Державний торговельно-економічний університет

Кафедра комп'ютерних наук та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА
на тему:

«Взаємодія з блокчейном Ethereum за допомогою мови програмування Python»

Студента 5 курсу, 2з групи, спеціальності 126 «Інформаційні системи та технології»
____
підпис студента
Хвостов Даніїл Павлович

Науковий керівник
кандидат фізико-математичних наук , доцент
____
підпис керівника
Філімонова Тетяна Олегівна

Гарант освітньої програми
PhD з інформаційних технологій, старший викладач
____
підпис керівника
Тищенко Ігор Анатолійович

Київ 2025

Державний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра комп’ютерних наук та інформаційних систем
Спеціальність 126 «Інформаційні системи та технології»
Освітня програма «Інформаційні системи та технології»

Затверджую
Зав. кафедри ____________Пурський О.І.
«5» березня 2024р.

Завдання
на кваліфікаційну роботу студенту

Хвостову Даніїлу Павловичу
(прізвище, ім’я, по батькові)

1. Тема кваліфікаційної роботи
«Взаємодія з блокчейном Ethereum за допомогою мови програмування Python»
Затверджена наказом ректора від *«01» березня 2024 р. № 730*

2. Строк здачі студентом закінченої роботи *11 лютого 2025 року*

3. Цільова установка та вихідні дані до роботи
Мета роботи: дослідити взаємодію з блокчейном Ethereum за допомогою мови програмування Python, досліджуючи можливості створення децентралізованих застосунків (DApps) та виконання смарт-контрактів.
Об’єкт дослідження: процес розробки системи взаємодії з блокчейном Ethereum за допомогою мови програмування Python.
Предмет дослідження: інструменти та бібліотеки, які дозволяють здійснювати взаємодію з блокчейном за допомогою Python, а також практичні аспекти використання цих інструментів для створення та управління смарт-контрактами, а також децентралізованими застосунками.

4. Перелік графічного матеріалу

____

5. Консультанти по роботі із зазначенням розділів, за якими здійснюється консультування:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Філімонова Т.О.	05.03.2024 р.	05.03.2024 р.
2	Філімонова Т.О.	05.03.2024 р.	05.03.2024 р.
3	Філімонова Т.О.	05.03.2024 р.	05.03.2024 р.

6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. ТЕОРЕТИЧНИЙ ОГЛЯД БЛОКЧЕЙНУ ETHEREUM ТА ЙОГО ФУНКЦІОНАЛУ

1.1. Опис основних елементів блокчейну та їх функціональності

1.2. Розгляд архітектури Ethereum та ролі віртуальної машини Ethereum (EVM)

РОЗДІЛ 2. РОЗРОБКА СМАРТ-КОНТРАКТУ ЗА ДОПОМОГОЮ МОВИ ПРОГРАМУВАННЯ SOLIDITY

2.1. Огляд мови програмування Solidity

2.2. Створення простого смарт-контракту

РОЗДІЛ 3. ВЗАЄМОДІЯ З БЛОКЧЕЙНОМ ETHEREUM ЗА ДОПОМОГОЮ PYTHON

3.1. Огляд інструментів Python для роботи з блокчейном Ethereum

3.2. Взаємодія з власним Solidity контрактом за допомогою Python

3.3.Взаємодія з defi протоколами за допомогою Python

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

7. Календарний план виконання роботи

№ Пор.	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	фактично
1	2	3	4
1	*Вибір теми кваліфікаційної роботи*	*10.02.2024*	*10.02.2024*
2	*Розробка та затвердження завдання на кваліфікаційну роботу*	*05.03.2024*	*05.03.2024*
3	*Вступ*	*03.04.2024*	*03.04.2024*
4	*РОЗДІЛ 1. Теоретичний огляд блокчейну ethereum та його функціоналу*	*12.06.2024*	*12.06.2024*
5	*РОЗДІЛ 2. Розробка смарт-контракту за допомогою мови програмування solidity*	*02.09.2024*	*02.09.2024*
6	*РОЗДІЛ 3. Взаємодія з блокчейном ethereum за допомогою python*	*01.11.2024*	*01.1.2024*
7	*Висновки*	*02.12.2024*	*02.12.2024*
8	*Здача кваліфікаційної роботи на кафедру науковому керівнику*	*31.01.2025*	*31.01.2025*
9	*Попередній захист кваліфікаційної роботи*	*04.02.2025*	*04.02.2025*
10	*Виправлення зауважень, зовнішнє рецензування кваліфікаційної роботи*	*07.02.2025*	*07.02.2025*
12	*Представлення готової зшитої кваліфікаційної роботи на кафедру*	*11.02.2025*	*11.02.2025*
13	*Публічний захист кваліфікаційної роботи*	*За розкладом роботи ЕК*	

Дата видачі завдання «05.03.2024 р.»

9. Керівник кваліфікаційної роботи Філімонова Т.О.
(прізвище, ініціали, підпис)

10. Гарант освітньої програми Тищенко І.А.
(прізвище, ініціали, підпис)

11. Завдання прийняв до виконання студент Хвостов Д.П.
(прізвище, ініціали, підпис)

12. Відгук керівника кваліфікаційної роботи

У кваліфікаційній роботі досліджено процес взаємодії з блокчейном Ethereum за допомогою мови програмування Python. Розглянуто процес розробки простого смарт-контракту за допомогою мови програмування Solidity. Визначено, як за допомогою Python можна взаємодіяти зі створеним смарт-контрактом та децентралізованими застосунками (DApps). Отримані результати можуть бути корисними для розробників , які прагнуть освоїти засоби та методи роботи з блокчейнами, такими як Ethereum та їх децентралізованими протоколами. Вважаю, що всі поставлені завдання виконані. Робота може бути допущена до захисту.

Керівник кваліфікаційної роботи Філімонова Т.О.
04.02.2025

13. Висновок про кваліфікаційну роботу

Кваліфікаційна робота студента Хвостов Д.П.
(прізвище, ініціали)
може бути допущена до захисту в екзаменаційній комісії.

Гарант освітньої програми Тищенко І.А.
(підпис, прізвище, ініціали)

Завідувач кафедри Пурський О.І.
(підпис, прізвище, ініціали)

«_____»_________________2025 р.

Анотація

У цій кваліфікаційній роботі досліджується процес взаємодії з блокчейном Ethereum за допомогою мови програмування Python. Проводиться детальний теоретичний огляд блокчейну Ethereum та його функціоналу. Розглядається процес розробки простого смарт-контракту за допомогою мови програмування Solidity. Досліджується , як за допомогою Python можна взаємодіяти зі створеним смарт-контрактом та децентралізованими застосунками (DApps). Отримані результати можуть бути корисними для розробників , які прагнуть освоїти засоби та методи роботи з блокчейнами, такими як Ethereum та їх децентралізованими протоколами.

Ключові слова: блокчейн, смарт-контракт, децентралізовані застосунки (DApps).

Anotation

This qualification work explores the process of interacting with the Ethereum blockchain using the Python programming language. A detailed theoretical overview of the Ethereum blockchain and its functionality is provided. The process of developing a simple smart contract using the Solidity programming language is considered. It is investigated how Python can be used to interact with the created smart contract and decentralised applications (DApps). The results obtained may be useful for developers seeking to master the tools and methods of working with blockchains such as Ethereum and their decentralised protocols.

Keywords: blockchain, smart contract, decentralised applications (DApps).

ЗМІСТ

ВСТУП……………………………………..…………………………...8

РОЗДІЛ 1. ТЕОРЕТИЧНИЙ ОГЛЯД БЛОКЧЕЙНУ ETHEREUM ТА ЙОГО ФУНКЦІОНАЛУ…..10

1.1. Опис основних елементів блокчейну та їх функціональності…...10

1.2 Розгляд архітектури Ethereum та ролі віртуальної машини Ethereum (EVM)………………...………………………………..……16

РОЗДІЛ 2. РОЗРОБКА СМАРТ-КОНТРАКТУ ЗА ДОПОМОГОЮ МОВИ ПРОГРАМУВАННЯ SOLIDITY……….……..................................21

2.1 Огляд мови програмування Solidity………......................................21

2.2 Створення простого смарт-контракту...24

РОЗДІЛ 3. ВЗАЄМОДІЯ З БЛОКЧЕЙНОМ ETHEREUM ЗА ДОПОМОГОЮ PYTHON…...34

3.1 Огляд інструментів Python для роботи з блокчейном Ethereum.....34

3.2 Взаємодія з власним Solidity контрактом за допомогою Python....42

3.3 Взаємодія з defi протоколами за допомогою Python………………46

ВИСНОВКИ………………………..…………………………………..…60

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ………...……………………..62

ДОДАТОК……..…………………………………..…………...………67

ВСТУП

У сучасному світі технології розвиваються з неймовірною швидкістю, і однією з найважливіших інновацій останніх років є блокчейн технології. Ці технології змінюють підхід до багатьох аспектів нашого життя, від фінансів до управління даними та навіть виборчих систем. Ethereum, як одна з провідних технологій блокчейн-платформ, дозволяє створювати децентралізовані застосунки та смарт-контракти, що робить її незамінним інструментом для розробників.

Однією з важливих особливостей є взаємодія з блокчейном за допомогою мов програмування , таких як Python. Вивчення можливостей Python у взаємодії з блокчейном Ethereum дає змогу розробникам створювати інноваційні рішення у фінансовому секторі, логістиці, управлінні даними та у багатьох інших сферах. Блокчейн технології мають потенціал привнести революційні рішення в різні галузі, надаючи їм прозорості, безпеки, та децентралізації. Тож **актуальність** теми, її мета та завдання зумовлені тим, що ключем для успіху у сфері розробки сучасних технологій та застосунків можуть стати знання про використання Python для взаємодії з блокчейнами.

Мета і завдання дослідження. Мета дослідження дослідити взаємодію з блокчейном Ethereum за допомогою мови програмування Python, досліджуючи можливості створення децентралізованих застосунків (DApps) та виконання смарт-контрактів. Для досягнення поставленої мети необхідно було вирішити наступні **завдання:**

- провести теоретичний огляд блокчейну Ethereum та його функцій;
- розробити простий смарт-контракт на мові програмування Solidity;
- вивчити інструменти та бібліотеки Python для взаємодії з блокчейном Ethereum, зокрема web3.py;
- реалізувати взаємодію зі створеним смарт-контрактом за допомогою Python;

- реалізувати можливості взаємодії з децентралізованими фінансовими протоколами за допомогою Python;

Об'єкт дослідження: процес розробки системи взаємодії з блокчейном Ethereum за допомогою мови програмування Python..

Предмет дослідження: інструменти та бібліотеки, які дозволяють здійснювати взаємодію з блокчейном за допомогою Python, а також практичні аспекти використання цих інструментів для створення та управління смарт-контрактами, та децентралізованими застосунками.

Методи дослідження: Теоретична база дослідження складається з наукових статей, технічної документації, статей з різних інтернет ресурсів та матеріалів від провідних фахівців в області блокчейн-технологій та програмування. Для практичного вирішення поставлених задач використовувалися такі методи:

- аналіз літератури, статей і технічної документації, що описують блокчейн Ethereum та мови програмування Python і Solidity;
- програмне моделювання для розробки та тестування контрактів на Solidity;
- алгоритмічне програмування для написання коду взаємодії Python з блокчейном Ethereum;

Практичне значення. Результатом роботи є детальні інструкції та приклади для розробників, які хочуть освоїти інструменти взаємодії з блокчейнами, такими як Ethereum. Дослідження допоможе розробникам зрозуміти, як створювати та взаємодіяти зі смарт-контрактами, децентралізованими фінансовими протоколами, а також дозволить створювати більш безпечні та ефективні децентралізовані застосунки.

Структура та обсяг кваліфікаційної роботи. Кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних джерел із 38 найменувань, додатків і містить 60 сторінок основного тексту, 37 рисунків і 1 таблицю.

РОЗДІЛ 1.
ТЕОРЕТИЧНИЙ ОГЛЯД БЛОКЧЕЙНУ ETHEREUM ТА ЙОГО ФУНКЦІОНАЛУ

1.1. Опис основних елементів блокчейну та їх функціональності

Сучасний світ важко уявити без технологій та способів здійснення платежів, упродовж останніх років ці два поняття поєднують у собі блокчейн технології. Раніше люди використовували бартер, на зміну якому прийшли дорогоцінні метали, після металів люди почали використовувати паперові гроші. Зараз все частіше люди надають перевагу банківським карткам та електронним коштам. Усі ці види розрахунків , хоча й стали основою світової економіки, упродовж свого існування мали багато недоліків, таких як незручність, висока вартість транзакцій, тривалість обробки платежів та потреба в довірі до централізованих органів, що і зазначили у дослідженні Alex та Don Tapscott [1]. Поява блокчейну змінила ситуацію. Завдяки блокчейну користувачі можуть здійснювати транзакції між собою, без потреби у посередниках, таких як банки. Такий спосіб забезпечує швидкість, зниження витрат та підвищення безпеки, які зумовлені майже миттєвими платежами, низькими комісіями та децентралізацією, що являють собою великий прорив у порівнянні з традиційними банківськими системами, про що детально розповідається у статті CoinDesk [2]. Блокчейн має відкритий реєстр транзакцій користувачів, який важко підробити, цим зумовлена прозорість та безпека. Ці аспекти роблять блокчейн ідеальним інструментом для створення децентралізованих фінансових систем, що підкреслюється у статті CoinMarketCap Academy [3]. Приклади успішних застосувань блокчейну вже можна побачити в різних галузях. Візьмемо за приклад сферу фінансових технологій, в якій деякі компанії використовують блокчейн для

швидких міжнародних платежів. Також важливо зазначити, що блокчейн-технології мають значний потенціал для покращення фінансових систем, зробивши їх доступнішими та ефективнішими. Вони можуть сприяти розвитку цифрової економіки, забезпечуючи нові можливості для бізнесу та інновацій, що теж зазначили у дослідженні Alex та Don Tapscott [1]. Не забуваємо про важливу роль блокчейну у забезпеченні фінансової інклюзії, надаючи можливість користуватися фінансовими послугами тим, хто не має банківських рахунків, або людям у віддалених регіонах, про що зазначено у статті на Brookings [4].

Тепер, коли ми зрозуміли, як блокчейн досяг успіху і які важливі функції він може виконувати, варто детальніше розглянути, що таке блокчейн Ethereum і в чому полягає його особливість. Почнемо з того, що Ethereum - це децентралізована платформа, заснована на блокчейн-технології, яка дозволяє розробникам створювати і використовувати смарт-контракти та децентралізовані застосунки. На відміну від біткоіна, мета якого зосереджена на наданні альтернативної форми валюти, Ethereum спрямований на створення інфраструктури для розробки децентралізованих застосунків [5].

В основі блокчейна Ethereum лежать: децентралізація, консенсусні алгоритми та смарт-контракти. Ці концепції є фундаментальними для розуміння того, як працює блокчейн. Розглянемо їх детальніше:

1. Децентралізація — взагалі вона означає відсутність центрального контролю або управління. Але у контексті блокчену це означає, що відомості записані у ньому зберігаються на багатьох вузлах, а не на одному центральному сервері [6]. Можна знову ж таки порівняти це з традиційними банківськими системами, де вся інформація зберігається у центральних банках, що робить їх вразливими до зломів та маніпуляцій. Перевагами децентралізації є висока стійкість до зломів, оскільки для зміни даних треба зламати всі вузли. Також

децентралізація забезпечує прозорість, яка перетікає в довіру, адже кожен учасник має доступ до тієї ж інформації, що і інші. І вочевидь відсутність централізованого контролю , що знижує ризик цензури та маніпуляцій. До недоліків можна віднести складність управління та узгодження змін і підвищену потребу у ресурсах для розподіленої мережі.

2. Консенсусні алгоритми — використовуються для забезпечення єдності та синхронізації між вузлами блокчейну [6]. Вони визначають, як нові блоки додаються до блокчейну. Найпопулярнішими є Proof of Work (PoW) та Proof of Stake (PoS). Різниця між ними полягає в тому, що PoW використовує обчислювальну потужність для вирішення складних математичних задач, в той час як PoS обирає валідаторів на основі кількості їх криптовалютних активів. Перевагами консенсусних алгоритмів є забезпечення цілісності та узгодженості даних у мережі, захист від подвійної витрати криптовалют, а також стимулювання учасників мережі до підтримки її безпеки та ефективності. Недоліками являються високе енергетичне споживання, якщо розглядати алгоритм PoW, тож виникають питання з приводу екології. У випадку PoS недоліком є централізація, якщо велика кількість криптовалюти знаходиться у невеликої кількості учасників.
3. Смарт-контракти — являють собою програмні контракти, які автоматично виконують умови угоди, в момент коли певні умови задовольняються [6]. Завдяки таким контрактам забезпечується безперервність та прозорість у торгівлі, а також автоматизація складних процесів. Наприклад, смарт-контракти можуть використовуватися для виплат зарплатні, нарахування премій, зберігання коштів та багато чого ще. Перевагами є автоматизація процесів, яка знижує потребу у посередниках, умови контракту

зазначені у блокчейні, що надає ще більше довіри та прозорості блокчейну. До недоліків можна віднести складність у розробці та тестуванні контрактів, із-за чого може з'явитися вразливість у коді та призвести до фінансових втрат.

Підсумовуючи можна зазначити, що Ethereum поєднує у собі інноваційні технології для забезпечення безпеки, прозорості та ефективності. Що в свою чергу дозволяє створювати децентралізовані додатки та рішення, які мають значний потенціал для зміни різних галузей.

Тепер розглянемо, що можна створити за допомогою цих технологій. Одними із найцікавіших напрямів є децентралізовані додатки (DApps), а саме напрям фінансових додатків, більш відомий як DeFi, що означає Decentralized Finance. Але спочатку зрозуміємо, що таке DApps. Почнемо з того, що це програми, які працюють у блокчейн-мережі, а не на одному сервері. Вони використовують смарт-контракти для автоматизації процесів, що робить їх прозорими, безпечними та незалежними від централізованого контролю, це гарно підкреслено на статті CoinDesk [7]. Також хочеться виділити, що DApps можуть використовуватися у найрізноманітніших галузях: від соціальних мереж і ігор, до фінансових сервісів та інструментів для голосування. Розглянемо роботу DApps, а саме роль смарт-контрактів на прикладах вище зазначених галузей. У сфері соціальних мереж, такі додатки можуть забезпечувати користувачам контроль над своїми даними і контентом, в іграх — створювати унікальні економіки ігрового світу, а у фінансових сервісах — сприяти децентралізованому управлінню фінансами.

Розглянемо детальніше DeFi, вони являються частиною DApps, яка займається фінансовими послугами без посередників. Це означає, що ви можете брати позики, вкладати гроші, торгувати активами та багато іншого без необхідності звертатися до банків або інших фінансових установ, про DeFi гарно розповіли Forbes у своїй статті [8]. DeFi дозволяє

зробити фінансові операції прозорими, швидкими та доступними для всіх, хто має інтернет, в незалежності від місця перебування. Однією з ключових особливостей DeFi є можливість створення «пулів ліквідності» (liquidity pools) — це смарт-контракти, які автоматично виконують операції з купівлі-продажу активів, забезпечуючи миттєву ліквідність для користувачів [9]. Наприклад, можна обмінювати криптовалюту безпосередньо з іншими користувачами через смарт-контракт, що працює на децентралізованій платформі. Порівнюючи DeFi з реальним життям, можна уявити, що це схоже на використання каси в магазині. Ви кладете гроші в касу і отримуєте квитанцію без необхідності звертатися до касира. Так само, DeFi дозволяє вам здійснювати фінансові операції без посередників.

