    y_user_token_info.clone(),
                    pool_y_token_info.clone(),
                    admin_info.clone(),
                    token_info.clone(),
                ],
            )?;

            Ok(())
        }

        pub fn liquidity_profit(
            amount: u64,
            total_lp: u64,
            token_x_in_pool: u64,
            token_y_in_pool: u64,
        ) -> [u64; 2] {
            if total_lp == 0 {
                return [0, 0];
            }
            let ratio = amount as f64 / total_lp as f64;
            [
                (token_x_in_pool as f64 * ratio) as u64,
                (token_y_in_pool as f64 * ratio) as u64,
            ]
        }

        pub  fn  withdraw_fee(accounts:  &[AccountInfo])  ->
ProgramResult {
            msg!("Withdraw commision");

            let acc_iter = &mut accounts.iter();
            let user_info = next_account_info(acc_iter)?;
            let withdraw_info = next_account_info(acc_iter)?;
            let            x_user_token_info              =
next_account_info(acc_iter)?;
```

```
            let              y_user_token_info              =
next_account_info(acc_iter)?;
            let               xy_lp_user_info               =
next_account_info(acc_iter)?;
            let _ = next_account_info(acc_iter)?;
            let _ = next_account_info(acc_iter)?;
            let             mint_lp_token_info              =
next_account_info(acc_iter)?;
            let      current_comission_x_tokem_info         =
next_account_info(acc_iter)?;
            let      current_comission_y_tokem_info         =
next_account_info(acc_iter)?;
            let            total_commision_info             =
next_account_info(acc_iter)?;
            let admin_info = next_account_info(acc_iter)?;
            let token_info = next_account_info(acc_iter)?;
            let rent_info = next_account_info(acc_iter)?;
            let             system_program_info             =
next_account_info(acc_iter)?;

            if !user_info.is_signer {
                return Err(PoolError::SignedRequired.into());
            }

            let (withdraw_pubkey, bump_seed) =

WithdrawedFee::get_withdraw_pubkey_with_bump(user_info.key);

            if withdraw_pubkey != *withdraw_info.key {
                return Err(PoolError::WrongWithdraw.into());
            }

            if withdraw_info.data_is_empty() {
                msg!("creating new withdraw");
                let withdraw = WithdrawedFee {
                    user_x_withdraw: 0,
                    user_y_withdraw: 0,
                };
                let space = withdraw.try_to_vec()?.len();
                let                  rent                   =
&Rent::from_account_info(rent_info)?;
                let lamports = rent.minimum_balance(space);
                let signer_seeds: &[&[_]] = &[
                    &user_info.key.to_bytes(),
                    POOL_SEED.as_bytes(),
                    &[bump_seed],
                ];
                invoke_signed(
                    &system_instruction::create_account(
                        user_info.key,
                        &withdraw_pubkey,
                        lamports,
                        space as u64,
                        &id(),
                    ),
```

```
                    &[
                        user_info.clone(),
                        withdraw_info.clone(),
                        system_program_info.clone(),
                    ],
                    &[signer_seeds],
                )?;
                let _ = withdraw.serialize(&mut &mut
withdraw_info.data.borrow_mut()[..]);
            }

            let (total_commision_pubkey, bump_seed) =
TotalCommision::get_total_pubkey_with_bump();

            if total_commision_pubkey !=
*total_commision_info.key {
                return Err(PoolError::WrongWithdraw.into());
            }

            if total_commision_info.data_is_empty() {
                msg!("creating total commision");
                let total = TotalCommision {
                    total_x_commision: 0,
                    total_y_commision: 0,
                };
                let space = total.try_to_vec()?.len();
                let rent =
&Rent::from_account_info(rent_info)?;
                let lamports = rent.minimum_balance(space);
                let signer_seeds: &[&[_]] =
&[&id().to_bytes(), POOL_SEED.as_bytes(), &[bump_seed]];
                invoke_signed(
                    &system_instruction::create_account(
                        user_info.key,
                        &total_commision_pubkey,
                        lamports,
                        space as u64,
                        &id(),
                    ),
                    &[
                        user_info.clone(),
                        total_commision_info.clone(),
                        system_program_info.clone(),
                    ],
                    &[signer_seeds],
                )?;
                let _ = total.serialize(&mut &mut
total_commision_info.data.borrow_mut()[..]);
            }

            let mut total_commision =

TotalCommision::try_from_slice(&total_commision_info.data.borr
ow())?;
```

```
            let token_x_commision =

Account::unpack_from_slice(&current_comission_x_tokem_info.dat
a.borrow())?.amount;
            let token_y_commision =

