КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

ЗАСОБИ ДЛЯ МОНІТОРИНГУ ТА АВТОМАТИЧНОГО ОБМІНУ ВЗАЄМОЗАМІННИХ ЦИФРОВИХ АКТИВІВ НА ДЕЦЕНТРАЛІЗОВАНИХ ПЛАТФОРМАХ

Виконав студент 4-го курсу
Георгій САДЧІКОВ _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Тетяна КАРНАУХ _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теоретичної кібернетики

«____»_____2023 р., протокол № ____

Завідувач кафедри

Юрій КРАК _____

(підпис)

Київ - 2023

РЕФЕРАТ

Обсяг роботи 59 сторінок, 12 ілюстрацій, 24 джерела посилань, 2 додатки.

АВТОМАТИЧНИЙ ОБМІН, БЛОКЧЕЙН, МОНІТОРИНГ КУРСІВ, ЦИФРОВИЙ АКТИВ, API 1INCH, API HASHFLOW, BINANCE, PYTHON.

Об’єктом роботи є децентралізовані біржі, дослідження можливості використання їх вразливостей та неефективностей ринку в цілому.

Метою роботи є створення програмного продукту, за допомогою якого можна автоматично відслідковувати курси цифрових активів та здійснювати обмін на децентралізованих біржах.

Інструменти, що були використані під час розроблення: інтегроване середовище розробки: Microsoft Visual Studio Code; провайдер інфраструктури блокчейну QuickNode; програмні інтерфейси (API) децентралізованих сервісів обміну цифровими активами Hashflow та 1inch.

Результати роботи: розроблена програма для моніторингу курсів стейблкоїнів на платформах 1inch та Hashflow; розроблена програма для моніторингу курсу та автоматичної купівлі й продажу Binance Coin на платформі Hashflow. Проаналізовано обмеження, суттєві для роботи вищезазначених програмних засобів. Обидві програми були успішно протестовані у реальних умовах, де за допомогою першої з них були помічені неефективності ринку, а за допомогою другої — використані вразливості біржі Hashflow.

Розроблене програмне забезпечення для моніторингу курсів стейблкоїнів може використовуватись як помічник для подальшого здійснення угод з ними, а програма для автоматичного обміну Binance Coin — для проведення угод на купівлю і продаж Binance Coin.

ЗМІСТ

С.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

Application Binary Interface, або ABI — це стандартизований інтерфейс, який визначає, як взаємодіяти з розгорнутим смарт-контрактом на блокчейні. Визначає функції, події та структури даних смарт-контракту, включаючи їх назви, типи та списки аргументів, дозволяє розробникам взаємодіяти зі смарт-контрактом без необхідності знати деталі його реалізації. ABI часто представляють у вигляді JSON-файлу або масиву об'єктів, де кожен об'єкт описує функцію або подію контракту разом з її властивостями.

Application Programming Interface, або API — це набір правил та протоколів, які дозволяють комунікувати між собою різним програмним додаткам без необхідності знати внутрішню реалізацію одне одного. Також API визначає, які функції та операції можуть бути виконані програмним додатком, які дані можуть бути передані, а також формат цих даних.

Avalanche — назва сумісного з Ethereum блокчейну.

Binance Smart Chain — сумісний з Ethereum блокчейн.

BNB або Binance Coin — це криптовалюта, що була випущена командою криптовалютної біржі Binance, а далі мігрована на власний блокчейн Binance Smart Chain.

BSC — скорочена назва блокчейну Binance Smart Chain.

BUSD, TUSD, USDC, USDP, USDT — назви стейблкоїнів, вартість яких дорівнює вартості долару США.

Ethereum — це блокчейн, який дозволяє створювати і виконувати смарт-контракти на своїй базі.

EVM або Ethereum Virtual Machine — віртуальна машина, що виконує код смарт-контрактів на мові програмування Solidity (основній мові програмування Ethereum), перетворюючи його в низку операцій, які виконуються на блокчейні.

Polygon — сумісний з Ethereum блокчейн.

ВСТУП

Оцінка сучасного стану об’єкта розробки. Останнім часом широкого розповсюдження набуває DeFi — нова система фінансових послуг та інструментів, що базується на технології блокчейн [1].

Найпопулярнішими представниками DeFi є децентралізовані біржі для обміну цифрових активів. Зазвичай вони не вимагають реєстрації акаунту, замість цього користувач підключає криптовалютний гаманець до платформи й одразу може розпочати роботу з біржею. Угоди з купівлі й продажу цифрових активів зазвичай виконуються за допомогою смарт-контрактів — довільного коду, розміщеного за певною адресою в блокчейні [2].

Найвідомішою децентралізованою біржею є Uniswap, де користувачі обмінюють активи за допомогою пулу ліквідності — смарт-контракту, що має певні правила встановлення курсу, програмний інтерфейс для здійснення та розрахунку результату обміну, певну кількість цифрових активів, надану постачальниками ліквідності [3]. Є й інші реалізації децентралізованого обміну цифрових активів, зокрема гібридні біржі, де угоди здійснюються на блокчейні, але курси активів і дані, необхідні для обміну, користувач отримує з централізованого джерела — серверу платформи. Прикладом такої біржі може слугувати досить нова платформа Hashflow [4], робота з якою можлива через API.

Також є сервіси, що агрегують інформацію про ліквідність і курси активів з різних децентралізованих бірж. Через агрегатори можна здійснити угоду, що буде виконана частинами одночасно на декількох платформах для обміну. Перевагами сервісів агрегування є відсутність необхідності використання окремих децентралізованих платформ, кращі обмінні курси, єдиний програмний інтерфейс для розробників [5]. Одним з найпопулярнішим агрегатором для децентралізованого обміну цифрових активів є 1inch.

Однак, ринок децентралізованих фінансів ще досить молодий і, як наслідок, не завжди є ефективним: можуть виникати ситуації, коли на різних децентралізованих біржах активи торгуються з різними цінами. Це є наслідком різного попиту й пропозиції на різних платформах, нестачі ліквідності для обміну, наявності малої кількості професійних торгових фірм на ринку.

Актуальність роботи та підстави для її виконання. 29-го вересня 2022-го року біржа Binance замінила стейблкоїни USDC, USDP, TUSD та BUSD одним цифровим активом — BUSD [6]. Це означає, що користувач може заводити на платформу будь-який актив з переліку стейблкоїнів вище і отримувати на баланс BUSD, і навпаки: виводити кошти в USDC, USDP та TUSD у співвідношенні 1:1 до залишку на рахунку в BUSD. Гіпотеза, через яку виникла ідея створення програмного засобу для моніторингу курсів стейблкоїнів, полягає в тому, що хоч для біржі Binance ці активи рівноцінні, їх ціни на децентралізованих платформах можуть відрізнятися.

Платформа Hashflow має перелік особливостей, одна з яких це те, що дані, необхідні для виконання обміну, що отримуються з серверу, дійсні лише обмежений час [7]. Підставою для створення програми, що відслідковує курси й виконує прибуткові обміни, стало припущення про те, що цього часу може бути достатньо для того, щоб виникла ситуація, коли в нас є досі дійсні дані для купівлі за меншою ціною і дані для продажу за трохи вищою ціною.

Створення подібних програмних засобів наочно продемонструє наявність неефективностей ринку та вразливості децентралізованих бірж. До того ж, результатом здійснення угод за допомогою них стане більш ефективний ринок.

Мета й завдання роботи. Мета кваліфікаційної роботи полягає у:

а) створенні консольного додатку, що відслідковує курси декількох стейблкоїнів та розбіжності між ними для подальшого ручного обміну між централізованою біржею Binance, агрегатором 1inch та гібридною біржею Hashflow;

б) створенні консольного додатку, що автоматично виконує прибутковий

обмін BNB, використовуючи вищезгадану особливість гібридної біржі Hashflow;

в) фіксуванні наявних неефективностей ринку цифрових активів, дослідженні вразливостей децентралізованих бірж за допомогою розроблюваних засобів.

Розроблювані засоби мають працювати на платформі Windows та використовувати API агрегатора 1nch та гібридної біржі Hashflow.

Для досягнення цієї мети поставлено такі завдання.