Далі розглянемо не менш важливі пункти, пов'язані з блокчейном, ці концепції є ключовими для розуміння, як він працює, а саме: блоки, ланцюги блоків, вузли та майнінг. Почнемо по порядку:

- Блоки — це окремі так би мовити «сторінки» великої цифрової книги під назвою блокчейн. Кожен блок містить список транзакцій, які були підтверджені мережею протягом певного періоду часу [10]. Наприклад, транзакції можуть включати передачу криптовалют між користувачами або виконання смарт-контрактів. Кожен блок має заголовок, який містить інформацію про нього, включаючи хеш попереднього блоку, час створення, а також інші важливі дані. Ці блоки створюються через процес, який називається майнінг, і після створення кожен блок стає частиною безперервного ланцюга. Блоки забезпечують прозорість та безпеку транзакцій. Оскільки всі транзакції записують в блоки, їх важко змінити або підробити без згоди всіх учасників мережі.
- Ланцюги блоків — іншими словами це і є блокчейн, так звана послідовність блоків, зв'язаних між собою хешами. Коли новий

блок створюється і додається до блокчейну, він отримує унікальний хеш, який є результатом криптографічного хешування [11]. Цей хеш використовується в наступному блоці, створюючи зв'язок між блоками. Довіра в блокчейн системі базується на ланцюгах блоків, вони є журналом всіх транзакцій [12].

- Вузли — це окремі комп'ютери або сервери, які підключені до блокчейн-мережі. Вони зберігають копії всього болкчейну та беруть участь у перевірці і додаванні нових блоків [13]. Вузли виконують різні функції залежно від їх типу. Наприклад, повні вузли зберігають повну копію блокчейну та перевіряють усі транзакції, тоді як легкі вузли зберігають лише частину блокчейну і використовуються для швидкого доступу до мережі. Вузли забезпечують децентралізацію та безпеку у блокчейні, гарантуючи, що всі учасники мають доступ до однієї й тієї ж інформації і що дані захищені від маніпуляцій [13].
- Майнінг — являє собою процес вирішення складних математичних задач для створення нових блоків та додавання їх до блокчейну [14]. Для вирішення криптографічних задач майнер використовує потужності власного комп'ютеру або інших вичислювальних машин. Таким чином до блокчейну додається новий блок, а майнер отримує винагороду у вигляді криптовалюти.

Для більшого розуміння я пропоную уявити блокчейн, як величезну бухгалтерську книгу, де кожна сторінка (у нашому випадку блок) містить записи про фінансові операції. В такому випадку ланцюги блоків будуть послідовністю цих сторінок, з'єднаних між собою. Вузли являються бухгалтерами, які зберігають копії всієї книги і стежать за тим, щоб записи були правильними. А майнінг — це процес, за допомогою якого ці бухгалтери додають нові сторінки до книги, перевіряють правильність записів.

1.2. Розгляд архітектури Ethereum та ролі віртуальної машини Ethereum (EVM)

Ethereum — це не просто блокчейн, це платформа для розвитку та інновацій, яка надає розробникам інструменти для створення наступного покоління цифрових рішень. Структура Ethereum має ключові особливості, які виділяють його серед інших блокчейнів [15]. Ethereum відомий, як перший блокчейн, що почав підтримувати смарт-контракти, завдяки яким почали створюватися DApps, блокчейн почав обростати власною інфраструктурою та все більше розробників почали цікавитися світом криптовалют, а саме розробкою блокчей-технологій, з кожним роком привносячи щось нове [16]. На сьогоднішній день Ethereum постійно розвивається завдяки спільноті розробників, що забезпечують його актуальність, а також адаптивність до нових вимог і технологій [17]. Щодня розробники працюють над створенням широкого спектру застосунків, від фінансових інструментів до ігрових платформ.

Але все це було б неможливим без «серця» Ethereum під назвою Ethereum Virtual Machine (EVM) [18]. По суті EVM являється ядром платформи. Завдяки EVM виконуються всі контракти та DApps. EVM виконує такі функції:

- Виконання смарт-контрактів є однією з ключових функцій EVM, забезпечуючи їх автоматичне виконання, коли певні умови задовільняються.
- Універсальність також можна віднести до функцій, оскільки EVM підтримує різні мови програмування, такі як Solidity і Vyper, що дозволяє розробникам створювати смарт-контракти за допомогою зручної для них мови, про те як це працює добре розписав у своїй роботі Daniel Drescher [19].
- Децентралізованість, яка забезпечена обробкою транзакцій та виконанням контрактів на всіх вузлах.

- Безпека користувачів є немаловажною функцією, яка досягається завдяки криптографічним методам, запобігаючи шахрайству та зловживанням.
- Стабільність, на останок хочеться виділити саме її, EVM забезпечує стабільне і безперервне виконання смарт-контрактів, навіть якщо деякі вузли виходять з ладу.

Тож значення EVM переоцінити неможливо. EVM дозволяє розробникам створювати і розгортати нові DApps, розширюючи можливості використання Ethereum. Завдяки стандарту ERC-20 і іншим, EVM забезпечує взаємодію між різними додатками і токенами [20]. Також EVM дозволяє легко оновлювати та вдосконалювати смарт-контракти, що сприяє інноваціям і поліпшенню системи. Таким чином завдяки EVM, Ethereum має потужну екосистему розробників, інвесторів і користувачів, які спільно працюють над розвитком платформи.

Тепер поговоримо про одне з останніх оновлень мережі Ethereum, а саме Ethereum 2.0, також відомий як Eth2 або Serenity, є значним оновленням блокчейну Ethereum, яке має на меті покращити швидкість, ефективність та масштабність платформи без втрати безпеки та децентралізації [21]. Основі зміни включають перехід на механізм Proof of Stake, використання shard chains та створення нової блокчейн-мережі під назвою beacon chain [22]. Proof of Stake змінює традиційний Proof of Work, що дозволяє користувачам приймати участь у процесі валідації транзакцій шляхом стейкінгу, замість використання високопотужних комп'ютерів для виконання складних математичних задач [23]. Це зменшує енергоспоживання блокчейну настільки, що Ethereum очікує зменшення вуглецевого сліду на 99,95% [23]. Shard chains в свою чергу дозволяють розподілити навантаження на кілька незалежних мереж, що підвищує швидкість обробки транзакцій [22]. А beacon chain виступає як центральна мережа, яка координує роботу shard chains та забезпечує безпеку та

стабільність системи [22]. Підсумовуючи можна сказати, що Ethereum 2.0 є важливим кроком для Ethereum, оскільки він дозволяє платформі зростати та адаптуватися до нових викликів, забезпечуючи її майбутнє як ключової платформи для децентралізованих застосунків та фінансових інструментів [23].

Тепер розглянемо шардінг (shard chains) більш детально, як ми зрозуміли це техніка, яка використовується для підвищення масштабованості блокчейн-мережі, розподіляючи її навантаження на кілька менших мереж або ж «шардів». Кожен шард обробляє частину загальної кількості транзакцій, що дозволяє мережі обробляти більше транзакцій одночасно та підвищує її продуктивність. Тож щоб дозволити мережі зменшити навантаження на кожен окремий вузол, кожен шард повинен обробляти свої власні транзакції та мати свої власні вузли для перевірки цих транзакцій. Шардінг є необхідним для масштабування блокчейнів, таких як Ethereum, щоб вони могли підтримувати більшу кількість користувачів і транзакцій.

Важливо згадати про Ролапи (Rollups), так називають технології масштабування блокчейн-мереж, яка використовується для підвищення їхньої пропускної здатності та зниження витрат на транзакції [24]. Вони відносяться до рішень Layer 2, що означає , що вони працюють на рівні над основною блокчейн-мережею, чи коротше кажучи над Layer 1 рівнем [25]. Ролапи працюють, об’єднуючи кілька транзакцій на Layer 2 блокчейні та передаючи їх як одну транзакцію на основну блокчейн мережу. Це дозволяє зменшити навантаження на основну мережу та знижує витрати на газ (комісії за транзакції). Ролапи використовуються в різних галузях, таких як DeFi, ігри та інші застосунки, які потребують високої пропускної здатності та низьких витрат за транзакції [26].

Ми зрозуміли, що таке Ролапи та що вони працюють на L2, а от що таке L2 ми зараз з вами з’ясуємо. L2 блокчейни є рішеннями для

масштабування, які працюють поверх основного блокчейну (L1). Їх створювали для підвищення пропускної здатності та зниження витрат за транзакції, залишаючись при цьому тісно інтегрованими з основним блокчейном. L2 обробляють транзакції офф-чейн, тобто поза основною мережею, і потім передають результати цих транзакцій на L1 для підтвердження та збереження. Цим і зумовлені зниження навантаження на основний блокчейн та прискорити оброку транзакцій. Також у нашому дослідженні слід згадати той факт, що вміючи взаємодіяти з L1, такими як Ethereum, ви автоматично вмієте взаємодіяти з L2. Це тому, що L2 рішення тісно інтегровані з L1 блокчейнами і використовують ті ж самі принципи та стандарти, що і L1. Можна бути розробником, як у L1 так і у L2 мережі, кому що більше до душі.

Наостанок розглянемо ERC20 стандарт токенів. Це технічний стандарт, який використовується для створення та випуску токенів на блокчейні Ethereum. Назва ERC20 розшифровується як «Ethereum Request for Comments 20». Цей стандарт визначає набір правил, яких повинні дотримуватися усі токени, які випускаються на основі Ethereum, що забезпечує їхню сумісність та взаємодію. ERC20 визначає набір функцій, що дозволяє їм бути сумісними з будь-якими додатками, які підтримують цей стандарт [20]. Завдяки стандартизації токени можуть безперешкодно взаємодіяти один з одним, а також з різними гаманцями, біржами та іншими DApps. Також ERC20 включає шість основних функцій, які дозволяють передавати токени, перевіряти баланс користувачів, а також надавати можливість здійснювати інші операції з токенами. Завдяки своєму широкому використанню, ERC20 став основою для багатьох проєктів та ICO (Initial Coin Offering), що дозволило швидко та ефективно залучати фінансування для нових проєктів. Взагалі ERC20 був створений для забезпечення стандартизації, що дозволяє розробникам швидко створювати нові токени, а також забезпечує їх взаємодію з різними

додатками та інфраструктурою. Це значно спрощує процес створення нових проєктів на основі Ethereum та забезпечує їх стабільність та надійність.

РОЗДІЛ 2.
РОЗРОБКА СМАРТ-КОНТРАКТУ ЗА ДОПОМОГОЮ МОВИ ПРОГРАМУВАННЯ SOLIDITY

2.1. Огляд мови програмування Solidity

Для взаємодії з блокчейном Ethereum або будь-якими іншими блокчейнами створеними на його основі - нам важливо розуміти, як створювати власні смарт-контракти, чи хоча б вміти їх читати, а для цього нам потрібні базові знання з мови програмування Solidity. Тож спочатку розберемося, що взагалі являє собою Solidity та чим відрізняється від інших мов програмування, на прикладі того ж самого Python. Спочатку треба зазначити, що Solidity є статично типовою, об'єктно орієнтованою мовою програмування, спеціально розробленою для створення смарт-контрактів на платформі Ethereum [27]. Вона використовує синтаксис, схожий на JavaScript, C++ та Python, що робить її доступною для тих, хто вже знайомий з цими мовами [27]. Завдяки статичному типуванню типи даних визначаються на стадії компіляції, що допомагає уникнути багато базових помилок. Також Solidity має підтримку наслідування, що надає змогу смарт-контрактам наслідувати інші контракти, таким чином надаючи можливість створювати модульні та повторно використовувані структури. У порівнянні з Python я би виділив такі пункти:

1. Мета використання — Solidity використовується для створення смарт-контрактів на блокчейні Ethereum, тоді як Python – це універсальна, я би навіть сказав «мультифункціональна» мова, яка використовується для веб-розробки, аналізу даних, машинного навчання та багато чого іншого. [28].

2. Синтаксис — у Solidity синтаксис більш складний та схожий на JavaScript або C++, тоді як Python відомий своїм чистим та легким для читання синтаксисом [28].
3. Виконання коду — код Solidity виконується на EVM, що може бути повільнішим порівняно з виконанням коду Python [28].

Для кращого розуміння Solidity розберемося зі змінними та якими вони бувають, а їх у нас 3 види: State Variables, Local Variables та Global Variables (табл 2.1). Змінні у Solidity є фундаментальним елементом, який дозволяє зберігати і управляти даними в смарт-контрактах.

Таблиця 2.1. Види змінних у Solidity

Змінні	**State Variables**	**Local Variables**	**Global Variables**
Сутність	Зберігаються у блокчейні і постійно утримують значення у собі	Визначаються всередині функції і існують тільки під час виконання цієї функції	Надаються Solidity і містять інформацію про блокчейн та транзакції
Область оголошення	Поза функціями контракту	Всередині функції контракту	Автоматично доступні в будь-якій частині контракту
Приклад	uint256 public myVariable;	uint256 localVar = 5;	msg.sender (адреса відправника транзакції)

Розуміння різних типів змінних у Solidity є ключовим для розробки ефективних смарт-контрактів. Використання змінних (табл 2.1) у

правильний спосіб дозволяє створювати більш ефективні та надійні смарт-контракти.

Також важливо розповісти про ABI, Read та Write функції. Почнемо з першої, ABI або ж Application Binary Interface є стандартним способом взаємодії з контрактами в екосистемі Ethereum [29]. Він визначає, як даний контракт може бути викликаний зовні або іншими контрактами. ABI включає опис всіх функцій контракту, їх типи входів і виходів, що дозволяє інтерпретатору декодувати дані. Також ABI дозволяє інтерфейсам взаємодіяти з контрактами, що є головним для створення DApps [30]. Ще ABI являються частиною процесу компіляції, коли код Solidity перетворюється на формат для зчитування машиною та який може бути інтерпретований EVM. Наостанок слід зазначити, що ABI стандартизує спосіб виклику функцій контракту, а це в свою чергу робить його універсальним і зручним для різних реалізацій [30].

Далі поговоримо про Read та Write функції, які в Solidity використовуються для зчитування та зміни так званих State Variables у контракті [31]. Read функції дозволяють отримувати значення State Variable без витрат газу, тоді як Write функції вимагають виклику транзакції для зміни значення. Вони є основою для створення інтерфейсів, які дозволяють взаємодіяти з контрактами, що робить їх важливими для розробки смарт-контрактів. Кажучи простими словами ABI, Read та Write функції є незамінними для створення децентралізованих додатків та реалізації складної логіки.

Також у Solidity є так звані Події та Логи (Events and Logging), які відіграють важливу роль у відстеженні та відображенні важливих подій, що відбуваються в смарт-контрактах. Вони дозволяють зберігати інформацію про транзакції та стан контрактів, що є критично важливим для аудиту, відладки та моніторингу. Події в Solidity використовуються для відправлення повідомлень про важливі події, які відбуваються в

контракті [32]. Ці повідомлення записуються у блокчейн і можуть бути використані для взаємодії з фронтендом або іншими контрактами. Наприклад коли ERC-20 токени переводяться з одного адресу на інший, викликається подія «Transfer», яка записується у блокчейн. Логи ж в свою чергу використовуються в Solidity для запису інформації про виконання контракту [32]. Це допомагає розробникам відстежувати виконання функцій, зберігати значення змінних та виявляти помилки. Події та логи значно підвищують прозорість та відповідальність смарт-контрактів [32]. Вони дозволяють розробникам і користувачам відстежувати історію подій, що відбуваються в контракті, а також забезпечують відповідність дій. Це особливо важливо для DApps, де прозорість та відповідальність є ключовими аспектами.

2.2. Створення простого смарт-контракту

Ми вже зрозуміли наскільки важливими є смарт-контракти, що з себе представляє мова програмування Solidity та багато іншого. Тепер настав час розробити наш власний «Банківський» смарт-контракт. Створимо, як у всім нам знайомих банках функції депозиту/виводу, а також зробимо імітацію вкладу під відсоток. У нашому дослідженні будемо використовувати сайт remix.ethereum.org, як середовище для написання та компіляції коду. Розглянемо розробку смарт-контракту поетапно. Почнемо з вказання ліцензії, версії Solidity та імпорту бібліотек (рис 2.1).

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
```

Рис 2.1. Вказання ліцензії, версії Solidity та імпорт бібліотек

Ліцензія SPDX вказує на ліцензійні умови використання контракту. Ліцензія MIT дозволяє вільне використання, модифікацію та

розповсюдження коду з відповідним зазначенням авторства [33]. Версію ми будемо використовувати 0.8.0, через її стабільність та безпеку. Нові версії звісно привносять певні поліпшення, виправлення помилок та нові можливості, але нам вистачить версії 0.8.0 для реалізації основних функцій необхідних для створення нашого смарт-контракту. Оскільки в нашому дослідженні стабільність грає не останню роль, то зупинимося саме на цій версії. Далі йде імпорт бібліотеки IERC20, яка являється інтерфейсом для роботи з токенами ERC20. Вона включає всі стандартні функції, які допоможуть нам взаємодіяти з токенами. Використання OpenZeppelin забезпечить надійність та безпеку нашого контракту [34]. Далі нам потрібно оголосити контракт, назвемо його «Bank», вкажемо основні змінні та власника контракту (рис 2.2).

```
contract Bank {
    IERC20 public token;
    uint256 public stakingRewardRate;
    uint256 public constant ONE_YEAR = 365 days;

    address public owner;
```

Рис 2.2. Оголошення контракту та основних змінних

Змінні:

- token — Посилання на контракт токену ERC20, який ми будемо використовувати.
- StakingRewardRate – Відповідає за відсоткову ставку нашої імітації вкладу під процент.
- ONE_YEAR – Константа, що визначає кількість секунд в одному році для розрахунків.

Також ми оголошуємо адресу, яка володіє контрактом і має привілеї адміністратора, такі як зміна процентної ставки або для екстреного виводу коштів у випадку, якщо щось піде не так.

Далі прописуємо структури та мапінги для зберігання даних (рис 2.3).

```
struct DepositInfo {
    uint256 amount;
    uint256 depositTime;
}

struct StakeInfo {
    uint256 amount;
    uint256 stakeTime;
    uint256 rewardDebt;
}

mapping(address => DepositInfo) public deposits;
mapping(address => StakeInfo) public stakes;
```

Рис 2.3. Структури та мапінги для зберігання даних

Структури, які ми будемо використовувати:

- DepositInfo – буде зберігати суму депозиту та час, коли він був зроблений.
- StakeInfo – буде зберігати інформацію про наш «вклад під процент», таку, як сума вкладу, час коли гроші були внесені і накопичені нагороди.

Наступним чином ми оголосимо події (рис 2.4). Як ми вже з’ясували, події у Solidity дозволяють записувати дії, що відбуваються в контракті, створюють лог для зручності відстеження та дебагінгу.

```
event Deposit(address indexed user, uint256 amount);
event Withdraw(address indexed user, uint256 amount);
event Stake(address indexed user, uint256 amount);
event ClaimRewards(address indexed user, uint256 reward);
event WithdrawStake(address indexed user, uint256 amount);
```

Рис 2.4. Оголошення подій

Для нашого проекту нам вистачить 5 подій, Deposit, який дозволить відстежувати всі внески на контракт та ідентифікувати користувачів.

Withdraw, який допомагає відстежувати всі виведення коштів з контракту. Stake для відстеження всіх дій, пов'язаних з нашим «вкладом під процент». Також нам знадобиться ClaimRewards для відстеження процесу отримання нагород користувачами. І наостанок добавимо WithdrawStake, який дозволить відстежувати виводи з «вкладу під процент». Також нам потрібно створити конструктор (рис 2.5).

```
constructor(IERC20 _token, uint256 _stakingRewardRate) {
    token = _token;
    stakingRewardRate = _stakingRewardRate;
    owner = msg.sender;
}
```

Рис 2.5. Створення конструктору

Конструктор виконується один раз при розгортанні контракту. Сам конструктор ініціалізує змінні контракту, в нашому випадку це token, stakingRewardRate та owner. Це важливо для налаштування початкових умов роботи контракту.

Задля забезпечення більшої безпеки контракту нам знадобиться створити модифікатор onlyOwner (рис 2.6).