Account::unpack_from_slice(&current_comission_y_tokem_info.dat
a.borrow())?.amount;
            let                 total_lp                 =
Mint::unpack_from_slice(&mint_lp_token_info.data.borrow())?.su
pply;
            let                 user_lp                  =
Account::unpack_from_slice(&xy_lp_user_info.data.borrow())?.am
ount;

            let [x_amount, y_amount] = Self::liquidity_profit(
                user_lp,
                total_lp,
                token_x_commision                            +
total_commision.total_x_commision,
                token_y_commision                            +
total_commision.total_y_commision,
            );

            let           mut           withdraw             =
WithdrawedFee::try_from_slice(&withdraw_info.data.borrow())?;

            let [x_amount, y_amount] = [
                x_amount - withdraw.user_x_withdraw,
                y_amount - withdraw.user_y_withdraw,
            ];

            withdraw.user_x_withdraw += x_amount;
            withdraw.user_y_withdraw += y_amount;
            total_commision.total_x_commision += x_amount;
            total_commision.total_y_commision += y_amount;

            let    _    =    withdraw.serialize(&mut    &mut
withdraw_info.data.borrow_mut()[..]);
            let  _  =  total_commision.serialize(&mut  &mut
total_commision_info.data.borrow_mut()[..]);

            let ix = spl_token::instruction::transfer(
                token_info.key,
                current_comission_x_tokem_info.key,
                x_user_token_info.key,
                admin_info.key,
                &[admin_info.key],
                x_amount,
            )?;
            invoke(
                &ix,
                &[
                    x_user_token_info.clone(),
                    current_comission_x_tokem_info.clone(),
```

```
                admin_info.clone(),
                token_info.clone(),
            ],
        )?;

        let iy = spl_token::instruction::transfer(
            token_info.key,
            current_comission_y_tokem_info.key,
            y_user_token_info.key,
            admin_info.key,
            &[admin_info.key],
            y_amount,
        )?;
        invoke(
            &iy,
            &[
                y_user_token_info.clone(),
                current_comission_y_tokem_info.clone(),
                admin_info.clone(),
                token_info.clone(),
            ],
        )?;

        Ok(())
    }
}
```

Вміст файлу state.rs для смарт-контакту, який реалізовує liqiudity pool

```
use borsh::{BorshDeserialize, BorshSerialize};
use solana_program::pubkey::Pubkey;

use crate::{id, POOL_SEED};

#[derive(BorshSerialize, BorshDeserialize, Debug)]
pub struct WithdrawedFee {
    pub user_x_withdraw: u64,
    pub user_y_withdraw: u64,
}

impl WithdrawedFee {
    pub fn get_withdraw_pubkey_with_bump(user: &Pubkey) -
> (Pubkey, u8) {
        Pubkey::find_program_address(&[&user.to_bytes(),
POOL_SEED.as_bytes()], &id())
    }

    pub fn get_withdraw_pubkey(user: &Pubkey) -> Pubkey {
        let          (pubkey,          _)          =
Self::get_withdraw_pubkey_with_bump(user);
        pubkey
    }
}
```

```
#[derive(BorshSerialize, BorshDeserialize, Debug)]
pub struct TotalCommision {
    pub total_x_commision: u64,
    pub total_y_commision: u64,
}

impl TotalCommision {
    pub fn get_total_pubkey_with_bump() -> (Pubkey, u8) {
        Pubkey::find_program_address(&[&id().to_bytes(),
POOL_SEED.as_bytes()], &id())
    }

    pub fn get_total_pubkey() -> Pubkey {
        let (pubkey, _) =
Self::get_total_pubkey_with_bump();
        pubkey
    }
}
```

Вміст файлу functional.rs для смарт-контакту, який реалізовує liqiudity pool

```
#![cfg(feature = "test-bpf")]
use std::assert_eq;

use borsh::BorshDeserialize;
use solana_program::{program_pack::Pack, pubkey::Pubkey,
system_instruction};
use solana_program_test::{
    processor,
    tokio::{self},
    ProgramTest, ProgramTestContext,
};

use spl_token::state::{Account, Mint};

use pool::{entrypoint::process_instruction, id,
instruction::PoolInstruction};
use solana_sdk::signature::{Keypair, Signer};
use solana_sdk::transaction::Transaction;

struct Env {
    ctx: ProgramTestContext,
    admin: Keypair,
    user_01: Keypair,
    user_02: Keypair,
    mint_lp_account: Keypair,
    user_01_x_token_account: Keypair,
    user_01_y_token_account: Keypair,
    user_01_lp_token_account: Keypair,
    user_02_x_token_account: Keypair,
    user_02_y_token_account: Keypair,
    user_02_lp_token_account: Keypair,
    pool_x_token_account: Keypair,
```