- Визначити особливості API 1inch, Hashflow, офіційно інтегруватися з API Hashflow
- З’ясувати, як працювати з інтерфейсом смарт-контракту Hashflow
- Ознайомитись з бібліотекою Python web3 для керування криптовалютним гаманцем.
- Підібрати решту засобів розроблення та розробити додатки.
- Підібрати напрямки обміну активів, оптимальну кількість активів для обміну.
- За допомогою розроблених додатків зафіксувати неефективності ринку, дослідити можливості використання вразливостей бірж.

Об’єкт, методи й засоби розроблення. Об’єктом роботи є дослідження можливостей використання неефективностей ринку й вразливостей децентралізованих бірж.

Використовувались такі інструментальні засоби, бібліотеки та технології: Microsoft Visual Studio Code, QuickNode API, 1inch Pathfinder API, Hashflow API, модулі Python web3 5.31.3, aiohttp 3.8.3, python-binance 1.0.16, pyTelegramBotAPI 4.8.0, requests, asyncio.

Можливі сфери застосування. Розроблене програмне забезпечення для відстеження курсів стейблкоїнів можна використовувати як інструмент для допомоги користувачу при здійсненні подальших угод з цими активами. Також, програма для автоматичного обміну Binance Coin може бути використана для проведення угод на купівлю і продаж BNB.

1 ЕЛЕМЕНТИ БЛОКЧЕЙН-ТЕХНОЛОГІЇ

Блокчейн — технологія, що забезпечує безпечний обмін даними із збереженням їх у вигляді блоків, які ланцюжкуються за допомогою криптографічних методів. Кожен блок містить інформацію про попередній блок, утворюючи неперервний ланцюжок записів. Ця технологія забезпечує надійність, прозорість та незмінність даних, що дозволяє застосовувати її для різноманітних цілей, включаючи криптовалюту, контракти, логістику та багато іншого [8].

Смарт-контракт — це довільний програмний код, який зберігається за певною адресою у блокчейні і автоматично виконує певні дії при виконанні визначених умов.

EVM-сумісні блокчейни — це блокчейни, які використовують Ethereum Virtual Machine для виконання коду смарт-контрактів [9]. Така сумісність блокчейнів дозволяє розробникам створювати та запускати смарт-контракти, які будуть працювати на різних блокчейнах однаково, забезпечуючи сумісність з існуючим екосистемою Ethereum. Це означає, що розробники можуть використовувати існуючі інструменти, бібліотеки і середовища Ethereum для створення додатків на EVM-сумісних блокчейнах. Наприклад, Binance Smart Chain, Polygon, Avalanche є прикладами EVM-сумісних блокчейнів, які дозволяють виконувати смарт-контракти, зберігати криптовалюту та розробляти децентралізовані додатки на основі EVM.

Клієнт Geth — один з найпопулярніших клієнтів Ethereum, що реалізує повний вузол блокчейну Ethereum. Він надає інтерфейс для спілкування з мережею Ethereum та дозволяє користувачам взаємодіяти з блокчейном, створювати та виконувати смарт-контракти, відправляти транзакції і перевіряти стан мережі.

Консенсус — це процес досягнення згоди щодо транзакцій та стану

блокчейну серед різних вузлів мережі. Це забезпечує надійність, цілісність та безпеку системи, де кожен вузол має узгоджений вид блокчейну.

Proof of Authority, або PoA — алгоритм консенсусу, що використовується в блокчейн-системах, де спеціально обрані вузли, називані авторитетами, мають право генерувати та підтверджувати нові блоки.

Котирування — цінова пропозиція, яка вказує ціну, за якою продавець готовий продати актив, і ціну, за якою покупець готовий придбати актив [10]. Котирування можуть змінюватись у реальному часі відповідно до попиту і пропозиції на ринку.

Криптовалюта — це цифровий або віртуальний актив, який використовує криптографічні методи для забезпечення безпеки та контролю над створенням нових одиниць, а також для здійснення безпечних та анонімних транзакцій.

Криптовалютний гаманець — це програмне або апаратне забезпечення для зберігання і управління криптовалютою. Він містить приватний ключ, що дозволяє власнику гаманця підписувати та автентифікувати транзакції, а також публічний ключ, який використовується для отримання платежів.

Стейблкоїн — вид криптовалюти, який підтримує свою вартість за рахунок резерву в реальних активах або інших механізмів, які допомагають зберігати стабільну ціну [11].

Токен — це цифровий актив, що функціонує на базі блокчейну [12]. Представляє собою цифровий еквівалент, який може бути використаний для передачі, зберігання та обміну значенням в електронній формі.

2 ПРОГРАМНІ ІНТЕРФЕЙСИ 1INCH, HASHFLOW ТА BINANCE

Консольний додаток, що моніторить курси стейблкоїнів, отримує котирування з трьох джерел: API централізованої біржі Binance, API агрегатору децентралізованих бірж 1inch та API гібридної біржі Hashflow. Програма, що моніторить курс BNB та автоматично здійснює обміни звертається лише до API Hashflow. Розглянемо використовувані елементи.

2.1 Отримання котирування через програмний інтерфейс 1inch

Запит до API 1inch для отримання курсів цифрових активів може бути сформований за URL [13]: https://api.1inch.io/v5.0/{chainId}/quote, де **chainId** — ідентифікатор блокчейну. Наприклад, для блокчейну Ethereum це 1.
Цей GET-запит приймає наступні обов'язкові параметри:

- **fromTokenAddress** — адреса вхідного токена (того цифрового активу, що ми обмінюємо).
- **toTokenAddress** — адреса цільового токена (того цифрового активу, що ми отримуємо в результаті обміну).
- **Amount** — кількість вхідного токена, яку потрібно обміняти. Важливо пам’ятати, що токени можуть мати різну кількість знаків після десяткової коми. Зазвичай це 18. API 1inch приймає на вхід кількість токенів помножену на десять у степені кількості знаків після десяткової коми.
  Наприклад, один токен позначається, як 1000000000000000000.

Запит поверне відповідь у форматі JSON. Нас цікавить ключ **toTokenAmount** — кількість цільового токена, яку можна отримати, якщо здійснити такий обмін.

Наприклад, якщо потрібно дізнатися, скільки USDC можна отримати при обміні ста тисяч BUSD у блокчейні BSC з ідентифікатором 56, то потрібно

відправити GET запит з наступними параметрами:

- **fromTokenAddress**, що буде дорівнювати 0xe9e7cea3dedca5984780bafc599bd69add087d56 (адреса смарт-контракту токену USDC у блокчейні BSC [14])
- **toTokenAddress**, що буде дорівнювати 0x8ac76a51cc950d9822d68b83fe1ad97b32cd580d (адреса смарт-контракту токену BUSD у блокчейні BSC [15])
- **amount** із значенням 100000000000000000000000, що відповідає 100000 USDC.

Нижче наведено код програми, що відправляє запит та виводить відповідь.

```
inch_endpoint =
“https://api.1inch.io/v5.0/56/quote?fromTokenAddress=0xe9e7cea3dedca59847
80bafc599bd69add087d56&toTokenAddress=0x8ac76a51cc950d9822d68b83fe1ad97b3
2cd580d&amount=100000000000000000000000”

r = requests.get(url = inch_ endpoint)
print(“Answer:”, r.json(), “\n”)
print(“toTokenAmount:”, r.json()[‘toTokenAmount’])
```

Відповідь містить детальну інформацію про курс обміну, кількість активів, протоколи, використані для оптимального котирування, та інші параметри, необхідні для виконання угоди (див. рис. 1).

Рисунок 1— Тіло відповіді API на запит та вихідна кількість токенів

Значення 9999248815817290619926 0 ключа **toTokenAmount** відповідає вихідній кількості USDC приблизно у 99992.48 токенів.

Обмеження частоти запитів до 1inch є змінною величиною. На практиці я з ним стикався дуже рідко, хоча і надсилав одночасно декілька запитів кожні десять секунд протягом кількох днів.

2.2 Використання програмного інтерфейсу Hashflow

2.2.1 Запит цінової пропозиції

API Hashflow містить багато доступних запитів [16], серед яких у рамках цієї роботи нас цікавлять запити Request for Quote — запити цінової пропозиції. Запит до API для отримання курсів цифрових активів формується за URL https://api.hashflow.com/taker/v2/rfq. Тіло запиту є JSON-об'єктом з полями:

- **networkId** — ідентифікатор блокчейну;
- **source** — джерело запиту. Приймає значення ”hashflow” для неавтентифікованих користувачів, ”api” — для автентифікованих;
- **baseToken** — адреса вхідного токена;
- **quoteToken** — адреса цільового токена;
- **baseTokenAmount** або **quoteTokenAmount** — на вибір користувача кількість вхідного токена, яку потрібно обміняти, або кількість цільового токена, яку користувач хоче отримати;
- **trader** — адреса гаманця користувача, що хоче здійснити обмін;
- **rfqType** — вказує на те, яка сторона надсилає запит: звичайний користувач чи постачальник ліквідності біржі.