```
modifier onlyOwner() {
    require(msg.sender == owner, "Not owner");
    _;
}
```

Рис 2.6. Модифікатор onlyOwner

Ми будемо використовувати модифікатор для обмеження доступу до деяких важливих функцій, таким чином доступ буде лише у власника контракту, тобто нас. Отже ми забезпечимо додатковий рівень безпеки, запобігаючи несанкціонованим діям.

Настав час приступити до написання функцій. Почнемо з функції депозиту, в нашому випадку ця функція буде приймати основну криптовалюту мережі ETH (рис. 2.7).

```
function deposit() external payable {    infinite gas
    require(msg.value > 0, "Cannot deposit 0 ETH");

    deposits[msg.sender].amount += msg.value;
    deposits[msg.sender].depositTime = block.timestamp;

    emit Deposit(msg.sender, msg.value);
}
```

Рис 2.7. Функція deposit

Наша функція дозволяє користувачам вносити ETH на контракт для безпечного зберігання. Варто звернути увагу на додану нами умову, яка перевіряє, чи сума депозиту більша за нуль, щоб уникнути помилок в майбутньому. Також ця функція записує суму депозиту та час внесення в мапінг «deposits».

Якщо ми вже почали з функції депозиту на контракт, то створимо наступною функцію виводу нашої криптовалюти ETH (рис 2.8).

```
function withdraw() external {    infinite gas
    uint256 depositAmount = deposits[msg.sender].amount;
    require(depositAmount > 0, "Nothing to withdraw");

    deposits[msg.sender].amount = 0; // Reset deposit amount

    payable(msg.sender).transfer(depositAmount);
    emit Withdraw(msg.sender, depositAmount);
}
```

Рис 2.8. Функція withdraw

Наша функція виводу дозволить користувачам вивести свої депозити у вигляді криптовалюти ETH. Також функція перевіряє, чи є у користувача депозит, і скидає його після виводу, щоб одна людина не змогла багато

разів виводити кошти. Для надсилання коштів користувачу використовує «transfer».

Наступною функцією буде «вклад під процент» (рис 2.9). Звісно вона буде максимально спростована, оскільки, як ми всі знаємо для отримання людьми відсотків до їх вкладів, банкам потрібно надавати кредити людям,

```
function depositWithInterest(uint256 _amount) external {    infinite gas
    require(_amount > 0, "Cannot deposit 0 tokens");
    require(stakes[msg.sender].amount == 0, "User can have only 1 deposit at time");
    token.transferFrom(msg.sender, address(this), _amount);

    stakes[msg.sender].amount += _amount;
    stakes[msg.sender].stakeTime = block.timestamp;

    emit Stake(msg.sender, _amount);
}
```

але нам вистачить функціоналу імітувати цей процес.

Рис 2.9. Функція depositWithInterest

Користувачі зможуть вносити токени для стейкінгу, чи простіше кажучи вкладу під процент. Не забуваємо про вимоги щодо мінімальної суми та одного депозиту на користувача. Функція виконує переведення токенів на контракт та записує інформацію у раніше нами створений мапінг «stakes».

Створили ми функцію вкладу під процент, тепер було б непогано створити функцію, яка цей процент буде розраховувати.

```
function calculateRewards(uint256 _amount, uint256 _duration) public view returns (uint256) {
    return (_amount * stakingRewardRate * _duration) / ONE_YEAR / 100;
}
```

Рис 2.10. Функція calculateRewards

Наша функція винагород буде розрахована на основі суми та тривалості нашого вкладу (рис 2.10).

Маючи функцію депозиту, для вкладу під процент, та функію розрахунку винагород, нам не вистачає лише функції, яка б ці винагороди виводила до нас на гаманець. Створимо таку функцію та назвемо її claimRewards (рис 2.11).

```
function claimRewards() external {    infinite gas
    StakeInfo storage userStake = stakes[msg.sender];
    require(userStake.amount > 0, "No tokens deposited");

    uint256 stakingDuration = block.timestamp - userStake.stakeTime;
    uint256 rewards = calculateRewards(userStake.amount, stakingDuration);

    require(rewards > 0, "No rewards at this moment");

    userStake.stakeTime = block.timestamp; // Update stake time

    require(token.transfer(msg.sender, rewards), "Reward transfer failed");

    emit ClaimRewards(msg.sender, rewards);
}
```

Рис 2.11. Функція claimRewards

Почнемо з перевірки, чи має користувач взагалі вклад під процент, якщо так, то далі обчислюємо тривалість стейкінгу та відповідні винагороди за допомогою створеної нами функції calculateRewards. Далі робимо ще одну перевірку, на наявність нагороди для видачі. Змінюємо час стейкінгу відповідно до поточного часу. Після цих всіх подій нагороди переводяться на адресу користувача, а факт отримання нагороди записується у викликану подію ClaimRewards.

Функція вкладу під відсоток — є, функція виводу винагороди — є, а ось про функцію виводу коштів ми зараз поговоримо (рис 2.12).

```
function withdrawDepositWithInterest() external {    infinite gas
    StakeInfo storage userStake = stakes[msg.sender];
    require(userStake.amount > 0, "No tokens deposited");

    uint256 stakingDuration = block.timestamp - userStake.stakeTime;
    uint256 amountToWithdraw = userStake.amount + calculateRewards(userStake.amount, stakingDuration);

    userStake.amount = 0; // Reset stake amount
    userStake.stakeTime = 0; // Reset stake time

    token.transfer(msg.sender, amountToWithdraw);
    emit WithdrawStake(msg.sender, amountToWithdraw);
}
```

Рис 2.12. Функція withdrawDepositWithInterest

Спочатку ми знову починаємо з перевірки, чи вклав щось користувач, чи ні. Якщо все ж таки він щось вкладав, і пройшло достатньо часу для накопичення винагороди, то винагорода додається до вкладу. Далі

вклад, та час від початку стейкінгу обнуляються, а винагорода разом із вклад перераховуються на адресу користувача. Знову ж таки викликається подія ClaimRewards для запису факту отримання нагороди.

Отже, основні функції ми прописали, але я вважаю доцільним розробити ще декілька на випадок непередбачуваних обставин. Першою ми розробимо, та розглянемо функцію setStakingRewardRate, яка буде змінювати процентну ставку винагороди (рис 2.13).

```
function setStakingRewardRate(uint256 _newRate) external onlyOwner {
    stakingRewardRate = _newRate;
}
```

Рис 2.13. Функція setStakingRewardRate

По суті у цієї функції є єдиний та ключовий функціонал — це оновлення ставки нагороди для стейкінгу. Тут потрібно звернути увагу на раніше нами розроблений модифікатор onlyOwner, який робить функцію доступною лише для власника контракту, тобто нас.

До непередбачуваних обставин можна додати і те, що хтось не зможе вивести свої кошти, чи щось піде не так при виводі, тут нам, як власникам знадобиться функціонал виводу коштів на свій гаманець, пропишемо вивід

```
function rescueEther() external onlyOwner {    infinite gas
    (bool success,) = payable(msg.sender).call{value: address(this).balance}("");
    require(success, "Failed to send ether");
}
```

криптовалюти ETH (рис 2.14).

Рис 2.14. Функція rescueEther

Також обов’язково добавляємо модифікатор onlyOwner інакше будь хто зможе вивести ETH з нашого контракту.

Оскільки наш контракт передбачає внесок ERC20 токенів, то їх вивід ми теж пропишемо (рис 2.15).

```
function rescueERC20(address _token) external onlyOwner {    infinite gas
    uint256 balance = IERC20(_token).balanceOf(address(this));
    require(balance > 0, "Nothing to rescue");

    require(IERC20(_token).transfer(msg.sender, balance), "Token transfer failed");
}
```

Рис 2.15. Функція rescueERC20

Прописавши вивод ETH та ERC20 токенів я би ще добавив зміну власника контракту, на всяк випадок (рис 2.16).

```
function transferOwnership(address _newOwner) external onlyOwner {
    require(_newOwner != address(0), "New owner is the zero address");

    owner = _newOwner;
}
```

Рис 2.16. Функція transferOwnership

Спочатку функція перевіряє, чи не є новий власник нульовою адресою, після чого оновлює адресу власника контракту на нову адресу.

Наш смарт-контрак успішно написаний, тепер діло лишилося замалим, нам потрібно додати наш контракт до блокчейну, щоб кожен хто захоче зміг взаємодіяти з ним. Тож першим чином у remix нам потрібно відкрити вкладку «Solidity compiler» (рис 2.17).

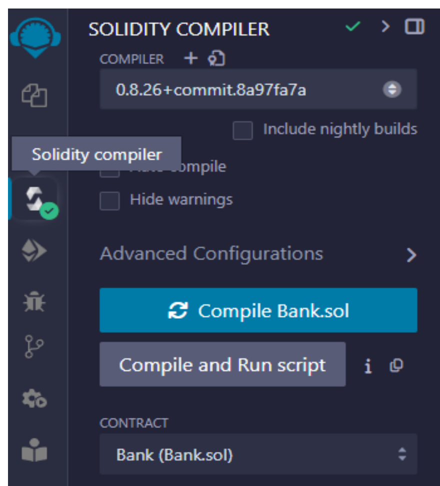


Рис 2.17. Solidity compiler

Після того, як натисли на «Solidity compiler» першим чином нам потрібно буде вибрати компілятор, я вибрав версію 0.8.26. Після вибору версії натискаємо на кнопку «Compile назва файлу.sol» і тут 2 варіанти: 1. У нас все добре, як на малюнку (рис 2.17); 2. У нас не все добре, а на місці зеленої галочки буде червона з надписом про помилку, яку потрібно буде усунути. Після підтвердження від компілятора, що все добре — переходимо на вкладку «Deploy & run transactions» (рис 2.18).

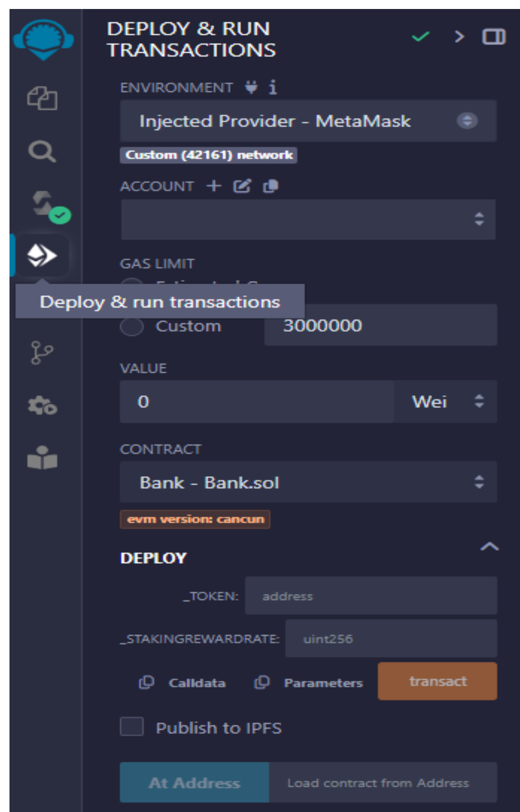
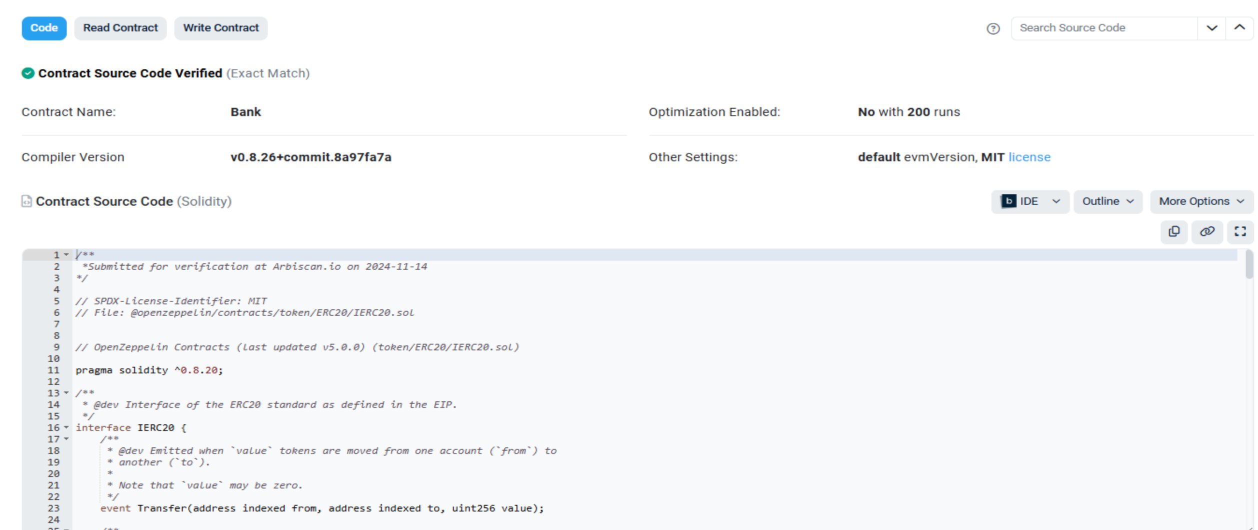


Рис 2.18. Deploy & run transaction

Спочатку в «ENVIRONMENT» нам потрібно вибрати «Injected Provider – MetaMask» та підключити наш гаманець до сайту . Контракт вибираємо наш, у моєму випадку це Bank.sol. Залишилося вказати параметри _TOKEN та _STAKINGREWARDRATE, які ми прописували ще на початку у конструкторі. Оскільки у нас імітація вкладу під процент, то у

дослідженні я скористаюся послугами сайту thirdweb.com для створення власного токену під назвою «thesis», контракт цього токену я і вказую у полі _TOKEN, будь хто може використати інші ERC20 токени за бажанням, а у полі _STAKINGREWARDRATE я вказую 500, що при діленні на 100 дасть нам 5% дохідності на рік. Наостанок натискаємо «transact» та підписуємо транзакцію, попередньо внісши кошти для плати за газ. Після підтвердження наш контракт додасться до блокчейну, а на сайті remix з'явиться функціонал для взаємодії та зміни контракту. Далі наш контракт відобразиться на сайті arbiscan. Наостанок бажано його верифікувати, щоб код з байтового перетворився в те, що ми писали та став «open source». Для цього на сторінці з контрактом нам потрібно вказати версію компілятора, ліцензію та сам код, якщо все добре, то ми побачимо «Contract Source Code Verified» (рис 2.19), а створення та



публікація контракту будуть завершені.

Рис 2.19. Верифікований контракт у блокчейні

РОЗДІЛ 3.
ВЗАЄМОДІЯ З БЛОКЧЕЙНОМ ETHEREUM ЗА ДОПОМОГОЮ PYTHON

3.1. Огляд інструментів Python для роботи з блокчейном Ethereum

Для взаємодії з блокчейном Ethereum, а вчасності зі смарт-контарктами нам знадобляться допоміжні інструменти, такі як наприклад бібліотека web3.py. Бібліотека Web3 є однією з найпопулярніших бібліотек для взаємодії з блокчейном Ethereum [35]. Завдяки їй ми можемо виконувати різні операції, такі як відправка транзакцій, взаємодія зі смарт-контрактами, відстеження подій та багато іншого [35]. Також крім Ethereum завдяки Web3.py ми можемо взаємодіяти зі всіма блокчейнами, що підтримують JSON-RPC API [35]. Основні можливості Web3.py:

- Запити (calls) — дозволяють отримувати дані з блокчейну без виконання транзакцій.
- Транзакції — дозволяє виконувати транзакції, такі як переведення ETH та багато інших.
- Контракти — головним для нас є підтримка взаємодії з смарт-контрактами, включаючи виклики методів контрактів та виклики констант.
- Підключення до ноди блокчейну — можливість підключення до різних Ethereum нод (Infura, Alchemy) для отримання доступу до блокчейну.

Web3.py є відкритим проектом, і багато розробників використовують його для створення децентралізованих додатків та сервісів. Він є майже незамінним інструментом для усіх, хто хоче використовувати блокчейн у своїх проектах за допомогою Python.

Не маловажливою є така річ, як JSON-RPC API, які я раніше вже згадував, по суті вони представляють собою стандартний набір методів, які використовуються для взаємодії з Ethereum [36]. API дозволяють програмам читати дані з блокчейну та надсилати дані до мережі [36]. Також з назви ми можемо зрозуміти, що для передачі даних використовується формат JSON.

Приклад основних методів JSON-RPC API:

- eth_getBalance – відповідає за отримання балансу адреси.
- eth_sendTransaction – надсилає транзакцію до мережі.
- eth_call – виконує запит до складного контракту без виконання транзакції.
- eth_getTransactionReceipt – відповідає за отримання підтвердження транзакцій.

Для більшого розуміння пропоную розібратися з базовими методами Web3 та оптимізувати їх для подальшого використання у нашому дослідженні. Створимо файл та назвемо його «utils.py», в якому будуть зберігатися основні методи, які найчастіше використовуються для роботи з такими блокчейнами, як Ethereum. Основні бібліотеки, які нам потрібно імпортувати будуть: asyncio, web3, а також деякі інші . Створимо ще один файл та назвемо його «config.py», у якому будуть зберігатися наші RPC для роботи (рис 3.1).

```
rpcs = {
    "eth": "https://rpc.ankr.com/eth",
    "arbi": "https://rpc.ankr.com/arbitrum",
}
```

Рис 3.1. RPC для роботи з блокчейном

Повернемося до нашого вайла «utils.py», спочатку створимо клас «Account» (рис 3.2). Далі ініціюємо Account, ми використовуємо приватний ключ від нашого гаманця з яким ми будемо працювати і для якого нам потрібно створити файл «.env» та створити поле «KEY =» та

вставити наш приватний ключ. Якщо його не вказати, то він не буде використовуватися.

```
class Account:
    def __init__(self, key=None, *, chain: Chain = Chains.eth, address_to: str = None):
        self._chain: Chain = chain
        self.account: LocalAccount = DefaultAccount.from_key(key) if key else DefaultAccount.create()
        self.address = self.account.address
        self.address_to = address_to
```

Рис 3.2. Ініціація класу Account

Chain ми будемо використовувати із спеціальної вибірки, в яких вказані данні про мережі. Створимо для них файл «models.py» та створимо клас «Chains», де пропишемо дані про мережі, які будемо використовувати (рис 3.3).

```
class Chains:
    eth = Chain(name='eth', token='ETH', scan='https://etherscan.io/tx/', chain_id=1)
    arbi = Chain(name='arbi', token='ETH', scan='https://arbiscan.io/tx/', chain_id=42161)
```

Рис 3.3. Клас з даними про блокчейн

Створимо у класі Account геттер «chain», який буде повертати поточний блокчейн, а також сеттер задля встановлення нового блокчейну, якщо він відрізняється від поточного (рис 3.4).

```
@property
def chain(self):
    return self._chain

@chain.setter
def chain(self, new_chain: Chain):
    if new_chain != self._chain:
        self._chain = new_chain
```

Рис 3.4. Геттер та сеттер chain

Також пропишемо саме підключення до блокчейну, яке буде повертати екземпляр AsyncWeb3 для поточного блокчейну, використовуючи HTTP провайдер, а також вкажемо таймаут у 30 секунд (рис 3.5). Модуль використовуємо eth для доступу до функцій Ethereum.

```
@property
def w3(self) -> AsyncWeb3:
    return AsyncWeb3(AsyncWeb3.AsyncHTTPProvider(rpcs[self.chain.name],
                                                 request_kwargs={'timeout': 30}), modules={'eth': (AsyncEth,)})
```

Рис 3.5. Підключення до блокчейну

Наступним етапом буде перевірка балансу гаманця у реальному часі Почнемо зі створення асинхронної функції під назвою «get_eth_ballance», як ми розуміємо з назви вона буде перевіряти кількість головної криптовалюти мережі ETH на гаманці (рис 3.6).