```
        pool_y_token_account: Keypair,
        commision_x_token_account: Keypair,
        commision_y_token_account: Keypair,
    }

    impl Env {
        async fn new() -> Self {
            let program_test = ProgramTest::new("pool", id(),
processor!(process_instruction));
            let            mut            ctx            =
program_test.start_with_context().await;

            let admin = Keypair::new();
            let user_01 = Keypair::new();
            let user_02 = Keypair::new();

            ctx.banks_client

.process_transaction(Transaction::new_signed_with_payer(
                    &[
                        system_instruction::transfer(
                            &ctx.payer.pubkey(),
                            &admin.pubkey(),
                            1_000_000_000,
                        ),
                        system_instruction::transfer(
                            &ctx.payer.pubkey(),
                            &user_01.pubkey(),
                            1_000_000_000,
                        ),
                        system_instruction::transfer(
                            &ctx.payer.pubkey(),
                            &user_02.pubkey(),
                            1_000_000_000,
                        ),
                    ],
                    Some(&ctx.payer.pubkey()),
                    &[&ctx.payer],
                    ctx.last_blockhash,
                ))
                .await
                .unwrap();

            let mint_x_account = Keypair::new();
            let mint_y_account = Keypair::new();
            let mint_lp_account = Keypair::new();
            let     mint_array     =     [&mint_x_account,
&mint_y_account, &mint_lp_account];

            let token_program = &spl_token::id();
            let                    rent                    =
ctx.banks_client.get_rent().await.unwrap();
            let mint_rent = rent.minimum_balance(Mint::LEN);

            for i in mint_array {
```

```
                let         token_mint_account_ix          =
solana_program::system_instruction::create_account(
                    &ctx.payer.pubkey(),
                    &i.pubkey(),
                    mint_rent,
                    Mint::LEN as u64,
                    token_program,
                );

                let            token_mint_a_ix             =
spl_token::instruction::initialize_mint(
                    token_program,
                    &i.pubkey(),
                    &admin.pubkey(),
                    None,
                    9,
                )
                .unwrap();

                let            token_mint_a_tx             =
Transaction::new_signed_with_payer(
                    &[token_mint_account_ix,
token_mint_a_ix],
                    Some(&ctx.payer.pubkey()),
                    &[&ctx.payer, &i],
                    ctx.last_blockhash,
                );

                ctx.banks_client
                    .process_transaction(token_mint_a_tx)
                    .await
                    .unwrap();
            }

            let user_01_x_token_account = Keypair::new();
            let user_01_y_token_account = Keypair::new();
            let user_01_lp_token_account = Keypair::new();
            let user_02_x_token_account = Keypair::new();
            let user_02_y_token_account = Keypair::new();
            let user_02_lp_token_account = Keypair::new();
            let user_wallets = [
                [&user_01_x_token_account,   &mint_x_account,
&user_01],
                [&user_01_y_token_account,   &mint_y_account,
&user_01],
                [&user_01_lp_token_account, &mint_lp_account,
&user_01],
                [&user_02_x_token_account,   &mint_x_account,
&user_02],
                [&user_02_y_token_account,   &mint_y_account,
&user_02],
                [&user_02_lp_token_account, &mint_lp_account,
&user_02],
            ];
```

```
            let                account_rent                =
rent.minimum_balance(Account::LEN);

            for [i, j, k] in user_wallets {
                let      token_associated_account_ix      =
solana_program::system_instruction::create_account(
                    &ctx.payer.pubkey(),
                    &i.pubkey(),
                    account_rent,
                    Account::LEN as u64,
                    token_program,
                );

                let        initialize_account_a_ix        =
spl_token::instruction::initialize_account(
                    token_program,
                    &i.pubkey(),
                    &j.pubkey(),
                    &k.pubkey(),
                )
                .unwrap();

                let  create_new_associated_token_account_tx  =
Transaction::new_signed_with_payer(
                    &[token_associated_account_ix,
initialize_account_a_ix],
                    Some(&ctx.payer.pubkey()),
                    &[&ctx.payer, &i],
                    ctx.last_blockhash,
                );

                ctx.banks_client

.process_transaction(create_new_associated_token_account_tx)
                    .await
                    .unwrap();
            }

            let pool_x_token_account = Keypair::new();
            let pool_y_token_account = Keypair::new();
            let pool_wallets = [
                [&pool_x_token_account, &mint_x_account],
                [&pool_y_token_account, &mint_y_account],
            ];

            for [i, j] in pool_wallets {
                let pool_token_associated_account_ix =

solana_program::system_instruction::create_account(
                        &ctx.payer.pubkey(),
                        &i.pubkey(),
                        account_rent,
                        Account::LEN as u64,
                        token_program,
                    );
```