У відповідь на запит із серверу повернеться котирування у форматі JSON, багато ключів якого дублюють відповідні ключі тіла запиту, тому нижче я зазначу лише

ті, що дійсно важливі:

- **signature** — підпис отриманого котирування приватним ключем з серверу, що буде перевірятися під час здійснення обміну;
- **hftTradingRewards** — кількість токенів HFT, що буде слугувати нагородою користувача за здійснення цього обміну;
- **quoteData** — вкладений JSON-об’єкт, що містить ключі **baseTokenAmount** та **quoteTokenAmount**, **quoteExpiry** — час, після якого котирування буде вже недійсним (під час обміну транзакція не завершиться вдало); **nonce** — параметр, що перевіряється під час здійснення обміну, щоб запобігти повторному виконанню користувачем того ж самого котирування.

2.2.2 Автентифікація

Спочатку я використовував публічний API, вказуючи в тілі запиту параметр **source** зі значенням «hashflow». Нажаль, API Hashflow надає досить обмежену кількість запитів у період часу, що не вистачило би для реалізації програмних продуктів у рамках цієї роботи. Тому було прийняте рішення офіційно інтегруватися з програмним інтерфейсом платформи, попросивши команду біржі надати автентифікаційний заголовок (див. рис. 2).


We're excited to integrate you as an api-wallet on Hashflow.

Your authentication key is:

T [redacted] P

When connecting to the API, you'll need to:
1. Set the field "source" to "api" and "wallet" to your evm wallet in the REST body/query
2. Set the "Authorization" header to your auth key

If you have any questions, don't hesitate to reach out!

Рисунок 2 — Відповідь команди Hashflow із автентифікаційним заголовком і інструкцією з його використання

Для використання API з підвищеними лімітами на кількість запитів необхідно відправляти заголовок «Authorization» із наданим значенням, а також у полі **trader** використовувати надану команді адресу свого криптовалютного гаманця [17].

2.2.3 Приклад отримання котирування

Наприклад, якщо потрібно дізнатися, скільки USDC можна отримати при обміні десяти тисяч BUSD у блокчейні Binance Smart Chain, то потрібно відправити POST запит з наступними параметрами:

- **networkId**, що дорівнює 56 **(**ідентифікатор блокчейну BSC);
- **source**, що приймає значення ”api”;
- **baseToken** зі значенням ”0xe9e7cea3dedca5984780bafc599bd69add087d56” (адреса смарт-контракту токену USDC у блокчейні BSC);
- **quoteToken** зі значенням ”0x8ac76a51cc950d9822d68b83fe1ad97b32cd580d” (адреса смарт-контракту токену BUSD у блокчейні BSC);
- **baseTokenAmount**, що буде приймати значення ”10000000000000000000” (десять тисяч помножити на десять у степені 18 — кількості знаків після коми для токену USDC);
- **trader** зі значенням адреси гаманця, котру я відправив команді Hashflow;
- **rfqType** зі значенням 0, що вказує на те, що обмін буде здійснювати користувач платформи.

Фрагмент коду

```
urlHashflow = ‘https://api.hashflow.com/taker/v2/rfq’
headersHashflow = {“Authorization”: “*Автентифікаційний заголовок*”}
```

```
dataHashflowBusdUsdcBSC =
{"networkId":56,"source":"api","baseToken":"0xe9e7cea3dedca5984780bafc599
bd69add087d56","quoteToken":"0x8ac76a51cc950d9822d68b83fe1ad97b32cd580d",
"baseTokenAmount”:”100000000000000000000”,"trader":"*адреса мого
гаманця*","rfqType":0}

r = requests.post(url = urlHashflow, json = dataHashflowBusdUsdcBSC,
headers = headersHashflow)
print(“Answer:”, r.json(), “\n”)
print(“quotetokenamount:”, r.json()[‘quoteData’][‘quoteTokenAmount’])
```

відправляє цей запит та виводить в консоль повне тіло відповіді API Hashflow разом із вихідною кількістю токену BUSD, що можливо отримати, виконавши відповідний поверненому котируванню обмін (рис. 3).

```
Answer: {'status': 'success', 'rfqId': '0xff696e626f756e64726671696466696c6c6572
7374720018eaa59c85acb80000', 'rfqType': 0, 'signature': '0xbfc315250dab2b182f1ea
c40cca301fb58196ad72534717685d550e06995a8ef1be1ae9ec4dc4bff61422f11bf2bd29e9b462
824e98183913f9269f0cf85a7a31c', 'quoteData': {'txid': '0xff67616e64616c667468656
2726f776e67786d786e690018eaa59cc3b3e30000', 'nonce': 1684852737219, 'rfqType': 0
, 'baseToken': '0xe9e7cea3dedca5984780bafc599bd69add087d56', 'quoteToken': '0x8a
c76a51cc950d9822d68b83fe1ad97b32cd580d', 'baseTokenAmount': '1000000000000000000
0000', 'quoteTokenAmount': '9998998850077512543066', 'quoteExpiry': 1684852857,
'trader': '                                          ', 'pool': '0x34a143ed5ef75
59b4db017d70df3128f782d0816'}, 'nativeTokenPriceUsd': 313.34, 'gasEstimate': 130
000, 'hftTradingRewards': '999700000000000000'}

quotetokenamount: 9998998850077512543066
>>>
```

Рисунок 3 — Тіло відповіді на запит та вихідна кількість цільового токену

Значення 9998998850077512543066 ключа **toTokenAmount**, вкладеного у відповідь JSON-об’єкта **quoteData**, відповідає вихідній кількості USDC приблизно у 9998.99 токенів.

2.3 Отримання курсів цифрових активів з біржі Binance через програмний інтерфейс

У біржі Binance є власний API [18], але я буду використовувати легку бібліотеку python-binance [19] для мови програмування Python, бо мене цікавлять лише **bidPrice** — найвища ціна на покупку; та **askPrice** — найменша ціна на продаж у конкретних торгових парах стейблоїнів.

Для отримання даних про торговельну пару BUSD/USDT на біржі Binance можна використати клас **Client** з модуля **binance.client**.

Спочатку створюю об'єкт **client** класу **Client** з використанням публічного та секретного ключів доступу до API Binance, далі метод **get_orderbook_tickers()** для створеного об'єкта повертає список усіх торгових пар із їх кращими цінами на покупку та продаж; потім проходжу по списку торгових пар та перевіряю умову, що поточна торговельна пара є шуканою. Нижче наведено фрагмент коду програми, що виводить у консоль ціни покупки та продажу .

```
BAPI_KEY = ‘*публічний ключ API*’
BSECRET_KEY = ‘*секретний ключ API*’
Client = Client(BAPI_KEY, BSECRET_KEY)
tickers = client.get_orderbook_tickers()
for i in tickers:
     if i[‘symbol’] == ‘BUSDUSDT’:
          print(“Bid price:”, i[‘bidPrice’])
          print(“Ask price:”, i[‘askPrice’])
```

Результати відповіді API (рис. 4) свідчать про те, що на біржі Binance один токен BUSD можна купити за 0.9999 токенів USDT, а продати за 0.9998 USDT.