```
async def get_eth_balance(self) -> Amount:
    try:
        balance = await self.w3.eth.get_balance(self.address)
        return Amount(
            wei_amount=balance,
            amount=format_from_wei(balance,  decimals: 18)
        )
    except Exception as e:
        logger.error(f'{self.acc_info} - {e}')
        await asyncio.sleep(0.5)
        return await self.get_eth_balance()
```

Рис 3.6. Функція get_eth_ballance

У бібліотеці web3 вже є функція get_balance її ми і використаємо, але у неї є один жорсткий недолік, а саме баланс ми отримуємо у «wei» форматі, щоб отримати баланс у eth нам потрібно використати формулу: баланс отриманий у wei / 10 ** 18. У всіх токенів є спеціальний параметр «deciamls», ось у ETH decimals = 18, але не у всіх токенів 18 є стандартною цифрою, також же є decimals = 6, наприклад у стейбл коїнів USDT та USDC та інші різні значення decimals. Створимо допоміжну функцію, яка буде розраховувати суму в залежності від параметра decimals (рис 3.7).

```
def format_from_wei(amount, decimals):
    unit = None
    if decimals >= 18:
        unit = "ether"
    elif decimals >= 9:
        unit = "gwei"

    if unit:
        return float(from_wei(amount, unit))
    else:
        return amount / 10 ** decimals
```

Рис 3.7. Функція format_from_wei

Як завжди почнемо з назви, назвемо «format_from_wei», усе що нам потрібно це два аргументи, а саме amount та decimals, тобто значення у wei, яке потрібно перевести та кількість десяткових знаків формату decimals. Створимо змінну unit та встановимо початкове значення None. Далі нам потрібно прописати вибір одиниці вимірювання, якщо decimals більше або дорівнює 18, то одиниця вимірювання — ether він же ефір. А якщо більше/дорівнює 9, то gwei. Після чого зробимо перетворення в обрану одиницю. Якщо одиниця вимірювання unit визначена, то значення буде перетворене за допомогою функції from_wei з бібліотеки web3, у відповідну одиницю і поверне число типу float. У іншому ж випадку, коли одиниця вимірювання не визначена, то значення ділиться на 10 у ступені decimals і повертається.

Вернемося до функції get_eth_balance, створивши функцію для конвертації wei можемо прописувати, що наша функція буде повертати об'єкт класу Amount. Але спочатку цей клас потрібно створити, створювати будемо його у файлі , почнемо з декоратора «dataclass» в якому пропишемо slots=True та frozen=True, що забезпечить нам зменшення витрати пам'яті та зробить екземпляри класу незмінними. В самому класі пропишемо, що «wei_amount» може мати тип wei або int, та встановимо, що за замовчуванням значення є 0. В свою чергу поле «amount» має тип даних float і також за замовчуванням є 0. Тож повертати будемо wei_amount, який дорівнює змінній balance, а також значення amount, у якому ми використаємо нашу функцію format_fro_wei та вкажемо баланс у wei, а також decimals, ще раз нагадаю, що у ETH decimals = 18. Також добавимо logger, який при виникненні помилки використовує logger.error, включаючи інформацію про обліковий запис.

Потім функція робить пузу на 0.5 секунд та повторно викликає саму себе і заново пробує отримати інформацію про баланс.

Зробимо таку саму функцію, але тепер для будь якого токену. В цьому випадку наша функція буде приймати 2 додаткові аргументи, першим буде сам token, який є об'єктом типу Token та адресу типу str. Почнемо з першого, у випадку з перевіркою балансу ETH ми використовували вже відомі параметри, але у випадку з різними токенами нам потрібні різні данні. Для більшої зручності у взаємодії з токенами створимо у файлі models.py клас «Token» (рис 3.8).

```
@dataclass
class Token:
    name: str = 'NaN token'
    _address: HexStr | str = None
    decimals: int = 0

    @classmethod
    def get_token(cls, name: str = None, token_address: HexStr | str = None, decimals: int = 0):
        return Token(name=name, _address=token_address, decimals=decimals)

    @property
    def address(self) -> ChecksumAddress | None:
        if self._address is None:
            return None
        return to_checksum_address(self._address)

    @address.setter
    def address(self, value: HexStr | str):
        self._address = value
```

Рис 3.8. Клас Token

Розберемося з полями класу, а їх у нас 3:

1. name – Поле, яке зберігає назву токена, за замовченням встановлено NaN token.
2. _address – Приватне поле, яке зберігає у собі адресу токену у форматі HexStr або str. За замовчуванням виставлено значення None.
3. decimals – У полі зберігається кількість десяткових знаків для токена. Дефолтне значення дорівнює 0.

Створимо класовий метод під назвою «get _token», декоратор вказує на те, що метод є методом класу і приймає перший аргумент cls, який представляє сам клас. Також нам знадобляться усі поля, які я описав вище.

У цьому методі ми будемо створювати та повертати новий екземпляр класу Token з вказаними аргументами. Для зменшення ймовірності помилок під час введення під час введення адреси нам знадобиться додаткова функція для роботи з адресою. Створимо функцію, назвемо її «address» та вкажемо декоратор, який перетворить її на геттер. Метод буде перетворювати _address у формат checksum за допомогою функції to_checksum_address. Також зразу створимо сеттер у якому вкажемо значення _address = value.

Після того, як розібралися із класом Token вказуємо необов'язкову адресу типу str та йдемо далі. Наступним етапом буде отримання контракту токену за допомогою методу get_contract за його адресою перетвореною у формат checksum. Для цього використовуємо стандартний ABI [37]. По суті ABI є набором функцій контракту з уявленням про те, які типи даних вона приймає і які повертає. Грубо кажучи це потрібно, щоб можливо було працювати з функціями контракту, тому що web3 може кодувати при посиланні запиту та декодувати при отриманні результату функції ґрунтуючись на даних ABI. Наступним є перевірка форматування адреси. Якщо аргумент не надано, тоді використовується адреса облікового запису self.address. Інакше використовується надана адреса. В обох випадках адреса перетворюється у формат checksum. Для отримання будемо використовувати вже звичну конструкцію, яку ми створили ще у функції для отримання балансу ETH, але з деякими змінами. Замість get_balance будемо використовувати balanceof. І в кінці також пропишемо обробку виключень.

Ми вже багато чого з вами створили, тепер настав час перевірити на практиці. Створимо файл для тестів із назвою «test_w3.py». Пропишемо імпорти та створимо клас Test (рис 3.9).

```python
class Test:
    def __init__(self, account: Account):
        self.account = account

    async def test_w3(self):

        res = await self.account.w3.is_connected()
        print(f"connect status {res}")


    async def get_block(self):
        block = await self.account.w3.eth.get_block('latest')
        pprint(dict(block))

    async def test_balance(self):
        eth = await self.account.get_eth_balance()
        thesis_bal = await self.account.get_balance(THESIS)
        print(f"eth balance in {self.account.chain.name} : {eth.amount} eth")
        print(f"thesis balance in {self.account.chain.name} : {thesis_bal.amount} thesis")
```

Рис 3.9. Клас Test

На початку за допомогою __init__ ініціюємо клас Test, який приймає об'єкт Account і зберігає його у властивість self.account. Першою у нас на черзі буде перевірка підключення до блокчейну, яка використовує метод is_connected з об'єкту w3 для перевірки підключення до блокчейну та виводить результат. Наступним у нас буде метод отримання блоку з блокчейну за допомогою методу get_block, я також використав допоміжну бібліотеку «pprint» для виведення результату у приємному на вигляд форматі словника. Та на останок напишемо метод перевірки балансу за допомогою раніше створених методів. Будемо перевіряти баланс гаманця з якого деплоїли контракт та створили наш токен, ось його ми й перевіримо. Але спочатку створимо для нашого токену tesis об'єкт класу токен, без якого буде неможливо перевірити баланс нашого токену на гаманці (рис 3.10).

```python
THESIS = Token('thesis', '0xf6dB5A2DFd4bC84BDaeF5374C5F93c096c338C11', 18)
```

Рис 3.10. Об'єкт класу Token з інформацією про токен tesis

Як ми бачмо використовується назва, адреса та decimals токену.

Створимо асинхронну функцію test, яка буде викликати асинхронні методи test_w3 та test_balance з об'єкту тест. Метод get_block закоментований та не виконується, оскільки інформації про блок дуже багато і вона займе багато місця. Також допишемо основний блок виконання коду (рис 3.11).

```
if __name__ == '__main__':
    asyncio.set_event_loop_policy(asyncio.WindowsSelectorEventLoopPolicy())
    asyncio.run(test())
```

Рис 3.11. Основний блок виконання

Якщо файл виконується, як основний модуль, тоді запускається асинхронна функція, а ми отримуємо результат (рис 3.12).

```
connect status True
eth balance in arbi : 0.002218247555619181 eth
thesis balance in arbi : 53771.38269870379 thesis
```

Рис 3.12. Результати виконання асинхронних функцій test_w3 та test_balance

Підводячи підсумки можна сказати, що наш код відпрацював на всі 100% та був максимально оптимізований. Нам вдалося продемонструвати базові функції Web3.

3.2. Взаємодія з власним Solidity контрактом за допомогою Python

Для кращої демонстрації можливостей Python, як інструменту взаємодії з блокчейном Ethereum, розглянемо його взаємодію із нашим, створеним власноруч контрактом. Для початку роботи з любим контрактом нам знадобиться його ABI. Записуємо його у змінну «BANK_ABI», зберігається він у форматі JSON. Створимо також клас із завданнями, які розробимо трохи згодом (рис 3.13). Виберемо основні функції на яких сконцентруємо увагу, а саме: депозит ETH на контракт,

вклад токенів під процент, отримання нагород, виведення ETH та виведення токенів з відсотками.

```
class Tasks(Enum):
    dep_eth = 'dep'
    dep_token_w_interest = "dep_w"
    reward = 'rew'
    withdraw_eth = 'with'
    withdraw_token_w_interest = "with_w"
```

Рис 3.13. Клас Tasks

Наступним чином нам знадобиться ще один клас, в якому ми створимо основні функції для взаємодії з контрактом (рис 3.14). Назвемо цей клас «Bank», а також зробимо ініціалізацію у якій створимо об’єкт для збереження облікового запису, а також наш контракт, отриманий за допомогою методу get_contract.

```
class Bank:
    def __init__(self, account: Account):
        self.account = account
        self.contract = self.account.get_contract( address: '0x03bcbfb6266c771f004d64765e01a7b50066ee26', BANK_ABI)
```

Рис 3.14. Ініціація класу Bank

Першими напишемо функції для зчитування даних з контракту (рис 3.15). Напишемо функції, які отримують інформацію про депозит, депозит під відсоток, а також вони повертають час депозиту у вигляді timestamp, але це нам не потрібно.

```
@retry_on_error()
async def get_deposit_by_address(self, address: str):
    data = await self.contract.functions.deposits(to_checksum_address(address)).call()
    return data[0]

@retry_on_error()
async def get_stakes_by_address(self, address: str):
    data = await self.contract.functions.deposits(to_checksum_address(address)).call()
    return data[0]
```

Рис 3.15. Функції зчитування інформації про депозити

Напишемо методи створення транзакцій для різних дій, таких як: депозит ETH, депозит токенів з відсотками, отримання, виведення ETH та виведення токенів з відсотками (рис 3.16). Основою для кодування

виклику функції смарт-контракту у форматі ABI є метод encode_abi(). Як я вже казав ABI є стандартом для взаємодії зі смарт-контрактами в Ethereum. Таким чином визначається, як функції і події смарт-контракту мають бути закодовані та декодовані.

```
def deposit_eth(self) -> HexStr:
    return self.contract.encode_abi('deposit',
                                    )

def deposit_token_with_interest(self, amount: int) -> HexStr:
    return self.contract.encode_abi('depositWithInterest',
                                    args=[
                                        amount,
                                    ])

def claim_reward_token_with_interest(self) -> HexStr:
    return self.contract.encode_abi('claimRewards',
                                    )

def withdraw_eth(self) -> HexStr:
    return self.contract.encode_abi('withdraw',
                                    )

def withdraw_eth_token_with_interest(self) -> HexStr:
    return self.contract.encode_abi('withdrawDepositWithInterest',
                                    )
```

Рис 3.16. Функції запису

Перейдемо до створення функції з нашими так званими «тасками» (рис 3.17). Створимо вибір завдання. Спочатку перевіряємо значення змінної, робимо кодування виклику функції, визначаємо значення транзакції у wei, далі йде визначення, чи потрібне підтвердження транзакції, і на останок встановлюємо текстовий опис дій.

```python
async def task(self, amount_eth: Optional[Amount] = None, amount_token: Optional[Amount] = None, task: Tasks = Tasks.dep_eth):

    if task.value == Tasks.dep_eth.value:
        data = self.deposit_eth()
        value = amount_eth.wei_amount
        need_approve = False
        text = 'deposited eth'
```

Рис 3.17. Приклад одного із завдань функції task

Наступним чином, ми пропишемо тестовий депозит та зняття токенів ETH та THESIS з контракту. Спочатку, створимо змінні, у яких вкажемо кількість токенів, а також перетворимо їх у wei формат. У випаду ETH, ми будемо використовувати 0.0001 токен, а для THESIS вкажемо 1000. Насправді, нам не важлива кількість, тож можна вказати будь-які значення, звісно, якщо токенів вистачить.

Перейдемо до створення функції депозиту (рис 3.18). Назвемо її «test_deposits». Створимо обліковий запис «a» та банк «b» за допомогою цього облікового запису. Спочатку внесемо на контракт токен ETH, почекаємо 5 секунд, та зразу внесемо наш токен THESIS під відсоток. Наостанок виведемо результати депозитів.

```python
async def test_deposits():
    a = Account(key=key, chain=Chains.arbi)
    b = Bank(a)

    await b.task(amount_eth=eth_deposit_to_contract, task=Tasks.dep_eth)
    await asyncio.sleep(5)  # little wait
    await b.task(amount_token=THESIS_deposit_to_contract, task=Tasks.dep_token_w_interest)

    logger.info(f"get_deposit_by_address result: {format_from_wei(await b.get_deposit_by_address(a.address),  decimals: 18)}")
    logger.info(f"get_stakes_by_address result: {format_from_wei(await b.get_deposit_by_address(a.address),  decimals: 18)}")
```

Рис 3.18. Функція test_deposits

Функція виводу (рис 3.19) створюється майже таким же чином, за виключенням тасків, спочатку ми отримуємо винагороду за вклад під процент, потім виводимо ETH і наостанок знімаємо THESIS. Та не забуваємо про відображення результатів.

```python
async def test_withdrawals():
    a = Account(key=key, chain=Chains.arbi)
    b = Bank(a)

    await b.task(task=Tasks.reward)
    await asyncio.sleep(5)  # little wait
    await b.task(task=Tasks.withdraw_eth)
    await asyncio.sleep(5)  # little wait
    await b.task(task=Tasks.withdraw_token_w_interest)

    logger.info(f"get_deposit_by_address result: {format_from_wei(await b.get_deposit_by_address(a.address), 18)}")
    logger.info(f"get_stakes_by_address result: {format_from_wei(await b.get_deposit_by_address(a.address), 18)}")
```

Рис 3.19. Функція test_withdrawals

Усю основну частину написали, тепер настав час перевірити наш код, створимо функцію для запуску тестових функцій, а також встановимо політику подій для Windows і виведемо результат (рис. 3.20).

```
ck_status_tx:72 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - awaiting transaction confirmation
task:113 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - succesfully deposited eth https://arbiscan.io/tx/0xdfa0b80c7c7905d447bbf37f2408d7f41e6afd2b9edc6c9e5958cb755753e83
rove_token:274 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - already approved
ck_status_tx:72 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - awaiting transaction confirmation
task:113 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - succesfully deposited THESIS https://arbiscan.io/tx/0x82025091d0546c49a4c39b242e2ddbe1d59731391c81a508699d8009794d
test_deposits:130 - get_deposit_by_address result: 0.0001
test_deposits:131 - get_stakes_by_address result: 0.0001
ck_status_tx:72 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - awaiting transaction confirmation
task:113 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - succesfully claimed rewards in THESIS https://arbiscan.io/tx/0x3940b02af70bc5d5e4ae6fb759cfd2e976af35fbf2171c81dd3
ck_status_tx:72 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - awaiting transaction confirmation
task:113 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - succesfully withdrawed eth https://arbiscan.io/tx/0xa2b2379d7d495974da0cc1f70c9ebf1000564e2088f3e546e5ebe1a5f90d23
ck_status_tx:72 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - awaiting transaction confirmation
task:113 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - succesfully withdrawed THESIS https://arbiscan.io/tx/0x42c83a30d551d17d292e275a8a3a2185de9e0bfb8d44146b83dbd948674
test_withdrawals:143 - get_deposit_by_address result: 0.0
test_withdrawals:144 - get_stakes_by_address result: 0.0
```

Рис 3.20. Результат взаємодії з нашим контрактом

Можна побачити, що код виконав крок за кроком, та спокійно завершив свою роботу.

3.3. Взаємодія з defi протоколами за допомогою Python

Можливості Python не закінчуються на наших контрактах, адже ми можемо використовувати його для взаємодії із чужими контрактами. Розглянемо це на прикладі взаємодії Python та Uniswap. Спочатку розберемося, що взагалі таке Uniswap. Uniswap є децентралізованою платформою обміну криптовалют, яка дозволяє користувачам купувати та продавати криптовалюти без посередників [38]. Завдяки взаємодії Python з Uniswap, ми можемо створювати децентралізовані програми (dApps) та автоматизовувати торгові стратегії.

Спочатку ми повинні прописати ABI (application binary interface), контракти та код ініціалізації. Зазвичай ця вся інформація є у sdk (software development kit) на github розробників, у нашому випадку розробників Uniswap. Важливо зазначити, що контракти Uniswap у всіх мережах мають однаковий функціонал, змінюється тільки адреса контракту. Візьмемо з sdk на github контракти «FACTORY» та «ROUTER», також нам знадобляться ABI для «PAIR», «FACTORY» та «ROUTER», і наостанок додамо код ініціалізації. Для більшого розуміння, поговоримо про структуру самого Uniswap, яка представлена у вигляді двох головних контрактів, а саме контракту «ROUTER» та «FACTORY», контракт «FACTORY» створює новий контракт пари, а пара тримає у собі два різних токени, наприклад нативний токен ETH та будь який інший. А контракт «ROUTER» є сполучною ланкою, він при спробі обміну шукає пару токену, який ми хочемо обміняти, та через цю пару симулює та робить обмін.

Прописавши ABI та контракти, ми переходимо до створення класу «UniswapFactory», у якому розглянемо його основні функції. Хоч ми їх не будемо використовувати в даному дослідженні, але розглянути їх варто, як мінімум для більшого розуміння теми (рис 3.21). Першою зробимо функцію, завдяки якій ми отримаємо загальну кількість задеплоїних пар. Кожна пара з двох токенів може існувати лише в одному екземплярі. Також напишемо отримання пар за порядковим номером, які будуть представлені у вигляді словника, в якому є порядковий номер та контракт пари. Наостанок, напишемо найчастіше виконувану функцію, завдяки якій ми можемо отримати контракт пари, маючи лише дві адреси, і якщо такої

```
class UniswapFactory:
    def __init__(self, account: Account):
        self.account = account

    @property
    def contract(self):
        return self.account.get_contract(
            V2_UNISWAP_FACTORY_ADDRESSES[self.account.chain.name], UNISWAP_V2_FACTORY_ABI
        )

    @retry_on_error()
    async def all_pairs_length(self):
        return await self.contract.functions.allPairsLength().call()

    @retry_on_error()
    async def get_pool_by_key(self, key: int):
        return await self.contract.functions.allPairs(key).call()

    @retry_on_error()
    async def get_pair(self, address1: ChecksumAddress, address2: ChecksumAddress):
        return await self.contract.functions.getPair(address1, address2).call()
```

пари не існує, то ми отримаємо у відповідь нульову адресу.

Рис 3.21. Клас UniswapFactory та його функціонал

Тепер створимо клас «UniswapPair». Конструктор класу повинен приймати такі об’єкти, як account, а також t0 та t1, які мають у собі інформацію про токен. Першою створимо функцію, яка вичислює адресу пула (рис 3.22). Спочатку ми сортуємо наші адреси токенів, для цього нам підійде звичайна функція пітону sorted, потім використовуємо кодування keccak, яке використовується у контракті, а також нам знадобиться

контракт factory та так званий v2 init code hash, і тим самим детерміновано, без функції get_pair з factory, виконуємо її у себе у проекті.