```
                let        initialize_account_a_ix        =
spl_token::instruction::initialize_account(
                    token_program,
                    &i.pubkey(),
                    &j.pubkey(),
                    &admin.pubkey(),
                )
                .unwrap();

                let
create_new_pool_associated_token_account_tx            =
Transaction::new_signed_with_payer(
                    &[pool_token_associated_account_ix,
initialize_account_a_ix],
                    Some(&ctx.payer.pubkey()),
                    &[&ctx.payer, &i],
                    ctx.last_blockhash,
                );

                ctx.banks_client

.process_transaction(create_new_pool_associated_token_account_
tx)
                    .await
                    .unwrap();
            }

            let commision_x_token_account = Keypair::new();
            let commision_y_token_account = Keypair::new();
            let commision_wallets = [
                [&commision_x_token_account,
&mint_x_account],
                [&commision_y_token_account,
&mint_y_account],
            ];

            for [i, j] in commision_wallets {
                let pool_token_associated_account_ix =

solana_program::system_instruction::create_account(
                        &ctx.payer.pubkey(),
                        &i.pubkey(),
                        account_rent,
                        Account::LEN as u64,
                        token_program,
                    );

                let        initialize_account_a_ix        =
spl_token::instruction::initialize_account(
                    token_program,
                    &i.pubkey(),
                    &j.pubkey(),
                    &admin.pubkey(),
                )
```

```
                .unwrap();

                let
create_new_pool_associated_token_account_tx                    =
Transaction::new_signed_with_payer(
                    &[pool_token_associated_account_ix,
initialize_account_a_ix],
                    Some(&ctx.payer.pubkey()),
                    &[&ctx.payer, &i],
                    ctx.last_blockhash,
                );

                ctx.banks_client

.process_transaction(create_new_pool_associated_token_account_
tx)
                    .await
                    .unwrap();
            }

            let need_to_mint = [
                [&mint_x_account, &user_01_x_token_account],
                [&mint_y_account, &user_01_y_token_account],
                [&mint_x_account, &user_02_x_token_account],
                [&mint_y_account, &user_02_y_token_account],
            ];

            for    [mint_account,    user_token_account]    in
need_to_mint {
                let              mint_user_token              =
Transaction::new_signed_with_payer(
                    &[spl_token::instruction::mint_to(
                        token_program,
                        &mint_account.pubkey(),
                        &user_token_account.pubkey(),
                        &admin.pubkey(),
                        &[&admin.pubkey()],
                        10000000,
                    )
                    .unwrap()],
                    Some(&ctx.payer.pubkey()),
                    &[&ctx.payer, &admin],
                    ctx.last_blockhash,
                );

                ctx.banks_client
                    .process_transaction(mint_user_token)
                    .await
                    .unwrap();
            }

            Env {
                ctx,
                admin,
                user_01,
```

```
                user_02,
                mint_lp_account,
                user_01_x_token_account,
                user_01_y_token_account,
                user_01_lp_token_account,
                user_02_x_token_account,
                user_02_y_token_account,
                user_02_lp_token_account,
                pool_x_token_account,
                pool_y_token_account,
                commision_x_token_account,
                commision_y_token_account,
            }
        }
    }

    // test of first user providing liquidity
    #[tokio::test]
    async fn provide_liquidity_first() {
        let mut env = Env::new().await;

        let tx = Transaction::new_signed_with_payer(
            &[PoolInstruction::provide_liquidity(
                &env.user_01.pubkey(),
                &env.admin.pubkey(),
                &env.user_01_x_token_account.pubkey(),
                &env.user_01_y_token_account.pubkey(),
                &env.user_01_lp_token_account.pubkey(),
                &env.pool_x_token_account.pubkey(),
                &env.pool_y_token_account.pubkey(),
                &env.mint_lp_account.pubkey(),
                &env.commision_x_token_account.pubkey(),
                &env.commision_y_token_account.pubkey(),
                5,
                15,
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01, &env.admin],
            env.ctx.last_blockhash,
        );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let acc = env
            .ctx
            .banks_client