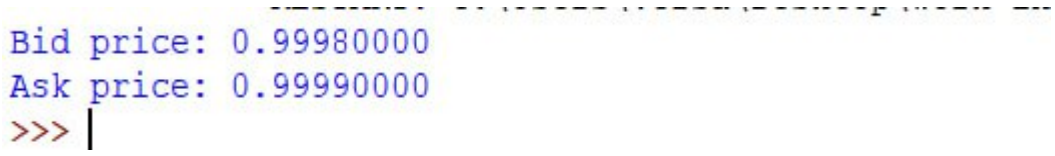


Рисунок 4 — Виведені у консоль ціни Bid та Ask

3 ВИКОРИСТОВУВАНІ ПРОГРАМНІ ЗАСОБИ МОВИ PYTHON

Програмні засоби, що були створені в рамках цієї роботи, були розроблені з використанням мови програмування Python версії 3.9 й деяких її стандартних та сторонніх бібліотек, а саме :

- **time** надає функціонал, пов'язаний з часом, і зазвичай використовується для введення затримок або вимірювання часових інтервалів у програмах на Python.
- **json** дозволяє робити операції з JSON-даними, такі як серіалізація та десеріалізація, кодування та декодування JSON-об'єктів.
- **datetime** — модуль, що надає класи для роботи з датами та часом у Python. Він дозволяє маніпулювати, форматувати та аналізувати дати і час.
- **Hexbytes** — це модуль, що надає клас **HexBytes**, який використовується для зображення шістнадцяткових послідовностей байтів у Python.
- **asyncio** — модуль, що надає інфраструктуру для написання однопотокового паралельного коду з використанням підпрограм, мультиплексування доступу до вводу/виводу через сокети та інші ресурси, запуску мережевих клієнтів і серверів та інших пов'язаних примітивів.
- **aiohttp 3.8.3** — бібліотека, що дозволяє робити асинхронні HTTP-запити в Python. Вона надає простий у використанні інтерфейс для надсилання HTTP-запитів та отримання відповідей.
- **pyTelegramBotAPI 4.8.0** — бібліотека для роботи з ботами для месенджеру Telegram.
- **python-binance 1.0.16** — модуль, який надає клієнтський доступ до Binance API.
- **web3 5.31.3** — це бібліотека Python для взаємодії з Ethereum-сумісними блокчейнами, яка надає високорівневий інтерфейс для взаємодії з блокчейнами, та смарт-контрактами.

- **geth_poa_middleware** — це проміжне програмне забезпечення, що надається бібліотекою web3 спеціально для взаємодії з блокчейнами Ethereum, які використовують алгоритм консенсусу Proof of Authority (PoA), реалізований клієнтом Geth.

3.1 Огляд модуля web3

Під час виконання роботи була потреба у детальному вивченні модуля **web3**, за допомогою якого можна взаємодіяти з блокчейнами. У рамках виконання кваліфікаційної роботи мене цікавив клас **Web3** цього модуля, за домогою якого можна підключатися до блокчейну, підписувати та відправляти транзакції, чекати на включення транзакції у блокчейн тощо.

Щоб підключитися до вузла блокчейну, необхідно ініціалізувати об'єкт класу **Web3** URL-адресою провайдера інфраструктури блокчейну. Це можна зробити, створив об'єкт класу **HTTPProvider**, який є спеціальною реалізацією провайдера інфраструктури блокчейну для **Web3**, що взаємодіє з вузлом блокчейну по HTTP. Він приймає URL-адресу провайдера в якості аргументу. Наприклад, за допомогою коду нижче ми можемо створити об'єкт класу **Web3** і присвоїти його змінній **w3**. Кінцевою точкою вузла блокчейну буде рядок ”https://my-example.com/123/”.

```
w3 = Web3(Web3.HTTPProvider(‘https://my-example.com/123/’))
```

Для того, щоб взаємодіяти зі смарт-контрактами на блокчейні, необхідно створити об'єкт класу **Contract** з модуля **web3.eth.contract**, передавши у конструктор два аргументи: адресу смарт-контракту й відповідний ABI. Наприклад, якщо ми хочемо взаємодіяти зі смарт-контрактом за адресою 0xeaE00113af330E6f67342B8298eFc40232cD9F09, що має тільки одну функцію **store**, яка приймає на вхід один аргумент **num** типу **uint256**, то можемо

використати такий код. Необхідний нам об'єкт буде значенням змінної **myContract**.

```
myAbi = [
{
    "inputs": [
      {
        "internalType": "uint256",
        "name": "num",
        "type": "uint256"
      }
    ],
    "name": "store",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  }
]
myContract = w3.eth.contract(
      address = ‘0xeaE00113af330E6f67342B8298eFc40232cD9F09’,
      abi = myAbi
)
```

Зазначимо, що модуль **web3** працює з адресами з контрольною сумою (checksum address). Адреса з контрольною сумою — це чутливе до регістру зображення адреси, яке додатково включає інформацію про контрольну суму, чим запобігає помилкам при введенні адреси і підвищує точність перевірки адреси. Наприклад, адреса 0xeaE00113af330E6f67342B8298eFc40232cD9**f**09 не є адресою з контрольною сумою, а адреса 0xeaE00113af330E6f67342B8298eFc40232cD9**F**09 — є. Для адреси **address** значення відповідної адреси у форматі контрольної суми можна задати виразом **Web3.toChecksumAddress(address)**.

Щоб викликати функцію **store**, необхідно отримати доступ до неї за

допомогою атрибута **functions** екземпляра контракту та передати аргумент **num**. Після цього на об’єкті функції необхідно викликати метод **build_transaction()**, що будує об’єкт транзакції. Наведений код нижче будує об’єкт транзакції, що викличе метод **store** екземпляру контракту **myContract** з аргументом **1**. Об’єкт транзакції буде збережено в змінну **tx**.

```
tx = myContract.functions.store(1).build_transaction()
```

Далі об’єкт транзакції можна підписати приватним ключем від гаманця-відправника за допомогою методу **sign_transaction()** модуля **eth.account**. Він приймає два аргументи: об'єкт транзакції, що підписується, і приватний ключ від криптовалютного гаманця, що підписуватиме транзакцію. Метод **sign_transaction()** повертає підписаний об'єкт транзакції. Код, наведений нижче, підпише раніше створений об’єкт транзакції **tx** за допомогою приватного ключа **privKey** та , а результат присвоїть змінній **tx_create**.

```
tx_create = w3.eth.account.sign_transaction(tx, privKey)
```

Після підписання транзакції залишається лише відправити її в блокчейн. Це можна зробити за допомогою методу **send_raw_transaction()** модуля **eth**. Він приймає атрибут **rawTransaction** об'єкта **tx_create** в якості аргументу, транслює транзакцію в мережу і повертає хеш транзакції, який присвоюється змінній **tx_hash**.

```
tx_hash = w3.eth.send_raw_transaction(tx_create.rawTransaction)
```

Ще можна почекати, поки транзакція буде включена в блокчейн за допомогою методу **wait_for_transaction_receipt()** модуля **eth**. Він приймає хеш транзакції **tx_hash** у якості аргументу.

```
w3.eth.wait_for_transaction_receipt(tx_hash)
```

Отже, за допомогою модуля **web3** можна підключатися до блокчейну, створювати об'єкти смарт-контрактів, взаємодіяти зі смарт-контрактами, підписувати, надсилати транзакції тощо. Я буду його використовувати у програмному засобі для відслідковування курсу й автоматичного обміну BNB.

4 МОНІТОРИНГ КУРСІВ СТЕЙБЛКОЇНІВ

4.1 Загальна ідея, використовувані напрямки обміну, блокчейни

Під час виконання роботи я обмежився тим, що користувача можуть цікавити обміни лише стейблокоїнів BUSD та USDC, після котрих його баланс у BUSD або USDC стане більшим. Наприклад, якщо можна обміняти кількість токенів BUSD, що дорівнює **n**, на кількість токенів USDC **m** таку, що **m** більше за **n**, то такий результат влаштує користувача.

Обміни у TUSD не були реалізовані через недостатню ліквідність цього стейблкоїна, а стейблкоїн USDT, що є найліквіднішим, але має досить непрозорі резерви [20], може бути лише проміжним етапом обміну. Наприклад, якщо у користувача на балансі криптовалютного гаманця є **n** токенів USDC або BUSD, то його може цікавити початковий обмін цієї кількості токенів USDC у **m** токенів USDT лише за умови, що подальший обмін **m** токенів USDT на централізованій біржі Binance дасть у результаті таку кількість токенів BUSD або USDC **k**, що **k** більше за **n.**

За допомогою API агрегатора децентралізованих бірж 1inch та гібридної біржі Hashflow програма отримує курси стейблкоїнів у блокчейнах Binance Smart Chain, Polygon та Avalanche. Більш того, біржа Hashflow також підтримує обміни стейблкоїнів між різними блокчейнами. Наприклад, можна обміняти певну кількість токенів USDC у блокчейні Polygon на відповідну кількість токенів BUSD у блокчейні Binance Smart Chain. Тому програма також реалізує отримання курсів для обміну стейблкоїнів з блокчейну Binance Smart Chain у Polygon і навпаки, з Binance Smart Chain у Avalanche і навпаки та з Avalanche у Polygon і навпаки.