```
@property
def pool_address(self):

    token_addresses = sorted([address.lower() for address in [self.t0.address, self.t1.address]])

    salt = keccak(
        packed.encode_packed(
            ["address", "address"],
            [*token_addresses],
        )
    )
    return to_checksum_address(
        keccak(
            HexBytes(0xFF)
            + HexBytes(V2_UNISWAP_FACTORY_ADDRESSES[self.account.chain.name])
            + salt
            + HexBytes(V2_UNISWAP_INIT_CODE_HASH)
        )[-20:]
    )
```

Рис 3.22. Функція pool_address

Наступним чином, розглянемо такі функції, як отримання контракту пари, отримання резервів, отримання токенів, а також можемо підрахувати вихідну кількість токенів (рис 3.23). Під час отримання контракту ми використаємо раніше нами написану функцію pool_address, а також ABI контракту пари. Завдяки функції отримання резервів, ми зможемо отримувати баланс токену 1 та токену 0. У функціях t0 та t1, ми отримуємо адреси токенів у парі. Ну і наостанок, буде функція «calculate_amount_out», яка розраховує вихідну кількість токенів та повертає amount_out, з урахуванням amount_in, резервів reserve_in, reserve_out і комісійних.

```python
@property
def contract(self):
    return self.account.get_contract(
        self.pool_address, UNISWAP_V2_PAIR_ABI
    )

@retry_on_error()
async def get_reserves(self) -> Tuple[int, int, int]:

    return await self.contract.functions.getReserves().call()

@retry_on_error()
async def t0(self):
    return await self.contract.functions.token0().call()

@retry_on_error()
async def t1(self):
    return await self.contract.functions.token1().call()

@staticmethod
def calculate_amount_out(amount_in: int, reserve_in: int, reserve_out: int):
    amount_in_with_fee = amount_in * 9975
    numerator = amount_in_with_fee * reserve_out
    denominator = reserve_in * 10000 + amount_in_with_fee
    amount_out = numerator // denominator
    return amount_out
```

Рис 3.23. Функції contract, get_reserves, t0, t1, calculate_amount_out

Тепер перейдемо до контракту роутеру, для цього створимо клас під назвою «uniswap». Першою створимо функцію «deadline», яка знадобиться нам для майбутніх обчислень (рис 3.24). У функції ми використовуємо нинішній timestamp, та додаємо до нього десять хвилин.

```python
@staticmethod
def deadline():
    return int(time.time() + 10 * 60)
```

Рис 3.24. Функція deadline

Якщо за вказаний проміжок часу транзакція не виконається, то у такому випадку вона відміниться.

Спочатку напишемо read функцію, тобто ми зможемо зчитувати якісь данні із блокчейну без підпису транзакції. У нашому випадку нам знадобиться функція отримання кількості токенів, маючи шлях обміну. Давайте розберемося, що таке шлях обміну взагалі. Наприклад, ми хочемо

обміняти токен один на токен два, у даному випадку шляхом обміну буде масив токену один та токену два і так далі, адрес може бути нескінчена кількість. Назвемо функцію «get_amounts_out_from_path» (рис 3.25). Скористуймося функцією контракту роутеру «getAmountsOut», яка буде майже ідентична до нашої, створеної власноруч функції «calculate_amount_out». Таким чином, кожна пара буде нескінченно вираховуватися, брати кожну пару, все вираховувати та повертати число, яке ми отримаємо. Зробимо також, щоб повертало останній параметр, оскільки перший це кількість резервів, а останній кількість токенів, які ми отримаємо.

```
@retry_on_error()
async def get_amounts_out_from_path(self, amount_in: int, path: List[ChecksumAddress]):
    data = await self.contract.functions.getAmountsOut(amount_in, path).call()
    return data[-1]
```

Рис 3.25. Функція get_amounts_out_from_path

Після read функцій, нарешті можемо перейти до write. Основна відмінність полягає у тому, що write функції вносять зміни до блокчейну завдяки підпису транзакцій, на відміну від read. У даному випадку, усі write функції повертають calldata для транзакції, яка буде відправлена до блокчейну. По суті своїй, calldata — це набір байтів, які ми відправляємо у блокчейн, завдяки їй контракт розуміє, що ми хочемо зробити. Завдяки функції encode_abi, ми зможемо правильно спакувати данні, та відправити до блокчейну, дані ми кодуємо по назві функції. Спочатку нам знадобляться функції додавання та виводу ліквідності до пулу нашого токену. Зазвичай ми працюємо з токенами, у яких уже є ліквідність, але оскільки ми працюємо з нашим токеном, то нам прийдеться робити це самим. Тож створимо функцію «add_liquidity», завдяки якій ми додамо ліквідність у парі з монетою ETH (рис 3.26). Вона повинна приймати адресу токену, кількість токенів, а також кількість ETH, який ми добавляємо у якості ліквідності. А також нам знадобиться функція

«remove_liquidity» (рис 3.26). По суті тут усе теж саме, тільки для того, щоб витягти ліквідність з ETH. Вказуємо адресу токену, а також «liquidity_amount» токен, який ми отримуємо при додаванні ліквідності, це потрібно, щоб блокчейн розумів, що ми володіємо якоюсь ліквідністю.

```python
def add_liquidity(self, token_address: ChecksumAddress, amount_token: int, amount_eth: int) -> HexStr:

    return self.contract.encode_abi('addLiquidityETH',
                                    args=[
                                        token_address,
                                        amount_token,
                                        0,
                                        0,
                                        self.account.address,
                                        self.deadline()
                                    ])

def remove_liquidity(self, token_address: ChecksumAddress, liquidity_amount: int) -> HexStr:

    return self.contract.encode_abi('removeLiquidityETH',
                                    args=[
                                        token_address,
                                        liquidity_amount,
                                        0,
                                        0,
                                        self.account.address,
                                        self.deadline()
                                    ])
```

Рис 3.26. Функції add_liquidity та remove_liquidity

Після добавлення ліквідності, ми можемо перейти до основної суті взаємодії з Uniswap, а саме до обміну криптовалют одна на одну. Нам знадобляться функції обміну такі, як обмін ETH на токен, обмін токену на ETH, та обмін токену на токен. Структура цих функцій дуже схожа на структуру функцій додавання ліквідності. У функції «swap_eth_for_tokens» буде використовуватися amount_out_min токену, який ми отримаємо після сліпажу, тобто ми обмінюємо ETH на наші токени (рис 3.27.). Також зробимо функцію «swap_tokens_for_eth», де ми міняємо вже токени на ETH (рис 3.27.). Використовується кількість токенів, яку ми відправляємо, щоб отримати ETH, та вказуємо мінімальну кількість ETH, який ми отримаємо. У останньому, ми будемо міняти токен на токен (рис 3.27.). Тут ми використовуємо вхідну та вихідну кількість токенів зі сліпажем.

```python
def swap_eth_for_tokens(self, amount_out_min: int, path: List[ChecksumAddress]) -> HexStr:

    return self.contract.encode_abi('swapExactETHForTokens',
                                    args=[
                                        amount_out_min,
                                        path,
                                        self.account.address,
                                        self.deadline()
                                    ])

def swap_tokens_for_eth(self, amount_in: int, amount_out_min: int, path: List[ChecksumAddress]) -> HexStr:

    return self.contract.encode_abi('swapExactTokensForETH',
                                    args=[
                                        amount_in,
                                        amount_out_min,
                                        path,
                                        self.account.address,
                                        self.deadline()
                                    ])

def swap_tokens_for_tokens(self, amount_in: int, amount_out_min: int, path: List[ChecksumAddress]) -> HexStr:

    return self.contract.encode_abi('swapExactTokensForTokens',
                                    args=[
                                        amount_in,
                                        amount_out_min,
                                        path,
                                        self.account.address,
                                        self.deadline()
                                    ])
```

Рис 3.27. Функції для обміну токенів

Також, щоб усе це працювало, нам знадобиться створити основну функцію обміну, так і назвемо її «swap». Першим чином, нам потрібно написати «base_path», також додатково додамо параметр «additional_to_token», який знадобиться у випадку, якщо путь знадобиться зробити більш довгим, наприклад не на два токени, а на три (рис 3.28). Далі створимо змінну «target_token» - це наш фінальний токен, який нам потрібен для обміну (рис 3.28). Наступним нам знадобиться шлях нашого обміну, назвемо його «path» (рис 3.28). У якому ми просто беремо адреси по порядку для усіх токенів з нашого base_path. Також нам знадобиться кількість токенів, які ми отримаємо з шуканої кількості токенів у форматі wei, а так же вкажемо path, який отримали вище (рис 3.28). Переходимо до самого процесу обміну. В першу чергу, перевіряємо, чи міняємо ми ETH на токен, або навпаки. У всіх випадках буде використовуватися WETH (wrapped eth), який є аналогом ETH, оскільки звичайна версія потребує більше енергоресурсів, тож і газ за роботу із ним вище, тому для обмінів

усі використовують його обернуту версію. У першому випадку, якщо ми міняємо ETH на токен, то ми будемо використовувати нашу функцію swap_eth_for_tokens, вказавши amount_out_min та path (рис 3.28). Також оскільки ми міняємо ETH, нам потрібно обов’язково вказати value, яке не може бути нульовим значенням, тому що ми у транзакції відправляємо ETH з нашого гаманця на контракт uniswap. Наостанок, вказуємо додатковий параметр need_approve. Він потрібен для усіх ERC20 токенів, так як він дозволяє контракту узяти наші токени і зробити з ними будь що. У нашому випадку - це обміняти один токен на інший. Наступним чином, перевіряємо чи міняємо ми токен на ETH (рис 3.28). Використаємо наступну нашу функцію, а саме swap_tokens_for_eth. Ми робимо все майже те саме, але вказуємо кількість токенів, шлях, оскільки ми нікуди ETH не відправляємо, то value буде 0, а також вкажемо need_approve True, нам потрібно попередньо надати контракту доступ. В інакшому випадку, наша транзакція ніяк не підтвердиться. І на останок залишається обмін токену на токен, у якому все теж саме, тільки функція буде використовуватися swap_tokens_for_tokens (рис 3.28).

```python
async def swap(self, from_token: Token, to_token: Token, amount: Amount, additional_to_token: Optional[Token] = None):
    base_path = [from_token, to_token] if not additional_to_token else [from_token, to_token, additional_to_token]

    target_token = base_path[-1]

    path = [token.address for token in base_path]

    amount_out_min = await self.get_amounts_out_from_path(amount.wei_amount, path)

    if from_token.name == 'WETH':
        data = self.swap_eth_for_tokens(amount_out_min, path)
        value = amount.wei_amount
        need_approve = False

    elif target_token.name == 'WETH':
        data = self.swap_tokens_for_eth(amount.wei_amount, amount_out_min, path)
        value = 0
        need_approve = True
    else:
        data = self.swap_tokens_for_tokens(amount.wei_amount, amount_out_min, path)
        value = 0
        need_approve = True
```

Рис 3.28. Початкові змінні та перевірка токенів у функції swap

Далі ми дадаємо перевірку (рис 3.29). Якщо токену потрібен approve, але у нас не вдалося це зробити, то на жаль, ми завершуємо роботу функції, тому що у нас щось пішло не так, та транзакція не підтвердилася. В іншому ж випадку ми переходимо до транзакції обміну (рис 3.29). Ми викликаємо транзакцію, у якій вказуємо адресу нашого роутеру, вказуємо нашу дату для обмінів, яку ми сформували вище. Кожна функція encode_abi завдяки ABI використовує певну функцію та наші аргументи, і пакує це все у масив байтів, які ми відправляємо на контракт. Далі, ми підписуємо та підтверджуємо нашу транзакцію (рис 3.29). І потім перевіряємо результати (рис 3.29). Якщо результати у нас успішні, то ми

```
if need_approve:
    if not await self.account.approve_token(
            self.contract.address,
            amount,
            from_token,
    ):
        return
tx = await self.account.build_tx(
    contract_address=self.contract.address,
    data=data,
    value=value,
)
result = await self.account.sign_and_send(tx)
if result.status:
    logger.success(f'{self.account.acc_info} - succesfully swapped {amount.amount} {from_token.name} for {base_path[-1].name} '
                   f'{self.account.explorer(result.tx_hash)}')
else:
    logger.error(f'{self.account.acc_info} - tx swap not succesfully {self.account.explorer(result.tx_hash)}')
```

успішно обміняли наші токени.

Рис 3.29. Перевірка підтвердження, транзакція обміну, підпис, результати

Наступна функція буде під назвою «liquidity», у функції ми вказуємо наш токен, у який ми хочемо додати, чи забрати ліквідність, а також кількість токенів (рис 3.30). Останнє буде опційним, оскільки у випадку витягування ліквідності воно нам не знадобиться. Точно так же буде і з токеном ETH. А також нам знадобиться параметр is_add, тобто якщо він, True і ми додаємо ліквідність, то нам знадобляться параметри amount та amount_eth, щоб їх додати. Для того, щоб додати ліквідність, нам спочатку буде потрібно підтвердити наш токен. Ми підтверджуємо токен, який ми будемо міняти, а також ми будемо підтверджувати наш WETH (рис 3.30). Після цього, ми робимо дату для додавання ліквідності, де вказуємо

адресу, wei_amount токену та wei_amount ETH (рис 3.30). Після створення дати, нам потрібно вказати value, тому що ми відправляємо деяку кількість

```
async def liquidity(self, token: Token, amount: Optional[Amount] = None, amount_eth: Optional[Amount] = None, is_add=True):
    if is_add:
        await self.account.approve_token(self.contract.address, amount, token)
        await self.account.approve_token(self.contract.address, amount, WETH[self.account.chain.name])
        data = self.add_liquidity(
            token.address,
            amount.wei_amount,
            amount_eth.wei_amount
        )
        value = amount_eth.wei_amount
        text = f"succesfully added {amount.amount} {token.name} and {amount_eth.amount} eth in liquidity"
```

ETH на контракт (рис 3.30). Та виводимо текст (рис 3.30).

Рис 3.30. Додавання ліквідності

Тепер розглянемо випадок, якщо ми хочемо не додати, а забрати нашу ліквідність. Першим, прописуємо адресу (рис 3.31). У якій беремо нашу пару та використовуємо функцію pool_address, яку створили майже у самому початку, завдяки їй ми обчислюємо адресу токена, який ми отримаємо під час додавання ліквідності. Зазвичай він потрібен, щоб витягти ліквідність та повернути свої токени. Пропишемо токен ліквідності (рис 3.31). Нам знадобиться назва пари, адреса, яку ми вище отримали, а також decimals, який завжди статичний, та дорівнює вісімнадцяти. Також нам знадобиться liquidity_amount (рис 3.31). Який буде відображати баланс на нашому рахунку. Щоб його витягти нам знадобиться підтвердження, оскільки це ERC20 токен. Тож створимо його по аналогічному принципу, як робили трохи раніше (рис 3.31). І таким же чином створюємо дату для remove_liquidity (рис 3.31). Але у цей раз value буде дорівнювати нулю, оскільки ми нікуди не будемо відправляти ETH.

```
else:
    address = UniswapPair(self.account, token, WETH[self.account.chain.name]).pool_address

    liq_token = Token(name=f'UNI PAIR {token.name}-WETH', _address=address, decimals=18)

    liquidity_amount = await self.account.get_balance(liq_token)

    await self.account.approve_token(self.contract.address, liquidity_amount, liq_token)

    data = self.remove_liquidity(
        token.address, liquidity_amount.wei_amount
    )
    value = 0
    text = f"succesfully removed liquidity"
```

Рис 3.31. Виведення ліквідності

По аналогічному принципу до минулої, створюємо саму транзакцію, у якій вказуємо контракт роутеру, дату і наш value. Далі, підписуємо та відправляємо транзакцію і отримуємо результати.

Усі основні функції для обміну токенів, додавання та виводу ліквідності ми написали, настав час провести декілька тестів. Першим тестом буде звичайний обмін токенів (рис 3.32). Будемо міняти WETH на USDT. Вказуємо вхідну кількість ETH. Спочатку ми викликаємо просто обмін з ETH у USDT, потім отримуємо баланс USDT після обміну і запускаємо зворотну функцію обміну з USDT у ETH.

```
async def test_swap1():
    a = Account(key=key, chain=Chains.arbi)
    u = Uniswap(a)

    await u.swap(
        WETH[a.chain.name], USDT_ARB, Amount(to_wei( number: 0.0005,  unit: 'ether'), 0.0005)
    )
    balance_usdt = await a.get_balance(USDT_ARB)
    await u.swap(USDT_ARB, WETH[a.chain.name], balance_usdt)
```

Рис 3.32. Тест обміну токенів

Для майбутнього обміну ETH на наш токен TESIS, нам знадобиться добавити ліквідність у цю пару. Тож створимо функцію під назвою «test_liq» (рис 3.33). У якій спочатку перевіримо параметр is_add, і якщо він True, то додамо ліквідність, в інакшому ж випадку виведемо її.

```
async def test_liq(is_add=True):
    a = Account(key=key, chain=Chains.arbi)
    u = Uniswap(a)

    if is_add:
        await u.liquidity(
            THESIS,
            Amount(to_wei( number: 25000,  unit: 'ether'), 25000),
            Amount(to_wei( number: 0.001,  unit: 'ether'), 0.001),
        )
    else:
        await u.liquidity(
            THESIS,
            is_add=False
        )
```

Рис 3.33. Тест додавання та виводу ліквідності

Тепер зробимо тест більш складного шляху обміну, щоб продемонструвати всі можливості. Створимо функцію і назвемо її «test_swap2» (рис 3.34). Спочатку напишемо обмін WETH на наш токен, тобто TESIS, у який ми до цього додали ліквідність. Далі, ми беремо щойно отриманий TESIS та міняємо вже його. Додатково додаємо additional_to_token USDT, тобто фінальний токен у нас буде не ETH, а USDT. У кінці ми перевіряємо баланс USDT та міняємо його на TESIS.

```
async def test_swap2():
    a = Account(key=key, chain=Chains.arbi)
    u = Uniswap(a)

    await u.swap(
        WETH[a.chain.name], THESIS, Amount(to_wei( number: 0.0005,  unit: 'ether'), 0.0005)
    )
    await u.swap(THESIS, WETH[a.chain.name], Amount(to_wei( number: 1000,  unit: 'ether'), 1000))
    await u.swap(THESIS, WETH[a.chain.name], Amount(to_wei( number: 1000,  unit: 'ether'), 1000), additional_to_token=USDT_ARB)
    balance_usdt = await a.get_balance(USDT_ARB)
    await u.swap(USDT_ARB, WETH[a.chain.name], balance_usdt, additional_to_token=THESIS)
```

Рис 3.34. Більш складний тест обміну токенів

Усі функції для роботи з uniswap ми зробили. Але для зручності пропоную зробити селектор за допомогою бібліотеки questionary. Нам знадобиться створити 3 функції для запуску наших тестів: перша для uniswap, друга для web3 і остання для тесту нашого банківського контракту. Створимо функцію під назвою uni_task (рис 3.35). У функції створюємо змінну task, яка завдяки селектору буде приймати у собі якесь значення, далі залежно від задачі ми обираємо потрібний нам варіант.

```python
async def uni_task():
    task = await questionary.select(
        message: "choice uniswap task...",
        choices=[
            Choice( title: "1) add liqiudity",  value: "add"),
            Choice( title: "2) remove",  value: "remove"),
            Choice( title: "3) swap test1", test_swap1),
            Choice( title: "4) swap test2", test_swap2),
            Choice( title: f" exit",  value: 'e'),
        ],
        qmark="",
        pointer="⟹ ",
    ).ask_async()
    if task == 'add':
        return await test_liq(True)
    if task == 'remove':
        return await  test_liq(False)
    if task == 'e':
        return
    else:
        return await task()
```

Рис 3.35. Функція uni_task

За тим же самим принципом, створимо функції для вибору режиму роботи для тесту можливостей web3 та нашого контракту (рис 3.36). Ну і так само, створимо функцію для вибору модулів для роботи із ними.