.get_account(env.user_01_lp_token_account.pubkey())
            .await
            .unwrap()
            .unwrap();

        let                user_lp                =
Account::unpack_from_slice(&acc.data.as_slice()).unwrap();
```

```
        let acc = env
            .ctx
            .banks_client
            .get_account(env.mint_lp_account.pubkey())
            .await
            .unwrap()
            .unwrap();

        let                    total_lp                    =
Mint::unpack_from_slice(&acc.data.as_slice()).unwrap();

        assert_eq!(user_lp.amount, 8);
        assert_eq!(total_lp.supply, 8);
    }

    // withdraw part of user`s liquidity
    #[tokio::test]
    async fn part_liquidity_withdraw() {
        let mut env = Env::new().await;

        let tx = Transaction::new_signed_with_payer(
            &[PoolInstruction::provide_liquidity(
                &env.user_01.pubkey(),
                &env.admin.pubkey(),
                &env.user_01_x_token_account.pubkey(),
                &env.user_01_y_token_account.pubkey(),
                &env.user_01_lp_token_account.pubkey(),
                &env.pool_x_token_account.pubkey(),
                &env.pool_y_token_account.pubkey(),
                &env.mint_lp_account.pubkey(),
                &env.commision_x_token_account.pubkey(),
                &env.commision_y_token_account.pubkey(),
                5,
                15,
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01, &env.admin],
            env.ctx.last_blockhash,
        );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let acc = env
            .ctx
            .banks_client

.get_account(env.user_01_x_token_account.pubkey())
            .await
            .unwrap()
            .unwrap();

        let                 user_x_token_start                 =
Account::unpack_from_slice(&acc.data.as_slice()).unwrap();
```

```
        let tx = Transaction::new_signed_with_payer(
            &[PoolInstruction::withdraw_liquidity(
                &env.user_01.pubkey(),
                &env.admin.pubkey(),
                &env.user_01_x_token_account.pubkey(),
                &env.user_01_y_token_account.pubkey(),
                &env.user_01_lp_token_account.pubkey(),
                &env.pool_x_token_account.pubkey(),
                &env.pool_y_token_account.pubkey(),
                &env.mint_lp_account.pubkey(),
                &env.commision_x_token_account.pubkey(),
                &env.commision_y_token_account.pubkey(),
                5,
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01, &env.admin],
            env.ctx.last_blockhash,
        );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let acc = env
            .ctx
            .banks_client

.get_account(env.user_01_lp_token_account.pubkey())
            .await
            .unwrap()
            .unwrap();

        let                     user_lp                     =
Account::unpack_from_slice(&acc.data.as_slice()).unwrap();

        let acc = env
            .ctx
            .banks_client
            .get_account(env.mint_lp_account.pubkey())
            .await
            .unwrap()
            .unwrap();

        let                     total_lp                     =
Mint::unpack_from_slice(&acc.data.as_slice()).unwrap();

        let acc = env
            .ctx
            .banks_client

.get_account(env.user_01_x_token_account.pubkey())
            .await
            .unwrap()
            .unwrap();
```

```
        let                    user_x_token_new              =
Account::unpack_from_slice(&acc.data.as_slice()).unwrap();

        let withdraw_user_x_token = user_x_token_new.amount -
user_x_token_start.amount;

        assert_eq!(user_lp.amount, 3);
        assert_eq!(total_lp.supply, 3);
        assert_eq!(withdraw_user_x_token, 3);
    }

    // test of tokens swap
    #[tokio::test]
    async fn swap() {
        let mut env = Env::new().await;

        let tx = Transaction::new_signed_with_payer(
            &[PoolInstruction::provide_liquidity(
                &env.user_01.pubkey(),
                &env.admin.pubkey(),
                &env.user_01_x_token_account.pubkey(),
                &env.user_01_y_token_account.pubkey(),
                &env.user_01_lp_token_account.pubkey(),
                &env.pool_x_token_account.pubkey(),
                &env.pool_y_token_account.pubkey(),
                &env.mint_lp_account.pubkey(),
                &env.commision_x_token_account.pubkey(),
                &env.commision_y_token_account.pubkey(),
                5,
                15,
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01, &env.admin],
            env.ctx.last_blockhash,
        );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let acc = env
            .ctx
            .banks_client