Обміни на блокчейні Ethereum у рамках цієї роботи не розглядалися через відносно високі комісії за транзакції обміну та переказу стейблкоїнів.

4.2 Опис роботи програми

Загалом роботу програми, що моніторить курси стейблкоїнів можна описати за допомогою діаграми (див. рис. 5).

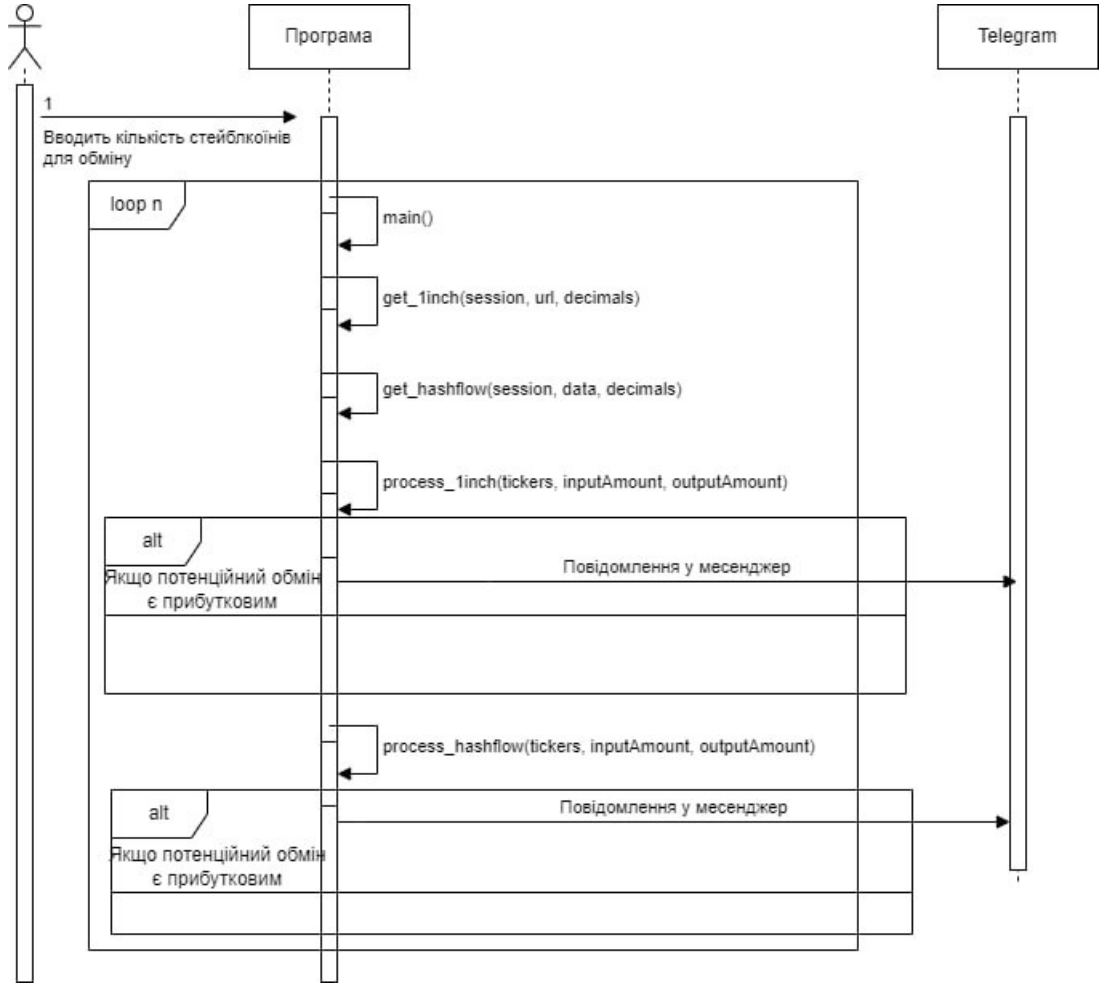


Рисунок 5 — Діаграма послідовностей роботи програми

Спочатку користувач вводить певну кількість стейблкоїнів, для різних блокчейнів і платформ для обміну, результат обміну якої хоче побачити. У програмі вже попередньо задані всі необхідні тіла запитів, URL API, необхідні заголовки запитів тощо. Далі програма у циклі асинхронно надсилає запити із різними напрямками обміну до API 1inch та Hashflow й обробляє відповіді запитів. Так як нам важлива вихідна кількість стейблкоїнів, то для кожного напрямку обміну програма порівнює вхідну й вихідну кількість стейблкоїнів.

Якщо вихідна кількість стейблкоїнів після обміну більша, ніж вхідна, то програма надсилає повідомлення користувачу в Telegram [21] з напрямком та потенційним результатом обміну (див. рис. 6).

Рисунок 6 — сповіщення в месенджері Telegram із вказаною платформою, напрямком та результатом обміну.

4.3 Деталі реалізації

Розроблений консольний додаток використовує модулі **time**, **aiohttp**, **asyncio**, **datetime**, **pyTelegramBotAPI** та **python-binance** мови Python.

Взаємодія з Telegram реалізується через глобальну змінну **bot**, значенням якої є об'єкт Telegram бота. Для виконання запитів до біржі Binance використовується Binance API, доступ до якого отримується через глобальну змінну **client**, значенням якої є об'єкт класу **Client** з модуля **binance.client**.

Після ініціалізації значень глобальних змінних **bot** та **client** користувач вводить три значення:

- **usdInputBSC1inchAndHashflow** — кількість стейблкоїнів для обміну, що буде використовуватись при запитах котирувань у блокчейні BSC на 1inch та Hashflow;
- **usdInputPolygonHashflow** — кількість стейблкоїнів для обміну, що буде використовуватись при запитах котирувань на Hashflow для усіх інших блокчейнів й для запитів котирувань для обмінів між різними блокчейнами;

- **usdcInputPolygon1inch** — кількість стейблкоїнів для обміну, що буде використовуватись тільки при запитах котирувань у блокчейні Polygon на 1inch.

Якщо користувач не введе жодного значення, за замовчуванням будуть використані значення 102000, 30000 та 20000 відповідно. Значення були підібрані, виходячи з ліквідності стейблкоїнів у різних блокчейнах.

Далі задаються URL-адреси для взаємодії з 1inch API на різних блокчейнах; заголовки для взаємодії з API 1inch; URL-адреса, що використовується для отримання запитів на цінові котирування з Hashflow; параметри авторизації для доступу до API Hashflow. (Параметри URL-адрес залежать від заданих користувачем значень.)

На наступному кроці створюються численні об'єкти, що зберігають завчасно задані тіла запитів для API Hashflow.

Взаємодія з 1inch API реалізується у функції **get_1inch()**. Ця функція виконує асинхронний GET-запит до 1inch API та повертає JSON-об’єкт, що містить дані про курс обміну, використані для оптимального котирування протоколи, кількість активів та інші параметри, що потрібні для виконання обміну.

За виникнення помилок (наприклад, через недоступність API, через ліміти запитів чи просто через неможливість здійснення обміну) повертається фіксоване невелике значення, яке є надто малим, щоб сприйняти його за реальний результат, але при цьому дозволяє залежним від нього компонентам програми працювати без неочікуваних результатів.

Функція **get_hashflow()** виконує аналогічні дії для API Hashflow.

Функції **process_hashflow()** та **process_1inch()** призначені обробляти результати, отримані з функцій **get_1inch()** та **get_hashflow()** відповідно. Якщо різниця між кількостями цільового та вхідного стейблкоїнів більше заданого значення, то кожна з цих функцій надсилає повідомлення в Telegram. Незалежно від цього на консоль виводиться повідомлення з назвою платформи, напрямком та результатом обміну.

Основна робота програми виконується асинхронною функцією **main()**. Відслідковуються кращі ціни купівлі та продажу у торгових парах BUSD/USDT та USDC/USDT.

Функція **main()** запускає нескінченний цикл, який отримує та обробляє дані один раз у шість секунд. На кожній ітерації циклу функцією **asyncio.ensure_future()** на виконання запускаються HTTP-запити до різних кінцевих точок, обгорнуті у виклики функцій **get_1inch()** і **get_hashflow()**. Після запуску за допомогою **await**-виклику функції **asyncio.gather()** очікуємо завершення цих завдань.