```python
async def main():
    while True:
        c = await questionary.select(
            message: "Select modules to work with...",
            choices=[
                Choice( title: "1) test w3", w3_task),
                Choice( title: "2) test uniswap", uni_task),
                Choice( title: "3) test bank contract", bank_test),
                Choice( title: f" exit",  value: 'e'),
            ],
            qmark="",
            pointer="⟹ ",
        ).ask_async()
        if c == 'e':
            sys.exit()
        else:
            await c()
```

Рис 3.36. Функція вибору модуля для роботи

Залишився останній, та самий цікавий етап у будь якій роботі — а саме результат. Запустимо нашу програму та подивимося, що з усього цього вийшло. Оскільки результати тестів взаємодії з web3 та взаємодії з нашим контрактом ми бачили, пропоную тільки звернути увагу на

взаємодію з uniswap. Спочатку додамо ліквідність до пари ефіру з нашим токеном (рис 3.37.).

```
utils:approve_token:248 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - approving thesis
utils:check_status_tx:71 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - awaiting transaction confirmation
utils:approve_token:264 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi- successfully approved thesis
ee6dd275736746058fb2191990bd2b1262
utils:approve_token:248 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - approving WETH
utils:check_status_tx:71 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - awaiting transaction confirmation
utils:approve_token:264 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi- successfully approved WETH
a38d6ba0abf98a16a26245aeeb4df905ca
utils:check_status_tx:71 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - awaiting transaction confirmation
__main__:liquidity:314 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - succesfully added 50000 thesis and
```

Рис 3.37 Тест додавання ліквідності до пулу

Все пройшло успішно, спочатку програма надала підтвердження для WETH та THESIS, а потім додала їх до пулу ліквідності. Тепер, коли усі приготування завершено, ми можемо перейти до тесту обміну токенів (рис 3.38).

```
utils:check_status_tx:72 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - awaiting transaction confirmation
__main__:swap:282 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - succesfully swapped 0.0005 WETH for thesis ht
utils:approve_token:249 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - approving thesis
utils:check_status_tx:72 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - awaiting transaction confirmation
utils:approve_token:265 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi- successfully approved thesis
:7a3cc0262399860f429dad79c4c94083a
utils:check_status_tx:72 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - awaiting transaction confirmation
__main__:swap:282 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - succesfully swapped 10000 thesis for WETH htt
utils:approve_token:249 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - approving thesis
utils:check_status_tx:72 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - awaiting transaction confirmation
utils:approve_token:265 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi- successfully approved thesis
)7e609790c1f603fbbf3822bf8d43e4bec
utils:check_status_tx:72 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - awaiting transaction confirmation
__main__:swap:282 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - succesfully swapped 10000 thesis for USDT htt
utils:approve_token:249 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - approving USDT
utils:check_status_tx:72 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - awaiting transaction confirmation
utils:approve_token:265 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi- successfully approved USDT
31ef8b05389a5ab37a608bd8ff36027ebc
utils:check_status_tx:72 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - awaiting transaction confirmation
__main__:swap:282 - 0xA93afA7FB2C6edee238f1589E2E8680331C83682:arbi - succesfully swapped 0.958903 USDT for thesis
```

Рис 3.38 Тест обміну токенів

Як ми бачимо, програма успішно обміняла WETH на TESIS, після чого надала підтвердження та обміняла назад у WETH. Також іншу частину вона обміняла на USDT та надала йому підтвердження. Таким чином наша програма успішно пройшла усі випробування, а дослідження можна вважати завершеним.

ВИСНОВКИ

У цій кваліфікаційній роботі було проведено глибокий теоретичний аналіз основних елементів блокчейну та розроблено приклад розумного контракту на мові програмування Solidity. Ми також дослідили можливості інтеграції з блокчейном Ethereum за допомогою мови програмування Python. Це дозволило отримати комплексне уявлення про сучасні блокчейн технології та їх практичне застосування. В результаті проведених досліджень отримано такі **висновки**:

1. В роботі було детально розглянуто основні поняття та елементи блокчейну, такі як децентралізація, консенсусні алгоритми, смарт-контракти, блоки, ланцюги блоків і тд. Це забезпечило глибоке розуміння принципів функціонування блокчейн-систем та ознайомлення з основами блокчейн-технологій.
2. Було проведено аналіз структури Ethereum та ролі його віртуальної машини, або ж просто EVM. Розглянуто важливі зміни в Ethereum 2.0, такі як шардінг та ролапи. Ці знання є критично важливими для розробки та впровадження смарт-контрактів на цій платформі.
3. Описано основи мови програмування Solidity та створено простий смарт-контракт з функціональністю, аналогічною банківським системам. Демонстрація базових концепцій Solidity забезпечила практичний досвід розробки смарт-контрактів.
4. Було досліджено інструменти для взаємодії з блокчейном Ethereum за допомогою Python, такі як web3.py. Демонстрація основних методів, таких як підключення та отримання інформації з блокчейну, дозволила отримати практичний досвід взаємодії із ним.

5. Ми навчилися працювати не лише зі своїми, а також із чужими контрактами на прикладі defi протоколу Uniswap. Під час взаємодії з яким, будо додано ліквідність, проведені обміни токенів, і все це завдяки Python. Це демонструє нашу здатність інтегруватися з існуючими децентралізованими фінансовими платформами, що є важливим кроком у сучасному світі блокчейн-технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Tapscott A., Tapscott D. Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World / A. Tapscott, D. Tapscott // New York, Portfolio, Penguin Group. – 2016. – P. 23–45, 158–182.
2. What is Blockchain Technology? [Електронний ресурс]. – Режим доступу: https://www.coindesk.com/learn/what-is-blockchain-technology?form=MG0AV3
3. What is Decentralized Finance (DeFi)? [Електронний ресурс]. – Режим доступу: https://coinmarketcap.com/academy/article/1788c0e6-96c2-4716-967b-15b32416860d
4. Debunking the Narratives About Cryptocurrency and Financial Inclusion [Електронний ресурс]. – Режим доступу: https://www.brookings.edu/articles/debunking-the-narratives-about-cryptocurrency-and-financial-inclusion/?form=MG0AV3.
5. What Is Ethereum and How Does It Work? [Електронний ресурс]. – Режим доступу: https://www.investopedia.com/terms/e/ethereum.asp.
6. Antonopoulos A. M., Wood G. Mastering Ethereum: Building Smart Contracts and DApps / A. M. Antonopoulos, G. Wood // Sebastopol, CA: O’Reilly Media. – 2018. – P. 267–324.
7. What Is a Dapp? Decentralized Apps Explained [Електронний ресурс]. – Режим доступу: https://www.coindesk.com/learn/what-is-a-dapp-decentralized-apps-explained/?form=MG0AV3
8. How DeFi Will Reconfigure Financial Services In The Next Decade [Електронний ресурс]. – Режим доступу: https://www.forbes.com/councils/forbesfinancecouncil/2022/04/19/how-defi-will-reconfigure-financial-services-in-the-next-decade/?form=MG0AV3

9. What Are Liquidity Pools in DeFi? [Електронний ресурс]. – Режим доступу: https://academy.binance.com/en/articles/what-are-liquidity-pools-in-defi?form=MG0AV3
10. Blockchain Facts: What Is It, How It Works, and How It Can Be Used [Електронний ресурс]. – Режим доступу: https://www.investopedia.com/terms/b/blockchain.asp
11. Bashir I. Mastering Blockchain: Unlocking the Power of Cryptocurrencies, Smart Contracts, and Decentralized Applications / I. Bashir // Birmingham, Packt Publishing. – 2017. - P. 45–60.
12. The Truth About Blockchain [Електронний ресурс]. – Режим доступу: https://hbr.org/2017/01/the-truth-about-blockchain?form=MG0AV3
13. Nodes and Clients [Електронний ресурс]. – Режим доступу: https://ethereum.org/en/developers/docs/nodes-and-clients/?form=MG0AV3
14. What Is Crypto Mining and How Does It Work? [Електронний ресурс]. – Режим доступу: https://academy.binance.com/uk/articles/what-is-crypto-mining-and-how-does-it-work?form=MG0AV3
15. Ethereum Whitepaper [Електронний ресурс]. – Режим доступу: https://ethereum.org/en/whitepaper/?form=MG0AV3
16. Understanding Ethereum [Електронний ресурс]. – Режим доступу: https://medium.com/mindroast/understanding-ethereum-d818ea4e70a9
17. Ethereum Roadmap [Електронний ресурс]. – Режим доступу: https://ethereum.org/en/roadmap/?form=MG0AV3
18. Antonopoulos A. M., Wood G. Mastering Ethereum: Building Smart Contracts and DApps / A. M. Antonopoulos, G. Wood // Sebastopol, CA: O’Reilly Media. – 2018. – P. 297.
19. Drescher D. Blockchain Basics: A Non-Technical Introduction in 25 Steps / D. Drescher // New York: Apress. – 2017. – P. 85–100.

20. ERC-20 Tokens [Електронний ресурс]. – Режим доступу: https://ethereum.org/en/developers/docs/standards/tokens/erc-20/?form=MG0AV3
21. What Is Ethereum 2.0 And Why Does It Matter? [Електронний ресурс]. – Режим доступу: https://academy.binance.com/en/articles/what-is-ethereum-2-0-and-why-does-it-matter?form=MG0AV3
22. What Is Ethereum 2.0? Ethereum's Consensus Layer and Merge Explained [Електронний ресурс]. – Режим доступу: https://decrypt.co/resources/what-is-ethereum-2-0
23. What Is Ethereum 2.0? Understanding The Ethereum Merge [Електронний ресурс]. – Режим доступу: https://www.forbes.com/advisor/investing/cryptocurrency/what-is-ethereum-2-merge/?form=MG0AV3
24. What Are Blockchain Rollups? [Електронний ресурс]. – Режим доступу: https://www.ledger.com/academy/what-are-blockchain-rollups?form=MG0AV3
25. What Are Rollups? ZK Rollups and Optimistic Rollups Explained [Електронний ресурс]. – Режим доступу: https://www.coindesk.com/learn/what-are-rollups-zk-rollups-and-optimistic-rollups-explained/?form=MG0AV3
26. What Are Rollups? How blockchain rollups work [Електронний ресурс]. – Режим доступу: https://www.moonpay.com/learn/blockchain/what-are-rollups?form=MG0AV3
27. Solidity Documentation [Електронний ресурс]. – Режим доступу: https://docs.soliditylang.org/en/v0.8.28/?form=MG0AV3
28. Solidity vs Python [Електронний ресурс]. – Режим доступу: https://www.gyata.ai/solidity/solidity-vs-python?form=MG0AV3
29. ABI Specification [Електронний ресурс]. – Режим доступу: https://docs.soliditylang.org/en/latest/abi-spec.html?form=MG0AV3

30. Solidity ABI Overview [Електронний ресурс]. – Режим доступу: https://www.alchemy.com/overviews/solidity-abi?form=MG0AV3.
31. Reading and Writing to a State Variable [Електронний ресурс]. – Режим доступу: https://solidity-by-example.org/state-variables/?form=MG0AV3
32. Solidity Events Logging Essential Guide [Електронний ресурс]. – Режим доступу: https://www.soliditylibraries.com/guides/solidity-events-logging-essential-guide/?form=MG0AV3
33. MIT License [Електронний ресурс]. – Режим доступу: https://spdx.org/licenses/MIT.html?form=MG0AV3
34. ERC20 Token API [Електронний ресурс]. – Режим доступу: https://docs.openzeppelin.com/contracts/4.x/api/token/erc20?form=MG0AV3
35. Web3.py Documentation [Електронний ресурс]. – Режим доступу: https://web3py.readthedocs.io/en/stable/
36. JSON-RPC API [Електронний ресурс]. – Режим доступу: https://ethereum.org/en/developers/docs/apis/json-rpc/?form=MG0AV3
37. Smart Contract ABI [Електронний ресурс]. – Режим доступу: https://metana.io/blog/smart-contract-abi/?form=MG0AV3
38. What Is Uniswap And How Does It Work? [Електронний ресурс]. – Режим доступу: https://academy.binance.com/uk/articles/what-is-uniswap-and-how-does-it-work?form=MG0AV3

ДОДАТОК

Програмний код реалізації банківського смарт-контракту

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract Bank {
    IERC20 public token;
    uint256 public stakingRewardRate;
    uint256 public constant ONE_YEAR = 365 days;
```

```
address public owner;

struct DepositInfo {
    uint256 amount;
    uint256 depositTime;
}

struct StakeInfo {
    uint256 amount;
    uint256 stakeTime;
    uint256 rewardDebt;
}

mapping(address => DepositInfo) public deposits;
mapping(address => StakeInfo) public stakes;

event Deposit(address indexed user, uint256 amount);
event Withdraw(address indexed user, uint256 amount);
event Stake(address indexed user, uint256 amount);
event ClaimRewards(address indexed user, uint256 reward);
event WithdrawStake(address indexed user, uint256 amount);

constructor(IERC20 _token, uint256 _stakingRewardRate) {
    token = _token;
    stakingRewardRate = _stakingRewardRate;
    owner = msg.sender;
```

```
}

modifier onlyOwner() {
    require(msg.sender == owner, "Not owner");
    _;
}

// Deposit ETH into the contract
function deposit() external payable {
    require(msg.value > 0, "Cannot deposit 0 ETH");

    deposits[msg.sender].amount += msg.value;
    deposits[msg.sender].depositTime = block.timestamp;

    emit Deposit(msg.sender, msg.value);
}

// Withdraw ETH from the contract
function withdraw() external {
    uint256 depositAmount = deposits[msg.sender].amount;
    require(depositAmount > 0, "Nothing to withdraw");

    deposits[msg.sender].amount = 0; // Reset deposit amount

    payable(msg.sender).transfer(depositAmount);
    emit Withdraw(msg.sender, depositAmount);
}
```

```
    // Deposit tokens into the contract for staking
    function depositWithInterest(uint256 _amount) external {
        require(_amount > 0, "Cannot deposit 0 tokens");
        require(stakes[msg.sender].amount == 0, "User can have only 1 deposit at time");
        token.transferFrom(msg.sender, address(this), _amount);

        stakes[msg.sender].amount += _amount;
        stakes[msg.sender].stakeTime = block.timestamp;

        emit Stake(msg.sender, _amount);
    }

    // Calculate rewards based on amount and duration
    function calculateRewards(uint256 _amount, uint256 _duration) public view returns (uint256) {
        return (_amount * stakingRewardRate * _duration) / ONE_YEAR / 100;
    }

    // Claim rewards for staked tokens
    function claimRewards() external {
        StakeInfo storage userStake = stakes[msg.sender];
        require(userStake.amount > 0, "No tokens deposited");

        uint256 stakingDuration = block.timestamp - userStake.stakeTime;
        uint256 rewards = calculateRewards(userStake.amount, stakingDuration);
```

```
        require(rewards > 0, "No rewards at this moment");

        userStake.stakeTime = block.timestamp; // Update stake time

        require(token.transfer(msg.sender, rewards), "Reward transfer failed");

        emit ClaimRewards(msg.sender, rewards);
    }

    // Withdraw staked tokens with rewards
    function withdrawDepositWithInterest() external {
        StakeInfo storage userStake = stakes[msg.sender];
        require(userStake.amount > 0, "No tokens deposited");

        uint256 stakingDuration = block.timestamp - userStake.stakeTime;
        uint256 amountToWithdraw = userStake.amount +
calculateRewards(userStake.amount, stakingDuration);

        userStake.amount = 0; // Reset stake amount
        userStake.stakeTime = 0; // Reset stake time

        token.transfer(msg.sender, amountToWithdraw);
        emit WithdrawStake(msg.sender, amountToWithdraw);
    }

    // Set a new staking reward rate
    function setStakingRewardRate(uint256 _newRate) external onlyOwner {
        stakingRewardRate = _newRate;
```

```
    }

    // Rescue Ether from the contract
    function rescueEther() external onlyOwner {
        (bool success,) = payable(msg.sender).call{value: address(this).balance}("");
        require(success, "Failed to send ether");
    }

    // Rescue any ERC20 tokens from the contract
    function rescueERC20(address _token) external onlyOwner {
        uint256 balance = IERC20(_token).balanceOf(address(this));
        require(balance > 0, "Nothing to rescue");

        require(IERC20(_token).transfer(msg.sender, balance), "Token transfer failed");
    }

    // Change ownership of the contract
    function transferOwnership(address _newOwner) external onlyOwner {
        require(_newOwner != address(0), "New owner is the zero address");

        owner = _newOwner;
    }
}
```

Програмний код реалізації взаємодії з базовими методами Web3

```
from pprint import pprint
from config import key
```

```
from constants import THESIS
from models import Chains
from utils import Account
import asyncio

class Test:
    def __init__(self, account: Account):
        self.account = account

    async def test_w3(self):

        res = await self.account.w3.is_connected()
        print(f"connect status {res}")



    async def get_block(self):
        block = await self.account.w3.eth.get_block('latest')
        pprint(dict(block))

    async def test_balance(self):
        eth = await self.account.get_eth_balance()
        thesis_bal = await self.account.get_balance(THESIS)
        print(f"eth balance in {self.account.chain.name} : {eth.amount} eth")
        print(f"thesis balance in {self.account.chain.name} : {thesis_bal.amount}
thesis")

async def test():
    a = Account(key=key, chain=Chains.arbi)
```

```
    t = Test(a)

    await t.test_w3()
    print('\n')
    await t.get_block()
    print("\n")
    await t.test_balance()

if __name__ == '__main__':
    asyncio.set_event_loop_policy(asyncio.WindowsSelectorEventLoopPolicy())
    asyncio.run(test())
```

Програмний код реалізації взаємодії з банківським смарт-контрактом

```
import asyncio
from enum import Enum
from typing import Tuple, Optional

from eth_typing import HexStr
from eth_utils import to_checksum_address, to_wei
from loguru import logger

from config import key
from constants import THESIS
from helpful_utils import retry_on_error, format_from_wei
from models import Amount, Chains
from utils import Account

BANK_ABI = '[{"inputs":[{"internalType":"contract IERC20","name":"_token","type":"address"},{"internalType":"uint256","name"
```