.get_account(env.user_01_y_token_account.pubkey())
            .await
            .unwrap()
            .unwrap();

        let                      user_y_start                =
Account::unpack_from_slice(&acc.data.as_slice()).unwrap();

        let acc = env
            .ctx
            .banks_client
```

```
.get_account(env.user_01_x_token_account.pubkey())
            .await
            .unwrap()
            .unwrap();

        let                  user_x_start                  =
Account::unpack_from_slice(&acc.data.as_slice()).unwrap();

        let tx = Transaction::new_signed_with_payer(
            &[PoolInstruction::swap_tokens(
                &env.user_01.pubkey(),
                &env.admin.pubkey(),
                &env.user_01_x_token_account.pubkey(),
                &env.user_01_y_token_account.pubkey(),
                &env.pool_x_token_account.pubkey(),
                &env.pool_y_token_account.pubkey(),
                &env.commision_x_token_account.pubkey(),
                13,
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01, &env.admin],
            env.ctx.last_blockhash,
        );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let acc = env
            .ctx
            .banks_client

.get_account(env.user_01_y_token_account.pubkey())
            .await
            .unwrap()
            .unwrap();

        let                  user_y_new                  =
Account::unpack_from_slice(&acc.data.as_slice()).unwrap();

        let acc = env
            .ctx
            .banks_client

.get_account(env.user_01_x_token_account.pubkey())
            .await
            .unwrap()
            .unwrap();

        let                  user_x_new                  =
Account::unpack_from_slice(&acc.data.as_slice()).unwrap();

        let  swap_y:  i64  =  user_y_new.amount  as  i64  -
user_y_start.amount as i64;
```

```
        let  swap_x:  i64  =  user_x_new.amount  as  i64  -
user_x_start.amount as i64;

        assert_eq!(swap_y, 13);
        assert_eq!(swap_x, -32);
    }

    // user first time withdraw commision
    #[tokio::test]
    async fn withdraw_fee_first() {
        let mut env = Env::new().await;

        let tx = Transaction::new_signed_with_payer(
            &[PoolInstruction::provide_liquidity(
                &env.user_01.pubkey(),
                &env.admin.pubkey(),
                &env.user_01_x_token_account.pubkey(),
                &env.user_01_y_token_account.pubkey(),
                &env.user_01_lp_token_account.pubkey(),
                &env.pool_x_token_account.pubkey(),
                &env.pool_y_token_account.pubkey(),
                &env.mint_lp_account.pubkey(),
                &env.commision_x_token_account.pubkey(),
                &env.commision_y_token_account.pubkey(),
                500000,
                750000,
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01, &env.admin],
            env.ctx.last_blockhash,
        );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let tx = Transaction::new_signed_with_payer(
            &[PoolInstruction::swap_tokens(
                &env.user_01.pubkey(),
                &env.admin.pubkey(),
                &env.user_01_x_token_account.pubkey(),
                &env.user_01_y_token_account.pubkey(),
                &env.pool_x_token_account.pubkey(),
                &env.pool_y_token_account.pubkey(),
                &env.commision_x_token_account.pubkey(),
                250000,
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01, &env.admin],
            env.ctx.last_blockhash,
        );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let acc = env
```

```
                .ctx
                .banks_client

.get_account(env.user_01_x_token_account.pubkey())
                .await
                .unwrap()
                .unwrap();

            let                   user_x_start               =
Account::unpack_from_slice(&acc.data.as_slice())
                .unwrap()
                .amount;

            let tx = Transaction::new_signed_with_payer(
                &[PoolInstruction::withdraw_fee(
                    &env.user_01.pubkey(),
                    &env.admin.pubkey(),
                    &env.user_01_x_token_account.pubkey(),
                    &env.user_01_y_token_account.pubkey(),
                    &env.user_01_lp_token_account.pubkey(),
                    &env.pool_x_token_account.pubkey(),
                    &env.pool_y_token_account.pubkey(),
                    &env.mint_lp_account.pubkey(),
                    &env.commision_x_token_account.pubkey(),
                    &env.commision_y_token_account.pubkey(),
                )],
                Some(&env.user_01.pubkey()),
                &[&env.user_01, &env.admin],
                env.ctx.last_blockhash,
            );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

            let acc = env
                .ctx
                .banks_client

.get_account(env.user_01_x_token_account.pubkey())
                .await
                .unwrap()
                .unwrap();

            let                   user_x_new                 =
Account::unpack_from_slice(&acc.data.as_slice())
                .unwrap()
                .amount;

            let get_commision = user_x_new - user_x_start;

            assert_eq!(get_commision, 750);
        }

        // user withdraw commision second time, but new commision
don`t arrived yet
```

```
    #[tokio::test]
    async fn withdraw_fee_second_time() {
        let mut env = Env::new().await;