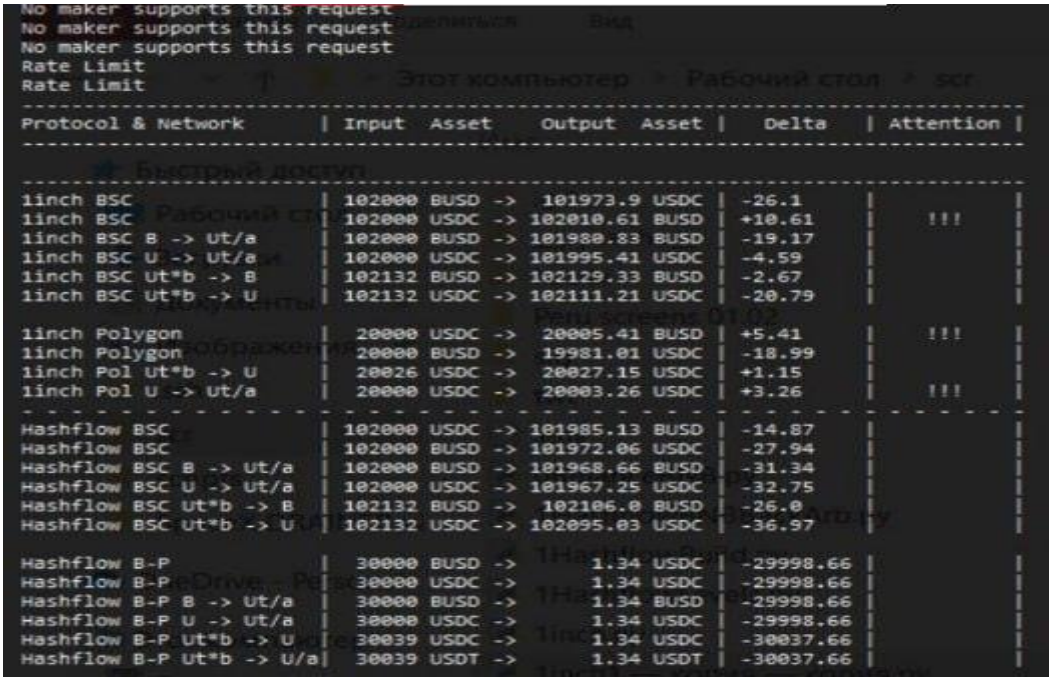
Після завершення виконання всіх запитів зібрані результати обробляються за допомогою функцій **process_1inch()** і **process_hashflow()**. Як результат, програма друкує таблицю, що містить назви стовпців для протоколу й блокчейну обміну, вхідної та вихідної суми, вхідної та вихідної валюти, можливий прибуток. У визначених раніше випадках надсилається повідомлення у Telegram.

Загалом, функція **main()** координує асинхронне виконання декількох завдань, отримує та обробляє дані, а також друкує відформатовані результати в табличному форматі. Вона продовжує виконуватися до нескінченності, багаторазово отримуючи та оновлюючи дані.

Повністю код програми наведено в додатку А.

4.4 Результат роботи програми

Після запуску та введення кількості стейблкоїнів для розрахунків можливих результатів обміну, програма виведе у консоль форматовані дані у таблицю (див. рис. 7).



```
No maker supports this request
No maker supports this request
No maker supports this request
Rate Limit
Rate Limit
-------------------------------------------------------------------------------
Protocol & Network      | Input  Asset    Output  Asset |   Delta   | Attention |
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
1inch BSC               | 102000 BUSD ->  101973.9 USDC | -26.1     |           |
1inch BSC               | 102000 USDC -> 102010.61 BUSD | +10.61    |    !!!    |
1inch BSC B -> Ut/a     | 102000 BUSD -> 101980.83 BUSD | -19.17    |           |
1inch BSC U -> Ut/a     | 102000 USDC -> 101995.41 USDC | -4.59     |           |
1inch BSC Ut*b -> B     | 102132 BUSD -> 102129.33 BUSD | -2.67     |           |
1inch BSC Ut*b -> U     | 102132 USDC -> 102111.21 USDC | -20.79    |           |

1inch Polygon           |  20000 USDC ->  20005.41 BUSD | +5.41     |    !!!    |
1inch Polygon           |  20000 BUSD ->  19981.01 USDC | -18.99    |           |
1inch Pol Ut*b -> U     |  20026 USDC ->  20027.15 USDC | +1.15     |           |
1inch Pol U -> Ut/a     |  20000 USDC ->  20003.26 USDC | +3.26     |    !!!    |
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Hashflow BSC            | 102000 USDC -> 101985.13 BUSD | -14.87    |           |
Hashflow BSC            | 102000 BUSD -> 101972.06 USDC | -27.94    |           |
Hashflow BSC B -> Ut/a  | 102000 BUSD -> 101968.66 BUSD | -31.34    |           |
Hashflow BSC U -> Ut/a  | 102000 USDC -> 101967.25 USDC | -32.75    |           |
Hashflow BSC Ut*b -> B  | 102132 BUSD ->  102106.0 BUSD | -26.0     |           |
Hashflow BSC Ut*b -> U  | 102132 USDC -> 102095.03 USDC | -36.97    |           |

Hashflow B-P            |  30000 BUSD ->      1.34 USDC | -29998.66 |           |
Hashflow B-P            |  30000 USDC ->      1.34 USDC | -29998.66 |           |
Hashflow B-P B -> Ut/a  |  30000 BUSD ->      1.34 BUSD | -29998.66 |           |
Hashflow B-P U -> Ut/a  |  30000 USDC ->      1.34 USDC | -29998.66 |           |
Hashflow B-P Ut*b -> U  |  30039 USDC ->      1.34 USDC | -30037.66 |           |
Hashflow B-P Ut*b -> U/a|  30039 USDT ->      1.34 USDT | -30037.66 |           |
```

Рисунок 7 — Друк результатів роботи програми у консоль

Спочатку виводяться помилки при спробах надсилання деяких запитів, наприклад, «Rate Limit» свідчить про те, що програма надсилає надто багато запитів до API Hashflow, а «No maker supports this request» — про те, що недостатньо ліквідності для деяких обмінів.

Далі друкується таблиця з назвою платформи, блокчейну та напрямком для обміну, вхідною й вихідною кількістю стейблкоїнів, назви відповідних стейблкоїнів й можливий прибуток від обміну. Деякі назви були скорочені, щоб зекономити місце на екрані під час друку. Ось тлумачення скорочень програми:

- Pol — Polygon
- B — BUSD
- U — USDC
- Ut — USDT
- B-P — обмін між блокченами BSC та Polygon
- Позначення виду «B -> Ut/a» свідчить про те, що під час обміну BUSD ми отримуємо USDT, які далі потрібно продати на біржі Binance у BUSD. Фінальна кількість отриманих BUSD буде дорівнювати кількості USDT,

поділеній на кращу ціну продажи BUSD.

- Позначення виду «Ut*b -> B» свідчить про те, що спочатку потрібно завести BUSD на біржу Binance, обміняти на USDT. При цьому кількість отриманих USDT буде дорівнювати кількості BUSD, помноженій на кращу ціну покупки BUSD. Далі USDT необхідно обміняти на BUSD на зазначеній платформі

У прикладі вище програма знайшла декілька прибуткових напрямків обміну. Наприклад, за тих умов можна було обміняти 102 000,0 USDC на 102 000,6 BUSD. Під час тестування програма регулярно знаходила потенційно прибуткові напрямки обміну, більшість з яких були вдало використані. Прикладом може слугувати обмін [22] (див. рис. 8).