:"_stakingRewardRate","type":"uint256"}],"stateMutability":"nonpayable","type
":"constructor"},{"anonymous":false,"inputs":[{"indexed":true,"internalType":"
address","name":"user","type":"address"},{"indexed":false,"internalType":"uint
256","name":"reward","type":"uint256"}],"name":"ClaimRewards","type":"eve
nt"},{"anonymous":false,"inputs":[{"indexed":true,"internalType":"address","na
me":"user","type":"address"},{"indexed":false,"internalType":"uint256","name"
:"amount","type":"uint256"}],"name":"Deposit","type":"event"},{"anonymous":
false,"inputs":[{"indexed":true,"internalType":"address","name":"user","type":"
address"},{"indexed":false,"internalType":"uint256","name":"amount","type":"
uint256"}],"name":"Stake","type":"event"},{"anonymous":false,"inputs":[{"ind
exed":true,"internalType":"address","name":"user","type":"address"},{"indexed
":false,"internalType":"uint256","name":"amount","type":"uint256"}],"name":"
Withdraw","type":"event"},{"anonymous":false,"inputs":[{"indexed":true,"inter
nalType":"address","name":"user","type":"address"},{"indexed":false,"internalT
ype":"uint256","name":"amount","type":"uint256"}],"name":"WithdrawStake","
type":"event"},{"inputs":[],"name":"ONE_YEAR","outputs":[{"internalType":"
uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"functio
n"},{"inputs":[{"internalType":"uint256","name":"_amount","type":"uint256"},
{"internalType":"uint256","name":"_duration","type":"uint256"}],"name":"calc
ulateRewards","outputs":[{"internalType":"uint256","name":"","type":"uint256"
}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"claimRewar
ds","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[],"
name":"deposit","outputs":[],"stateMutability":"payable","type":"function"},{"i
nputs":[{"internalType":"uint256","name":"_amount","type":"uint256"}],"name
":"depositWithInterest","outputs":[],"stateMutability":"nonpayable","type":"fun
ction"},{"inputs":[{"internalType":"address","name":"","type":"address"}],"na
me":"deposits","outputs":[{"internalType":"uint256","name":"amount","type":"
uint256"},{"internalType":"uint256","name":"depositTime","type":"uint256"}],
"stateMutability":"view","type":"function"},{"inputs":[],"name":"owner","outpu
ts":[{"internalType":"address","name":"","type":"address"}],"stateMutability":"
view","type":"function"},{"inputs":[{"internalType":"address","name":"_token"
,"type":"address"}],"name":"rescueERC20","outputs":[],"stateMutability":"nonp
ayable","type":"function"},{"inputs":[],"name":"rescueEther","outputs":[],"state
Mutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"uint25
6","name":"_newRate","type":"uint256"}],"name":"setStakingRewardRate","out
puts":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internal
Type":"address","name":"","type":"address"}],"name":"stakes","outputs":[{"inte

rnalType":"uint256","name":"amount","type":"uint256"},{"internalType":"uint256","name":"stakeTime","type":"uint256"},{"internalType":"uint256","name":"rewardDebt","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"stakingRewardRate","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"token","outputs":[{"internalType":"contract IERC20","name":"","type":"address"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"_newOwner","type":"address"}],"name":"transferOwnership","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[],"name":"withdraw","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[],"name":"withdrawDepositWithInterest","outputs":[],"stateMutability":"nonpayable","type":"function"}]'

```
class Tasks(Enum):
    dep_eth = 'dep'
    dep_token_w_interest = "dep_w"
    reward = 'rew'
    withdraw_eth = 'with'
    withdraw_token_w_interest = "with_w"


class Bank:
    def __init__(self, account: Account):
        self.account = account
        self.contract = self.account.get_contract('0x03bcbfb6266c771f004d64765e01a7b50066ee26', BANK_ABI)

    ########################
    # read
```

```
    ########################

    @retry_on_error()
    async def get_deposit_by_address(self, address: str):
        data = await self.contract.functions.deposits(to_checksum_address(address)).call()
        return data[0]

    @retry_on_error()
    async def get_stakes_by_address(self, address: str):
        data = await self.contract.functions.deposits(to_checksum_address(address)).call()
        return data[0]

    ########################
    # write
    ########################

    def deposit_eth(self) -> HexStr:
        return self.contract.encode_abi('deposit',
                                        )

    def deposit_token_with_interest(self, amount: int) -> HexStr:
        return self.contract.encode_abi('depositWithInterest',
                                        args=[
                                            amount,
                                        ])
```

```
    def claim_reward_token_with_interest(self) -> HexStr:
        return self.contract.encode_abi('claimRewards',
                                        )

    def withdraw_eth(self) -> HexStr:
        return self.contract.encode_abi('withdraw',
                                        )

    def withdraw_eth_token_with_interest(self) -> HexStr:
        return self.contract.encode_abi('withdrawDepositWithInterest',
                                        )

    async def task(self, amount_eth: Optional[Amount] = None, amount_token:
Optional[Amount] = None, task: Tasks = Tasks.dep_eth):

        if task.value == Tasks.dep_eth.value:
            data = self.deposit_eth()
            value = amount_eth.wei_amount
            need_approve = False
            text = 'deposited eth'
        elif task.value == Tasks.dep_token_w_interest.value:
            data = self.deposit_token_with_interest(amount_token.wei_amount)
            value = 0
            need_approve = True
            text = 'deposited THESIS'
        elif task.value == Tasks.reward.value:
            data = self.claim_reward_token_with_interest()
            value = 0
```

```
    need_approve = False
    text = 'claimed rewards in THESIS'
elif task.value == Tasks.withdraw_eth.value:
    data = self.withdraw_eth()
    value = 0
    need_approve = False
    text = 'withdrawed eth'
else:
    data = self.withdraw_eth_token_with_interest()
    value = 0
    need_approve = False
    text = 'withdrawed THESIS'
if need_approve:
    if not await self.account.approve_token(
            self.contract.address,
            amount_token,
            THESIS,
    ):
        return

tx = await self.account.build_tx(
    contract_address=self.contract.address,
    data=data,
    value=value,
)
result = await self.account.sign_and_send(tx)
if result.status:
```

```
            logger.success(f'{self.account.acc_info} - succesfully {text} {self.account.explorer(result.tx_hash)}')
        else:
            logger.error(f'{self.account.acc_info} - tx swap not succesfully {self.account.explorer(result.tx_hash)}')


eth_deposit_to_contract = Amount(to_wei(0.0001, 'ether'), 0.0001)
THESIS_deposit_to_contract = Amount(to_wei(1000, 'ether'), 1000)


async def test_deposits():
    a = Account(key=key, chain=Chains.arbi)
    b = Bank(a)

    await b.task(amount_eth=eth_deposit_to_contract, task=Tasks.dep_eth)
    await asyncio.sleep(5)  # little wait
    await b.task(amount_token=THESIS_deposit_to_contract, task=Tasks.dep_token_w_interest)

    logger.info(f"get_deposit_by_address result: {format_from_wei(await b.get_deposit_by_address(a.address), 18)}")
    logger.info(f"get_stakes_by_address result: {format_from_wei(await b.get_deposit_by_address(a.address), 18)}")


async def test_withdrawals():
    a = Account(key=key, chain=Chains.arbi)
    b = Bank(a)
```

```
    await b.task(task=Tasks.reward)
    await asyncio.sleep(5)  # little wait
    await b.task(task=Tasks.withdraw_eth)
    await asyncio.sleep(5)  # little wait
    await b.task(task=Tasks.withdraw_token_w_interest)

    logger.info(f"get_deposit_by_address result: {format_from_wei(await b.get_deposit_by_address(a.address), 18)}")
    logger.info(f"get_stakes_by_address result: {format_from_wei(await b.get_deposit_by_address(a.address), 18)}")


async def test_all():
    await test_deposits()
    await asyncio.sleep(5)  # little wait
    await test_withdrawals()


if __name__ == '__main__':
    asyncio.set_event_loop_policy(asyncio.WindowsSelectorEventLoopPolicy())
    asyncio.run(test_all())
```

Програмний код реалізації взаємодії з uniswap

```
import asyncio
import time
from pprint import pprint
from typing import Tuple, List, Optional
```

```
from eth_abi import packed
from eth_typing import ChecksumAddress, HexStr
from eth_utils import to_checksum_address, keccak, to_wei
from hexbytes import HexBytes
from loguru import logger

from config import key
from helpful_utils import retry_on_error
from models import Token, Amount, Chains
from utils import Account
from constants import WETH, USDT_ARB, THESIS

UNISWAP_V2_PAIR_ABI =
'[{"inputs":[],"payable":false,"stateMutability":"nonpayable","type":"constructor
"},{"anonymous":false,"inputs":[{"indexed":true,"internalType":"address","nam
e":"owner","type":"address"},{"indexed":true,"internalType":"address","name":
"spender","type":"address"},{"indexed":false,"internalType":"uint256","name":
"value","type":"uint256"}],"name":"Approval","type":"event"},{"anonymous":f
alse,"inputs":[{"indexed":true,"internalType":"address","name":"sender","type":
"address"},{"indexed":false,"internalType":"uint256","name":"amount0","type"
:"uint256"},{"indexed":false,"internalType":"uint256","name":"amount1","type
":"uint256"},{"indexed":true,"internalType":"address","name":"to","type":"addr
ess"}],"name":"Burn","type":"event"},{"anonymous":false,"inputs":[{"indexed"
:true,"internalType":"address","name":"sender","type":"address"},{"indexed":fa
lse,"internalType":"uint256","name":"amount0","type":"uint256"},{"indexed":f
alse,"internalType":"uint256","name":"amount1","type":"uint256"}],"name":"M
int","type":"event"},{"anonymous":false,"inputs":[{"indexed":true,"internalTyp
e":"address","name":"sender","type":"address"},{"indexed":false,"internalType"
:"uint256","name":"amount0In","type":"uint256"},{"indexed":false,"internalTy
pe":"uint256","name":"amount1In","type":"uint256"},{"indexed":false,"internal
Type":"uint256","name":"amount0Out","type":"uint256"},{"indexed":false,"int
ernalType":"uint256","name":"amount1Out","type":"uint256"},{"indexed":true,
"internalType":"address","name":"to","type":"address"}],"name":"Swap","type"
```

:"event"},{"anonymous":false,"inputs":[{"indexed":false,"internalType":"uint11
2","name":"reserve0","type":"uint112"},{"indexed":false,"internalType":"uint11
2","name":"reserve1","type":"uint112"}],"name":"Sync","type":"event"},{"anon
ymous":false,"inputs":[{"indexed":true,"internalType":"address","name":"from"
,"type":"address"},{"indexed":true,"internalType":"address","name":"to","type":
"address"},{"indexed":false,"internalType":"uint256","name":"value","type":"ui
nt256"}],"name":"Transfer","type":"event"},{"constant":true,"inputs":[],"name"
:"DOMAIN_SEPARATOR","outputs":[{"internalType":"bytes32","name":"","t
ype":"bytes32"}],"payable":false,"stateMutability":"view","type":"function"},{"
constant":true,"inputs":[],"name":"MINIMUM_LIQUIDITY","outputs":[{"inter
nalType":"uint256","name":"","type":"uint256"}],"payable":false,"stateMutabili
ty":"view","type":"function"},{"constant":true,"inputs":[],"name":"PERMIT_T
YPEHASH","outputs":[{"internalType":"bytes32","name":"","type":"bytes32"}
],"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"i
nputs":[{"internalType":"address","name":"","type":"address"},{"internalType":
"address","name":"","type":"address"}],"name":"allowance","outputs":[{"intern
alType":"uint256","name":"","type":"uint256"}],"payable":false,"stateMutabilit
y":"view","type":"function"},{"constant":false,"inputs":[{"internalType":"addre
ss","name":"spender","type":"address"},{"internalType":"uint256","name":"val
ue","type":"uint256"}],"name":"approve","outputs":[{"internalType":"bool","na
me":"","type":"bool"}],"payable":false,"stateMutability":"nonpayable","type":"f
unction"},{"constant":true,"inputs":[{"internalType":"address","name":"","type"
:"address"}],"name":"balanceOf","outputs":[{"internalType":"uint256","name":
"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"
},{"constant":false,"inputs":[{"internalType":"address","name":"to","type":"add
ress"}],"name":"burn","outputs":[{"internalType":"uint256","name":"amount0",
"type":"uint256"},{"internalType":"uint256","name":"amount1","type":"uint256
"}],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constan
t":true,"inputs":[],"name":"decimals","outputs":[{"internalType":"uint8","name"
:"","type":"uint8"}],"payable":false,"stateMutability":"view","type":"function"},
{"constant":true,"inputs":[],"name":"factory","outputs":[{"internalType":"addre
ss","name":"","type":"address"}],"payable":false,"stateMutability":"view","type
":"function"},{"constant":true,"inputs":[],"name":"getReserves","outputs":[{"int
ernalType":"uint112","name":"_reserve0","type":"uint112"},{"internalType":"ui
nt112","name":"_reserve1","type":"uint112"},{"internalType":"uint32","name":
"_blockTimestampLast","type":"uint32"}],"payable":false,"stateMutability":"vie
w","type":"function"},{"constant":false,"inputs":[{"internalType":"address","na

me":"_token0","type":"address"},{"internalType":"address","name":"_token1","type":"address"}],"name":"initialize","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":true,"inputs":[],"name":"kLast","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":false,"inputs":[{"internalType":"address","name":"to","type":"address"}],"name":"mint","outputs":[{"internalType":"uint256","name":"liquidity","type":"uint256"}],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":true,"inputs":[],"name":"name","outputs":[{"internalType":"string","name":"","type":"string"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[{"internalType":"address","name":"","type":"address"}],"name":"nonces","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":false,"inputs":[{"internalType":"address","name":"owner","type":"address"},{"internalType":"address","name":"spender","type":"address"},{"internalType":"uint256","name":"value","type":"uint256"},{"internalType":"uint256","name":"deadline","type":"uint256"},{"internalType":"uint8","name":"v","type":"uint8"},{"internalType":"bytes32","name":"r","type":"bytes32"},{"internalType":"bytes32","name":"s","type":"bytes32"}],"name":"permit","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":true,"inputs":[],"name":"price0CumulativeLast","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[],"name":"price1CumulativeLast","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":false,"inputs":[{"internalType":"address","name":"to","type":"address"}],"name":"skim","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":false,"inputs":[{"internalType":"uint256","name":"amount0Out","type":"uint256"},{"internalType":"uint256","name":"amount1Out","type":"uint256"},{"internalType":"address","name":"to","type":"address"},{"internalType":"bytes","name":"data","type":"bytes"}],"name":"swap","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":true,"inputs":[],"name":"symbol","outputs":[{"internalType":"string","name":"","type":"string"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":false,"inputs":[],"name":"sync","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":true,"inputs":[],"name":"token0","outputs":[{"internalType":"address","name":"","type":"address"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":

true,"inputs":[],"name":"token1","outputs":[{"internalType":"address","name":"
","type":"address"}],"payable":false,"stateMutability":"view","type":"function"}
,{"constant":true,"inputs":[],"name":"totalSupply","outputs":[{"internalType":"u
int256","name":"","type":"uint256"}],"payable":false,"stateMutability":"view","
type":"function"},{"constant":false,"inputs":[{"internalType":"address","name":
"to","type":"address"},{"internalType":"uint256","name":"value","type":"uint25
6"}],"name":"transfer","outputs":[{"internalType":"bool","name":"","type":"boo
l"}],"payable":false,"stateMutability":"nonpayable","type":"function"},{"consta
nt":false,"inputs":[{"internalType":"address","name":"from","type":"address"},{
"internalType":"address","name":"to","type":"address"},{"internalType":"uint25
6","name":"value","type":"uint256"}],"name":"transferFrom","outputs":[{"inter
nalType":"bool","name":"","type":"bool"}],"payable":false,"stateMutability":"n
onpayable","type":"function"}]'

UNISWAP_V2_FACTORY_ABI =
'[{"inputs":[{"internalType":"address","name":"_feeToSetter","type":"address"}
],"payable":false,"stateMutability":"nonpayable","type":"constructor"},{"anony
mous":false,"inputs":[{"indexed":true,"internalType":"address","name":"token0
","type":"address"},{"indexed":true,"internalType":"address","name":"token1","
type":"address"},{"indexed":false,"internalType":"address","name":"pair","type
":"address"},{"indexed":false,"internalType":"uint256","name":"","type":"uint2
56"}],"name":"PairCreated","type":"event"},{"constant":true,"inputs":[{"interna
lType":"uint256","name":"","type":"uint256"}],"name":"allPairs","outputs":[{"i
nternalType":"address","name":"","type":"address"}],"payable":false,"stateMuta
bility":"view","type":"function"},{"constant":true,"inputs":[],"name":"allPairsL
ength","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"pay
able":false,"stateMutability":"view","type":"function"},{"constant":false,"inputs
":[{"internalType":"address","name":"tokenA","type":"address"},{"internalTyp
e":"address","name":"tokenB","type":"address"}],"name":"createPair","outputs"
:[{"internalType":"address","name":"pair","type":"address"}],"payable":false,"st
ateMutability":"nonpayable","type":"function"},{"constant":true,"inputs":[],"na
me":"feeTo","outputs":[{"internalType":"address","name":"","type":"address"}]
,"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"in
puts":[],"name":"feeToSetter","outputs":[{"internalType":"address","name":"","
type":"address"}],"payable":false,"stateMutability":"view","type":"function"},{"
constant":true,"inputs":[{"internalType":"address","name":"","type":"address"},
{"internalType":"address","name":"","type":"address"}],"name":"getPair","outp
uts":[{"internalType":"address","name":"","type":"address"}],"payable":false,"st

ateMutability":"view","type":"function"},{"constant":false,"inputs":[{"internalType":"address","name":"_feeTo","type":"address"}],"name":"setFeeTo","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":false,"inputs":[{"internalType":"address","name":"_feeToSetter","type":"address"}],"name":"setFeeToSetter","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"}]'

UNISWAP_V2_ROUTER2_ABI =
'[{"inputs":[{"internalType":"address","name":"_factory","type":"address"},{"internalType":"address","name":"_WETH","type":"address"}],"stateMutability":"nonpayable","type":"constructor"},{"inputs":[],"name":"WETH","outputs":[{"internalType":"address","name":"","type":"address"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"tokenA","type":"address"},{"internalType":"address","name":"tokenB","type":"address"},{"internalType":"uint256","name":"amountADesired","type":"uint256"},{"internalType":"uint256","name":"amountBDesired","type":"uint256"},{"internalType":"uint256","name":"amountAMin","type":"uint256"},{"internalType":"uint256","name":"amountBMin","type":"uint256"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"addLiquidity","outputs":[{"internalType":"uint256","name":"amountA","type":"uint256"},{"internalType":"uint256","name":"amountB","type":"uint256"},{"internalType":"uint256","name":"liquidity","type":"uint256"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"token","type":"address"},{"internalType":"uint256","name":"amountTokenDesired","type":"uint256"},{"internalType":"uint256","name":"amountTokenMin","type":"uint256"},{"internalType":"uint256","name":"amountETHMin","type":"uint256"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"addLiquidityETH","outputs":[{"internalType":"uint256","name":"amountToken","type":"uint256"},{"internalType":"uint256","name":"amountETH","type":"uint256"},{"internalType":"uint256","name":"liquidity","type":"uint256"}],"stateMutability":"payable","type":"function"},{"inputs":[],"name":"factory","outputs":[{"internalType":"address","name":"","type":"address"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"uint256","name":"amountOut","type":"uint256"},{"internalType":"uint256","name":"reserveIn","type":"uint256"},{"internalType":"uint256","name":"reserveOut","type":"uint256"}],"name":"getAmountIn","outputs":[{"internalType":"uint256","name":"amountIn","type":"uint256"}],"stateMutability":"pure","type":"function"},{"inputs":[{"internalType":"

uint256","name":"amountIn","type":"uint256"},{"internalType":"uint256","nam
e":"reserveIn","type":"uint256"},{"internalType":"uint256","name":"reserveOut
","type":"uint256"}],"name":"getAmountOut","outputs":[{"internalType":"uint2
56","name":"amountOut","type":"uint256"}],"stateMutability":"pure","type":"fu
nction"},{"inputs":[{"internalType":"uint256","name":"amountOut","type":"uin
t256"},{"internalType":"address[]","name":"path","type":"address[]"}],"name":
"getAmountsIn","outputs":[{"internalType":"uint256[]","name":"amounts","typ
e":"uint256[]"}],"stateMutability":"view","type":"function"},{"inputs":[{"intern
alType":"uint256","name":"amountIn","type":"uint256"},{"internalType":"addr
ess[]","name":"path","type":"address[]"}],"name":"getAmountsOut","outputs":[
{"internalType":"uint256[]","name":"amounts","type":"uint256[]"}],"stateMuta
bility":"view","type":"function"},{"inputs":[{"internalType":"uint256","name":"
amountA","type":"uint256"},{"internalType":"uint256","name":"reserveA","typ
e":"uint256"},{"internalType":"uint256","name":"reserveB","type":"uint256"}],
"name":"quote","outputs":[{"internalType":"uint256","name":"amountB","type"
:"uint256"}],"stateMutability":"pure","type":"function"},{"inputs":[{"internalTy
pe":"address","name":"tokenA","type":"address"},{"internalType":"address","n
ame":"tokenB","type":"address"},{"internalType":"uint256","name":"liquidity",
"type":"uint256"},{"internalType":"uint256","name":"amountAMin","type":"ui
nt256"},{"internalType":"uint256","name":"amountBMin","type":"uint256"},{"
internalType":"address","name":"to","type":"address"},{"internalType":"uint25
6","name":"deadline","type":"uint256"}],"name":"removeLiquidity","outputs":[
{"internalType":"uint256","name":"amountA","type":"uint256"},{"internalType
":"uint256","name":"amountB","type":"uint256"}],"stateMutability":"nonpayabl
e","type":"function"},{"inputs":[{"internalType":"address","name":"token","typ
e":"address"},{"internalType":"uint256","name":"liquidity","type":"uint256"},{
"internalType":"uint256","name":"amountTokenMin","type":"uint256"},{"inter
nalType":"uint256","name":"amountETHMin","type":"uint256"},{"internalTyp
e":"address","name":"to","type":"address"},{"internalType":"uint256","name":"
deadline","type":"uint256"}],"name":"removeLiquidityETH","outputs":[{"intern
alType":"uint256","name":"amountToken","type":"uint256"},{"internalType":"
uint256","name":"amountETH","type":"uint256"}],"stateMutability":"nonpayab
le","type":"function"},{"inputs":[{"internalType":"address","name":"token","ty
pe":"address"},{"internalType":"uint256","name":"liquidity","type":"uint256"},
{"internalType":"uint256","name":"amountTokenMin","type":"uint256"},{"inte
rnalType":"uint256","name":"amountETHMin","type":"uint256"},{"internalTyp
e":"address","name":"to","type":"address"},{"internalType":"uint256","name":"