        let tx = Transaction::new_signed_with_payer(
            &[PoolInstruction::provide_liquidity(
                &env.user_01.pubkey(),
                &env.admin.pubkey(),
                &env.user_01_x_token_account.pubkey(),
                &env.user_01_y_token_account.pubkey(),
                &env.user_01_lp_token_account.pubkey(),
                &env.pool_x_token_account.pubkey(),
                &env.pool_y_token_account.pubkey(),
                &env.mint_lp_account.pubkey(),
                &env.commision_x_token_account.pubkey(),
                &env.commision_y_token_account.pubkey(),
                500000,
                750000,
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01, &env.admin],
            env.ctx.last_blockhash,
        );

env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let tx = Transaction::new_signed_with_payer(
            &[PoolInstruction::swap_tokens(
                &env.user_01.pubkey(),
                &env.admin.pubkey(),
                &env.user_01_x_token_account.pubkey(),
                &env.user_01_y_token_account.pubkey(),
                &env.pool_x_token_account.pubkey(),
                &env.pool_y_token_account.pubkey(),
                &env.commision_x_token_account.pubkey(),
                250000,
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01, &env.admin],
            env.ctx.last_blockhash,
        );

env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let tx = Transaction::new_signed_with_payer(
            &[PoolInstruction::withdraw_fee(
                &env.user_01.pubkey(),
                &env.admin.pubkey(),
                &env.user_01_x_token_account.pubkey(),
                &env.user_01_y_token_account.pubkey(),
                &env.user_01_lp_token_account.pubkey(),
                &env.pool_x_token_account.pubkey(),
                &env.pool_y_token_account.pubkey(),
```

```
                &env.mint_lp_account.pubkey(),
                &env.commision_x_token_account.pubkey(),
                &env.commision_y_token_account.pubkey(),
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01, &env.admin],
            env.ctx.last_blockhash,
        );

env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let acc = env
            .ctx
            .banks_client

.get_account(env.user_01_x_token_account.pubkey())
            .await
            .unwrap()
            .unwrap();

        let                  user_x_start                  =
Account::unpack_from_slice(&acc.data.as_slice())
            .unwrap()
            .amount;

        let tx = Transaction::new_signed_with_payer(
            &[PoolInstruction::withdraw_fee(
                &env.user_01.pubkey(),
                &env.admin.pubkey(),
                &env.user_01_x_token_account.pubkey(),
                &env.user_01_y_token_account.pubkey(),
                &env.user_01_lp_token_account.pubkey(),
                &env.pool_x_token_account.pubkey(),
                &env.pool_y_token_account.pubkey(),
                &env.mint_lp_account.pubkey(),
                &env.commision_x_token_account.pubkey(),
                &env.commision_y_token_account.pubkey(),
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01, &env.admin],
            env.ctx.last_blockhash,
        );

env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let acc = env
            .ctx
            .banks_client

.get_account(env.user_01_x_token_account.pubkey())
            .await
            .unwrap()
            .unwrap();
```

```
        let                    user_x_new                   =
Account::unpack_from_slice(&acc.data.as_slice())
            .unwrap()
            .amount;

        let get_commision = user_x_new - user_x_start;

        assert_eq!(get_commision, 0);
    }

    // test for user not immediately get new commision fees
after providing liquidity
    #[tokio::test]
    async fn provide_new_liquidity() {
        let mut env = Env::new().await;

        let tx = Transaction::new_signed_with_payer(
            &[PoolInstruction::provide_liquidity(
                &env.user_01.pubkey(),
                &env.admin.pubkey(),
                &env.user_01_x_token_account.pubkey(),
                &env.user_01_y_token_account.pubkey(),
                &env.user_01_lp_token_account.pubkey(),
                &env.pool_x_token_account.pubkey(),
                &env.pool_y_token_account.pubkey(),
                &env.mint_lp_account.pubkey(),
                &env.commision_x_token_account.pubkey(),
                &env.commision_y_token_account.pubkey(),
                50000,
                75000,
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01, &env.admin],
            env.ctx.last_blockhash,
        );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let tx = Transaction::new_signed_with_payer(
            &[PoolInstruction::swap_tokens(
                &env.user_01.pubkey(),
                &env.admin.pubkey(),
                &env.user_01_x_token_account.pubkey(),
                &env.user_01_y_token_account.pubkey(),
                &env.pool_x_token_account.pubkey(),
                &env.pool_y_token_account.pubkey(),
                &env.commision_x_token_account.pubkey(),
                25000,
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01, &env.admin],
            env.ctx.last_blockhash,
        );
```