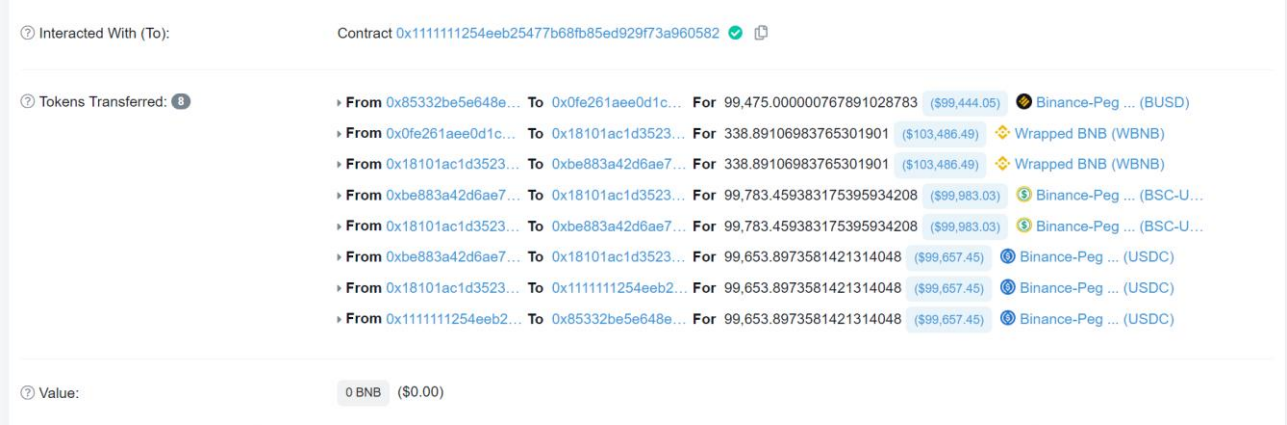


Рисунок 8 — Приклад вдалого обміну, можливість якого спочатку була зафіксована програмою

Отже, програма знаходить неефективності ринку навіть на прикладі стейблкоїнів, кожен з яких може бути конвертований у долар США у співвідношенні один до одного.

5 МОНІТОРИНГ КУРСУ ТА АВТОМАТИЧНИЙ ОБМІН BNB

5.1 Мотив розробки програми, вибір блокчейну й цифрового активу

Під час розробки програми для моніторингу курсів стейблкоїнів, мною була вивчена архітектура гібридної біржі Hashflow та її API. Як вже відомо, при запиті котирування через API, можна отримати підписане котирування, що дійсне обмежену кількість часу. Зазвичай це приблизно тридцять секунд. Під час збирання підписаних котирувань на різні торгові пари стало помітно, що через коливання цін на цифрові активи може виникнути ситуація, коли в масиві котирувань будуть такі два котирування:

- Досі дійсне котирування на покупку цифрового активу X за ціною n
- Досі дійсне котирування на продаж цифрового активу X за ціною m

При тому m буде більшим за n. Виходить, що можна було б здійснити два обміни за допомогою цих підписаних котирувань: перше на купівлю активу X за ціною n, а друге — на продаж за ціною m. Так як m більше за n, то обмін був би прибутковим. Ця ідея є мотивом розробки другої програми, що, на відміну від першої, буде не тільки моніторити курси, а ще й здійснювати обміни через смарт-контракти Hashflow. Більше того, відслідковування курсів стейблкоїнів у цьому випадку є недоречним через їх маленькі коливання у ціні.

Біржа Hashflow працює у декількох блокчейнах, але я вибрав блокчейн Binance Smart Chain через відносну дешевизну виконання транзакцій на ньому. Активом для обміну був вибраний Binance Coin — цифровий актив, яким сплачуються комісії блокчейну BSC. Провайдером інфраструктури, за допомогою якого програма буде підключатися до блокчейну й надсилати транзакції, був вибраний QuickNode, тому що цей сервіс є безкоштовним для використання можливостей, що потрібні програмі.

5.2 Опис роботи програми

Роботу програми можна умовно зобразити за допомогою діаграми (див. рис. 9).

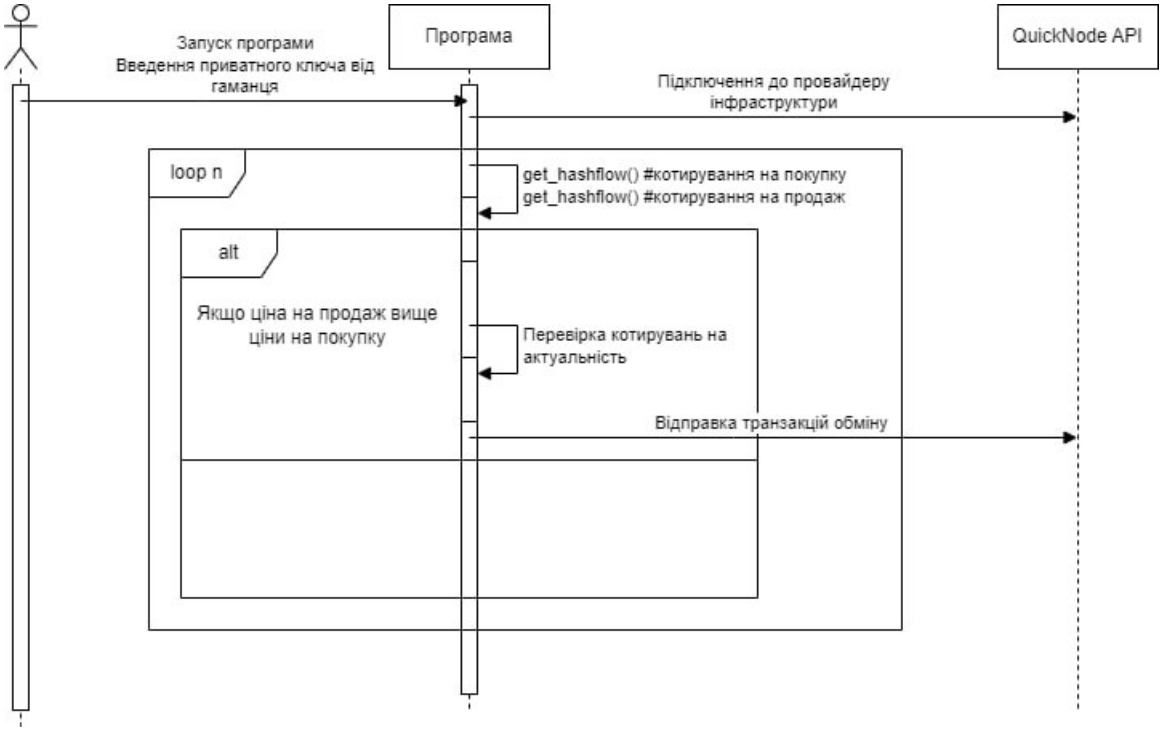


Рисунок 9 — Діаграма послідовностей роботи програми

Спочатку користувач запускає програму й вводить приватний ключ від гаманця, на якому є BUSD для купівлі BNB та BNB для сплати комісій блокчейну. Одразу після цього програма починає у нескінченному циклі отримувати підписані котирування на покупку й продаж BNB за BUSD через API Hashflow та зберігати їх у окремі масиви. Після цього у масивах програма шукає котирування на покупку BNB за найнижчою ціною та котирування на продаж BNB за найвищою ціною. Під час їх пошуку також видаляються вже недійсні котирування. Якщо програма знаходить котирування на продаж за ціною, вищою, ніж котирування на покупку, то до провайдеру інфраструктури блокчейну надсилається транзакція на покупку, підписана приватним ключем від криптовалютного гаманця. При цьому програма продовжує збирати котирування

на продаж до тих пір, поки ціна не почне знижуватися. Щоб запобігти збитків через помилки API, також перевіряється, щоб нові котирування були дійсними. Коли ціна починає знижуватися, й останнє котирування ще буде дійсним декілька секунд, програма відправляє транзакцію на продаж такої кількості BNB, що складається з перших чотирьох значущих цифр кількості BNB, що була куплена у першій транзакції. Це зроблено для того, щоб на гаманці завжди була якась кількість BNB для сплати комісій блокчейну.

5.3 Деталі реалізації

Програма використовує бібліотеки **time**, **json**, **aiohttp**, **asyncio**, **datetime**, **hexbytes**, **web3** та **geth_poa_middleware** мови програмування Python. Наявні в них засоби забезпечують наявність необхідних засобів для взаємодії з Hashflow API і блокчейном Ethereum.

Спочатку користувачу пропонується ввести приватний ключ від криптовалютного гаманця; без приватного ключа підписання транзакцій та доступу до коштів гаманця неможливе.

Потім створюється об'єкт класу **Web3**, який реалізує з'єднання з блокчейном BSC, і запам'ятовується в змінній **w3**. Цей об'єкт ініціалізується HTTP-провайдером, який вказує URL-адресу вузла BSC, до якого потрібно підключитися. У цьому випадку він підключається до кінцевої точки QuickNode за URL-адресою. Ця кінцева точка дозволяє зв'язок з блокчейном Binance Smart Chain.