deadline","type":"uint256"}],"name":"removeLiquidityETHSupportingFeeOnTr
ansferTokens","outputs":[{"internalType":"uint256","name":"amountETH","typ
e":"uint256"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"i
nternalType":"address","name":"token","type":"address"},{"internalType":"uint
256","name":"liquidity","type":"uint256"},{"internalType":"uint256","name":"a
mountTokenMin","type":"uint256"},{"internalType":"uint256","name":"amount
ETHMin","type":"uint256"},{"internalType":"address","name":"to","type":"add
ress"},{"internalType":"uint256","name":"deadline","type":"uint256"},{"interna
lType":"bool","name":"approveMax","type":"bool"},{"internalType":"uint8","n
ame":"v","type":"uint8"},{"internalType":"bytes32","name":"r","type":"bytes32
"},{"internalType":"bytes32","name":"s","type":"bytes32"}],"name":"removeLi
quidityETHWithPermit","outputs":[{"internalType":"uint256","name":"amount
Token","type":"uint256"},{"internalType":"uint256","name":"amountETH","ty
pe":"uint256"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"
internalType":"address","name":"token","type":"address"},{"internalType":"uin
t256","name":"liquidity","type":"uint256"},{"internalType":"uint256","name":"
amountTokenMin","type":"uint256"},{"internalType":"uint256","name":"amou
ntETHMin","type":"uint256"},{"internalType":"address","name":"to","type":"a
ddress"},{"internalType":"uint256","name":"deadline","type":"uint256"},{"inter
nalType":"bool","name":"approveMax","type":"bool"},{"internalType":"uint8",
"name":"v","type":"uint8"},{"internalType":"bytes32","name":"r","type":"bytes
32"},{"internalType":"bytes32","name":"s","type":"bytes32"}],"name":"remove
LiquidityETHWithPermitSupportingFeeOnTransferTokens","outputs":[{"intern
alType":"uint256","name":"amountETH","type":"uint256"}],"stateMutability":"
nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"t
okenA","type":"address"},{"internalType":"address","name":"tokenB","type":"a
ddress"},{"internalType":"uint256","name":"liquidity","type":"uint256"},{"inter
nalType":"uint256","name":"amountAMin","type":"uint256"},{"internalType":"
uint256","name":"amountBMin","type":"uint256"},{"internalType":"address","
name":"to","type":"address"},{"internalType":"uint256","name":"deadline","typ
e":"uint256"},{"internalType":"bool","name":"approveMax","type":"bool"},{"i
nternalType":"uint8","name":"v","type":"uint8"},{"internalType":"bytes32","na
me":"r","type":"bytes32"},{"internalType":"bytes32","name":"s","type":"bytes3
2"}],"name":"removeLiquidityWithPermit","outputs":[{"internalType":"uint256
","name":"amountA","type":"uint256"},{"internalType":"uint256","name":"amo
untB","type":"uint256"}],"stateMutability":"nonpayable","type":"function"},{"i
nputs":[{"internalType":"uint256","name":"amountOut","type":"uint256"},{"int

ernalType":"address[]","name":"path","type":"address[]"},{"internalType":"add
ress","name":"to","type":"address"},{"internalType":"uint256","name":"deadlin
e","type":"uint256"}],"name":"swapETHForExactTokens","outputs":[{"internal
Type":"uint256[]","name":"amounts","type":"uint256[]"}],"stateMutability":"pa
yable","type":"function"},{"inputs":[{"internalType":"uint256","name":"amount
OutMin","type":"uint256"},{"internalType":"address[]","name":"path","type":"a
ddress[]"},{"internalType":"address","name":"to","type":"address"},{"internalT
ype":"uint256","name":"deadline","type":"uint256"}],"name":"swapExactETHF
orTokens","outputs":[{"internalType":"uint256[]","name":"amounts","type":"ui
nt256[]"}],"stateMutability":"payable","type":"function"},{"inputs":[{"internalT
ype":"uint256","name":"amountOutMin","type":"uint256"},{"internalType":"ad
dress[]","name":"path","type":"address[]"},{"internalType":"address","name":"t
o","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint2
56"}],"name":"swapExactETHForTokensSupportingFeeOnTransferTokens","ou
tputs":[],"stateMutability":"payable","type":"function"},{"inputs":[{"internalTy
pe":"uint256","name":"amountIn","type":"uint256"},{"internalType":"uint256",
"name":"amountOutMin","type":"uint256"},{"internalType":"address[]","name"
:"path","type":"address[]"},{"internalType":"address","name":"to","type":"addr
ess"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"
swapExactTokensForETH","outputs":[{"internalType":"uint256[]","name":"am
ounts","type":"uint256[]"}],"stateMutability":"nonpayable","type":"function"},{
"inputs":[{"internalType":"uint256","name":"amountIn","type":"uint256"},{"int
ernalType":"uint256","name":"amountOutMin","type":"uint256"},{"internalTyp
e":"address[]","name":"path","type":"address[]"},{"internalType":"address","na
me":"to","type":"address"},{"internalType":"uint256","name":"deadline","type"
:"uint256"}],"name":"swapExactTokensForETHSupportingFeeOnTransferToke
ns","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"
internalType":"uint256","name":"amountIn","type":"uint256"},{"internalType":
"uint256","name":"amountOutMin","type":"uint256"},{"internalType":"address
[]","name":"path","type":"address[]"},{"internalType":"address","name":"to","t
ype":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}
],"name":"swapExactTokensForTokens","outputs":[{"internalType":"uint256[]",
"name":"amounts","type":"uint256[]"}],"stateMutability":"nonpayable","type":"
function"},{"inputs":[{"internalType":"uint256","name":"amountIn","type":"uin
t256"},{"internalType":"uint256","name":"amountOutMin","type":"uint256"},{
"internalType":"address[]","name":"path","type":"address[]"},{"internalType":"
address","name":"to","type":"address"},{"internalType":"uint256","name":"dea

dline","type":"uint256"}],"name":"swapExactTokensForTokensSupportingFeeOnTransferTokens","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"uint256","name":"amountOut","type":"uint256"},{"internalType":"uint256","name":"amountInMax","type":"uint256"},{"internalType":"address[]","name":"path","type":"address[]"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"swapTokensForExactETH","outputs":[{"internalType":"uint256[]","name":"amounts","type":"uint256[]"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"uint256","name":"amountOut","type":"uint256"},{"internalType":"uint256","name":"amountInMax","type":"uint256"},{"internalType":"address[]","name":"path","type":"address[]"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"deadline","type":"uint256"}],"name":"swapTokensForExactTokens","outputs":[{"internalType":"uint256[]","name":"amounts","type":"uint256[]"}],"stateMutability":"nonpayable","type":"function"},{"stateMutability":"payable","type":"receive"}]'

```
V2_UNISWAP_ROUTER_ADDRESSES = {
    'eth': '0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D',
    'arbi': '0x4752ba5dbc23f44d87826276bf6fd6b1c372ad24',
}

V2_UNISWAP_FACTORY_ADDRESSES = {
    'eth': '0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f',
    'arbi': '0xf1D7CC64Fb4452F05c498126312eBE29f30Fbcf9',
}

V2_UNISWAP_INIT_CODE_HASH = '0x96e8ac4277198ff8b6f785478aa9a39f403cb768dd02cbee326c3e7da348845f'


class UniswapFactory:
```

```
    def __init__(self, account: Account):
        self.account = account

    @property
    def contract(self):
        return self.account.get_contract(
            V2_UNISWAP_FACTORY_ADDRESSES[self.account.chain.name],
UNISWAP_V2_FACTORY_ABI
        )

    @retry_on_error()
    async def all_pairs_length(self):
        return await self.contract.functions.allPairsLength().call()

    @retry_on_error()
    async def get_pool_by_key(self, key: int):
        return await self.contract.functions.allPairs(key).call()

    @retry_on_error()
    async def get_pair(self, address1: ChecksumAddress, address2:
ChecksumAddress):
        return await self.contract.functions.getPair(address1, address2).call()



class UniswapPair:
    def __init__(self, account: Account, t0: Token, t1: Token):
        self.account = account
```

```
        self.t0 = t0
        self.t1 = t1

    @property
    def pool_address(self):
        """
        generates the address of a pair of two tokens,
        as in the factory contract in the create pair function
        """
        token_addresses = sorted([address.lower() for address in [self.t0.address,
self.t1.address]])

        salt = keccak(
            packed.encode_packed(
                ["address", "address"],
                [*token_addresses],
            )
        )
        return to_checksum_address(
            keccak(
                HexBytes(0xFF)
                +
HexBytes(V2_UNISWAP_FACTORY_ADDRESSES[self.account.chain.name]
)
                + salt
                + HexBytes(V2_UNISWAP_INIT_CODE_HASH)
            )[-20:]
        )
```

```
@property
def contract(self):
    return self.account.get_contract(
        self.pool_address, UNISWAP_V2_PAIR_ABI
    )

@retry_on_error()
async def get_reserves(self) -> Tuple[int, int, int]:

    return await self.contract.functions.getReserves().call()

@retry_on_error()
async def t0(self):
    return await self.contract.functions.token0().call()

@retry_on_error()
async def t1(self):
    return await self.contract.functions.token1().call()

@staticmethod
def calculate_amount_out(amount_in: int, reserve_in: int, reserve_out: int):
    amount_in_with_fee = amount_in * 9975
    numerator = amount_in_with_fee * reserve_out
    denominator = reserve_in * 10000 + amount_in_with_fee
    amount_out = numerator // denominator
    return amount_out
```

```
class Uniswap:
    def __init__(self, account: Account):
        self.account = account

    @property
    def contract(self):
        """return router contract"""
        return self.account.get_contract(
            V2_UNISWAP_ROUTER_ADDRESSES[self.account.chain.name],
UNISWAP_V2_ROUTER2_ABI
        )

    @staticmethod
    def deadline():
        return int(time.time() + 10 * 60)

    ##########################################################
    # read funcs ##########################################
    ##########################################################

    @retry_on_error()
    async def get_amounts_out_from_path(self, amount_in: int, path:
List[ChecksumAddress]):
        data = await self.contract.functions.getAmountsOut(amount_in, path).call()
        return data[-1]
```

```
    ########################################################
    # write funcs ############################################
    ########################################################

    # liquidity funcs

    def add_liquidity(self, token_address: ChecksumAddress, amount_token: int, amount_eth: int) -> HexStr:
        """
        solidity code:~
        function addLiquidityETH(
            address token,
            uint amountTokenDesired,
            uint amountTokenMin,
            uint amountETHMin,
            address to,
            uint deadline)
        """
        return self.contract.encode_abi('addLiquidityETH',
                                args=[
                                    token_address,
                                    amount_token,
                                    0,
                                    0,
                                    self.account.address,
                                    self.deadline()
                                ])
```

```
    def remove_liquidity(self, token_address: ChecksumAddress, liquidity_amount: int) -> HexStr:
        """
        solidity code:~
        function removeLiquidityETH(
          address token,
          uint liquidity,
          uint amountTokenMin,
          uint amountETHMin,
          address to,
          uint deadline)
        """
        return self.contract.encode_abi('removeLiquidityETH',
                                        args=[
                                            token_address,
                                            liquidity_amount,
                                            0,
                                            0,
                                            self.account.address,
                                            self.deadline()
                                        ])

    # swap funcs
    def swap_eth_for_tokens(self, amount_out_min: int, path: List[ChecksumAddress]) -> HexStr:

        return self.contract.encode_abi('swapExactETHForTokens',
                                        args=[
```

```
                    amount_out_min,
                    path,
                    self.account.address,
                    self.deadline()
                ])

def swap_tokens_for_eth(self, amount_in: int, amount_out_min: int, path: List[ChecksumAddress]) -> HexStr:

    return self.contract.encode_abi('swapExactTokensForETH',
                args=[
                    amount_in,
                    amount_out_min,
                    path,
                    self.account.address,
                    self.deadline()
                ])

def swap_tokens_for_tokens(self, amount_in: int, amount_out_min: int, path: List[ChecksumAddress]) -> HexStr:

    return self.contract.encode_abi('swapExactTokensForTokens',
                args=[
                    amount_in,
                    amount_out_min,
                    path,
                    self.account.address,
                    self.deadline()
```

```
                ])

    async def swap(self, from_token: Token, to_token: Token, amount: Amount, additional_to_token: Optional[Token] = None):
        base_path = [from_token, to_token] if not additional_to_token else [from_token, to_token, additional_to_token]

        target_token = base_path[-1]

        path = [token.address for token in base_path]

        amount_out_min = await self.get_amounts_out_from_path(amount.wei_amount, path)

        if from_token.name == 'WETH':
            data = self.swap_eth_for_tokens(amount_out_min, path)
            value = amount.wei_amount
            need_approve = False

        elif target_token.name == 'WETH':
            data = self.swap_tokens_for_eth(amount.wei_amount, amount_out_min, path)
            value = 0
            need_approve = True
        else:
            data = self.swap_tokens_for_tokens(amount.wei_amount, amount_out_min, path)
            value = 0
            need_approve = True
```

```
        if need_approve:
            if not await self.account.approve_token(
                    self.contract.address,
                    amount,
                    from_token,
            ):
                return
        tx = await self.account.build_tx(
            contract_address=self.contract.address,
            data=data,
            value=value,
        )
        result = await self.account.sign_and_send(tx)
        if result.status:
            logger.success(f'{self.account.acc_info} - succesfully swapped {amount.amount} {from_token.name} for {base_path[-1].name} '
                           f'{self.account.explorer(result.tx_hash)}')
        else:
            logger.error(f'{self.account.acc_info} - tx swap not succesfully {self.account.explorer(result.tx_hash)}')



    async def liquidity(self, token: Token, amount: Optional[Amount] = None, amount_eth: Optional[Amount] = None, is_add=True):
        if is_add:
            await self.account.approve_token(self.contract.address, amount, token)
```

```
            await self.account.approve_token(self.contract.address, amount, WETH[self.account.chain.name])
            data = self.add_liquidity(
                token.address,
                amount.wei_amount,
                amount_eth.wei_amount
            )
            value = amount_eth.wei_amount
            text = f"succesfully added {amount.amount} {token.name} and {amount_eth.amount} eth in liquidity"
        else:
            address = UniswapPair(self.account, token, WETH[self.account.chain.name]).pool_address

            liq_token = Token(name=f'UNI PAIR {token.name}-WETH', _address=address, decimals=18)

            liquidity_amount = await self.account.get_balance(liq_token)

            await self.account.approve_token(self.contract.address, liquidity_amount, liq_token)

            data = self.remove_liquidity(
                token.address, liquidity_amount.wei_amount
            )
            value = 0
            text = f"succesfully removed liquidity"

        tx = await self.account.build_tx(
```

```
            contract_address=self.contract.address,
            data=data,
            value=value,
        )
        result = await self.account.sign_and_send(tx)
        if result.status:
            logger.success(f'{self.account.acc_info} - {text}
{self.account.explorer(result.tx_hash)}')
        else:
            logger.error(f'{self.account.acc_info} - tx not succesfully
{self.account.explorer(result.tx_hash)}')


async def test_swap1():
    a = Account(key=key, chain=Chains.arbi)
    u = Uniswap(a)

    await u.swap(
        WETH[a.chain.name], USDT_ARB, Amount(to_wei(0.0005, 'ether'),
0.0005)
    )
    balance_usdt = await a.get_balance(USDT_ARB)
    await u.swap(USDT_ARB, WETH[a.chain.name], balance_usdt)

async def test_liq(is_add=True):
    a = Account(key=key, chain=Chains.arbi)
    u = Uniswap(a)
```

```
    if is_add:
        await u.liquidity(
            THESIS,
            Amount(to_wei(25000, 'ether'), 25000),
            Amount(to_wei(0.001, 'ether'), 0.001),
        )
    else:
        await u.liquidity(
            THESIS,
            is_add=False
        )


async def test_swap2():
    a = Account(key=key, chain=Chains.arbi)
    u = Uniswap(a)


    await u.swap(
        WETH[a.chain.name], THESIS, Amount(to_wei(0.0005, 'ether'), 0.0005)
    )
    await u.swap(THESIS, WETH[a.chain.name], Amount(to_wei(1000, 'ether'),
1000))
    await u.swap(THESIS, WETH[a.chain.name], Amount(to_wei(1000, 'ether'),
1000), additional_to_token=USDT_ARB)
    balance_usdt = await a.get_balance(USDT_ARB)
    await u.swap(USDT_ARB, WETH[a.chain.name], balance_usdt,
additional_to_token=THESIS)


if __name__ == '__main__':
```

```
    asyncio.set_event_loop_policy(asyncio.WindowsSelectorEventLoopPolicy())
    asyncio.run(test_liq())
```

Програмний код реалізації селектору для вибору режиму роботи

```
import asyncio
import sys

import questionary
from questionary import Choice

from projects.bank_contract import test_deposits, test_withdrawals, test_all
from projects.uniswap_v2 import test_liq, test_swap1, test_swap2
from projects.test_w3 import test


async def uni_task():
    task = await questionary.select(
        "choice uniswap task...",
        choices=[
            Choice("1) add liqiudity", "add"),
            Choice("2) remove", "remove"),
            Choice("3) swap test1", test_swap1),
            Choice("4) swap test2", test_swap2),
            Choice(f" exit", 'e'),
        ],
        qmark="",
        pointer="⟹ ",
    ).ask_async()
```

```
    if task == 'add':
        return await test_liq(True)
    if task == 'remove':
        return await  test_liq(False)
    if task == 'e':
        return
    else:
        return await task()



async def w3_task():
    task = await  questionary.select(
        "choice w3 task...",
        choices=[
            Choice("1) test w3", test),
            Choice(f" exit", 'e'),
        ],
        qmark="",
        pointer="⟹ ",
    ).ask_async()
    if task == 'e':
        return
    else:
        return await task()



async def bank_test():
```

```
    task = await  questionary.select(
        "choice w3 task...",
        choices=[
            Choice("1) test_deposits", test_deposits),
            Choice("2) test_withdrawals", test_withdrawals),
            Choice("3) test all", test_all),
            Choice(f" exit", 'e'),
        ],
        qmark="",
        pointer="⟹ ",
    ).ask_async()
    if task == 'e':
        return
    else:
        return await task()


async def main():
    while True:
        c = await questionary.select(
            "Select modules to work with...",
            choices=[
                Choice("1) test w3", w3_task),
                Choice("2) test uniswap", uni_task),
                Choice("3) test bank contract", bank_test),
                Choice(f" exit", 'e'),
            ],
```

```
            qmark="",
            pointer="⟹ ",
        ).ask_async()
        if c == 'e':
            sys.exit()
        else:
            await c()


if __name__ == '__main__':
    asyncio.set_event_loop_policy(asyncio.WindowsSelectorEventLoopPolicy())
    asyncio.run(main())
```