```
env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let tx = Transaction::new_signed_with_payer(
            &[PoolInstruction::provide_liquidity(
                &env.user_02.pubkey(),
                &env.admin.pubkey(),
                &env.user_02_x_token_account.pubkey(),
                &env.user_02_y_token_account.pubkey(),
                &env.user_02_lp_token_account.pubkey(),
                &env.pool_x_token_account.pubkey(),
                &env.pool_y_token_account.pubkey(),
                &env.mint_lp_account.pubkey(),
                &env.commision_x_token_account.pubkey(),
                &env.commision_y_token_account.pubkey(),
                75000,
                50000,
            )],
            Some(&env.user_02.pubkey()),
            &[&env.user_02, &env.admin],
            env.ctx.last_blockhash,
        );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let tx = Transaction::new_signed_with_payer(
            &[PoolInstruction::withdraw_fee(
                &env.user_01.pubkey(),
                &env.admin.pubkey(),
                &env.user_01_x_token_account.pubkey(),
                &env.user_01_y_token_account.pubkey(),
                &env.user_01_lp_token_account.pubkey(),
                &env.pool_x_token_account.pubkey(),
                &env.pool_y_token_account.pubkey(),
                &env.mint_lp_account.pubkey(),
                &env.commision_x_token_account.pubkey(),
                &env.commision_y_token_account.pubkey(),
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01, &env.admin],
            env.ctx.last_blockhash,
        );


env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let tx = Transaction::new_signed_with_payer(
            &[PoolInstruction::provide_liquidity(
                &env.user_01.pubkey(),
                &env.admin.pubkey(),
                &env.user_01_x_token_account.pubkey(),
                &env.user_01_y_token_account.pubkey(),
                &env.user_01_lp_token_account.pubkey(),
```

```
                &env.pool_x_token_account.pubkey(),
                &env.pool_y_token_account.pubkey(),
                &env.mint_lp_account.pubkey(),
                &env.commision_x_token_account.pubkey(),
                &env.commision_y_token_account.pubkey(),
                75000,
                50000,
            )],
            Some(&env.admin.pubkey()),
            &[&env.user_01, &env.admin],
            env.ctx.last_blockhash,
        );

env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let acc = env
            .ctx
            .banks_client

.get_account(env.user_01_x_token_account.pubkey())
            .await
            .unwrap()
            .unwrap();

        let                  user_x_start                    =
Account::unpack_from_slice(&acc.data.as_slice())
            .unwrap()
            .amount;

        let tx = Transaction::new_signed_with_payer(
            &[PoolInstruction::withdraw_fee(
                &env.user_01.pubkey(),
                &env.admin.pubkey(),
                &env.user_01_x_token_account.pubkey(),
                &env.user_01_y_token_account.pubkey(),
                &env.user_01_lp_token_account.pubkey(),
                &env.pool_x_token_account.pubkey(),
                &env.pool_y_token_account.pubkey(),
                &env.mint_lp_account.pubkey(),
                &env.commision_x_token_account.pubkey(),
                &env.commision_y_token_account.pubkey(),
            )],
            Some(&env.user_01.pubkey()),
            &[&env.user_01, &env.admin],
            env.ctx.last_blockhash,
        );

env.ctx.banks_client.process_transaction(tx).await.unwrap();

        let acc = env
            .ctx
            .banks_client
```

```
.get_account(env.user_01_x_token_account.pubkey())
            .await
            .unwrap()
            .unwrap();

        let                    user_x_new                    =
Account::unpack_from_slice(&acc.data.as_slice())
            .unwrap()
            .amount;

        let get_commision = user_x_new - user_x_start;

        assert_eq!(get_commision, 0);
    }
```

Вміст файлу Cargo.toml s для смарт-контакту, який реалізовує liqiudity pool

```
[package]
name = "pool"
version = "0.1.0"
authors = ["RequescoS"]
edition = "2021"

[features]
no-entrypoint = []
test-bpf = []

[dependencies]
borsh = "0.9.3"
thiserror = "1.0.30"
solana-program = "1.9.9"
spl-token  =  {  version  =  "3.1.1",  features  =  ["no-
entrypoint"] }


[dev-dependencies]
solana-program-test = "1.9.9"
solana-sdk = "1.9.9"
spl-token  =  {  version  =  "3.1.1",  features  =  ["no-
entrypoint"] }
spl-associated-token-account = "1.1.1"


[lib]
crate-type = ["cdylib", "lib"]
```