Клієнт Geth, який використовується для підключення до блокчейну BSC, використовує алгоритм консенсусу Proof of Authority, для обробки якого призначене проміжне програмне забезпечення **geth_poa_middleware**, що вбудовується в стек **middleware_onion** проміжного програмного забезпечення об'єкта **w3**.

Після цього змінній **hftContract** присвоюється об'єкт класу **Contract** з модуля **web3.eth**. Він зображує смарт-контракт в блокчейні Binance Smart Chain та ініціалізується двома параметрами:

— **address** вказує адресу (з контрольною сумою) контракту в блокчейні.

— **abi** описує ABI контракту. ABI необхідний для взаємодії з функціями контракту та отримання даних з нього. У даному випадку ABI являє собою список з єдиним елементом, яким є словник, що зображує функцію **tradeSingleHop** структури **IQuote.RFQTQuote**.

До функції **tradeSingleHop** [16] відносяться:

- **pool** — адреса пулу ліквідності для угоди;
- **externalAccount** — адреса зовнішнього рахунку;
- **trader** — адреса гаманця;
- **effectiveTrader** — адреса гаманця користувача;
- **baseToken** — адреса вхідного токена;
- **quoteToken** — адреса цільового токена;
- **effectiveBaseTokenAmount** — цілочисельне значення, що задає суму базового токена;
- **maxBaseTokenAmount** — максимальна сума базового токена;
- **maxQuoteTokenAmount** — максимальна сума цільового токена;
- **quoteExpiry** — значення терміну дії котирування;
- **nonce** — значення параметра nonce. Блокчейн перевіряє значення параметра **nonce**, щоб переконатися, що транзакції виконуються в правильному порядку і не дублюються;
- **txid** — ідентифікатор транзакції у вигляді bytes32;
- **signature** — підпис котирування в байтах.

Усі адреси мають бути з контрольною сумою.

Обміни програма буде виконувати, викликаючи функцію **tradeSingleHop** смарт-контракту біржі Hashflow. Код, за допомогою якого, створюється об'єкт класу **hftContract**, зокрема вигляд опису ABI, повністю наведено в додатку Б.

Використовувані глобальні змінні **urlHashflow** і **headersHashflow** зберігають URL-адресу і заголовок запиту до API Hashflow для отримання підписаних котирувань. Словники **dataHashflowBUSDBNBBSC** і **dataHashflowBNBBUSDBSC**, призначені для зберігання параметрів даних, що необхідні для здійснення API-запитів до платформи Hashflow, зокрема, для запиту котирувань і виконання угод.

Словник **dataHashflowBUSDBNBBSC** містить ключі:

- **networkId** — ідентифікатор блокчейну Binance Smart Chain, який має значення 56;
- **source** — джерело запиту, яке має значення "api";
- **baseToken** — адреса смарт-контракту вхідного токена, яким є BUSD;
- **quoteToken** — адреса смарт-контракту цільового токена, яким є BNB;
- **baseTokenAmount** — кількість вхідного токена для обміну. У даному випадку це 10000 BUSD;
- **trader** — адреса гаманця, що буде виконувати обмін;
- **rfqType** — тип запиту на котирування, який встановлено на 0.

Словник **dataHashflowBNBBUSDBSC** містить аналогічні параметри, як і **dataHashflowBUSDBNBBSC**, але з вхідним і цільовим токенами, поміняними місцями. Він відображає обмін BNB на BUSD в блокчейні BSC.

Функція з заголовком

```
def buildAndSendTx(answer, effectiveAmount, traderNonce):
```

відповідає за правильну побудову і відправку транзакції для здійснення угоди на основі наданих вхідних даних.

Вона отримує параметри **answer**, **effectiveAmount** і **traderNonce**, що використовуються для правильної побудови транзакції. Аргумент **answer** є підписаним котируванням, що його повернув API Hashflow; **effectiveAmount** є округленою до ста тисяч кількістю вхідного токена, а **traderNonce** — це послідовний номер, який відповідає кількості транзакцій, відправлених з певної адреси. Він використовується для забезпечення дотримання порядку і

запобігання атакам повторного відправлення транзакцій. Кожна транзакція містить таке значення, і воно повинно збігатися зі значенням параметру **nonce** гаманця відправника.

Послідовність дій функції **buildAndSendTx()** виглядає так.

1. За отриманими через параметр **answer** значеннями формуються аргументи виклику та викликається функція **tradeSingleHop()** смарт-контракту **hftContract**. Отримується об'єкт транзакції.

2. За параметром **effectiveAmount** встановлюється значення **value**, що буде використовуватись у транзакції.

3. Метод **build_transaction()** побудованому об'єкту транзакції передає додаткові параметри.

4. Транзакція підписується методом **w3.eth.account.sign_transaction()**.

5. Підписана транзакція відправляється до провайдера інфраструктури QuickNode за допомогою функції **w3.eth.send_raw_transaction()**. Отримуємо хеш транзакції.

6. Програма через виклик **w3.eth.wait_for_transaction_receipt()** чекає на включення транзакції у блокчейн. Нарешті, у консоль виводиться повідомлення про успішне включення транзакції, яке містить хеш транзакції у шістнадцятковому форматі.

Асинхронна функція **get_hashflow()** виконує POST-запит до вказаної кінцевої точки API Hashflow, що зберігається в змінній **urlHashflow**, використовуючи aiohttp-сесію. За успішного виконання запиту повертає словник з його результатом, інакше словник **{'status': 'fail'}**.

Алгоритм торгівлі реалізовано в асинхронній функції **main()**.

Спочатку створюється об'єкт **aiohttp.ClientSession**. Надалі створена сесія буде використовуватися для створення HTTP-запитів у функції **get_hashflow()**.

Далі ініціалізуються два порожніх об’єкти **buys** і **sells** класів **BuyQuoteArray** та **SellQuoteArray** для зберігання котирувань на купівлю і продаж відповідно, та задаються початкові значення змінних **bestSellRateQuote**

і **bestBuyRateQuote**. У коді визначено три класи: **QuoteArray**, **BuyQuoteArray** та **SellQuoteArray**. Ці класи надають додаткову функціональність для виконання операцій на котируваннями.

Почнемо з класу **QuoteArray**, що успадковується від вбудованого класу **list**. Він ініціалізує атрибут **best** значенням за замовчуванням з атрибуту **WORSE**, визначеного пізніше в підкласах. Містить метод **deleteExpired()** для видалення прострочених котирувань зі списку на основі переданого у метод параметру **maxSecondsLeft** (заданої максимальної кількості секунд, що залишилися). Метод виконує наступні дії:

1. Отримує поточну мітку часу за допомогою **datetime.now().timestamp()**.
2. Записує в значення атрибуту **best** значення копії атрибуту **WORSE**.
3. Отримує довжину списку котирувань та ініціалізує дві змінні, **i** та **j**, значенням 0. Ці змінні використовуються у якості індексів для ітерації по списку.
4. Далі входить у цикл **while**, який триває до тих пір, поки **i** не стане меншим за довжину списку. Усередині циклу перевіряється, чи не закінчився термін дії котирування за індексом **i**. Час закінчення обчислюється шляхом віднімання **maxSecondsLeft** від мітки часу **quoteExpiry** котирування і порівняння його з поточною міткою часу.
5. Якщо термін дії котирування не закінчився, воно переміщується до індексу **j** у списку, присвоюючи елементу списку із індексом **j** значення елементу під індексом **i**. Також викликається метод **_update_best()**, щоб оновити найкраще котирування, а **j** збільшується на одиницю
6. Після завершення циклу решта елементів, починаючи з індексу **j**, видаляються зі списку за допомогою **del self[j:]**. Це видаляє котирування, що вже недійсні.

Таким чином, цей метод ітераційно переглядає котирування у списку, перевіряє, чи не закінчився термін дії кожного котирування, оновлює найкраще котирування на основі критеріїв, визначених у підкласах, і видаляє зі списку

котирування, термін дії яких закінчився. Загальна часова складність методу **deleteExpired()** становить **O(n)**, де **n** — кількість елементів у списку.

Також клас **QuoteArray** перевизначає метод **append()** для додавання котирувань до списку. Він додає переданий елемент до списку котирувань тільки тоді, якщо ключ котирування **status** має значення **success**. Крім того, для кожного котирування він обчислює і зберігає ключ **myRate** за допомогою виклику методу **_transfrom()**.

Раніше згадані методи **_transfrom()** та **_update_best()** мають специфічну реалізацію у класах **BuyQuoteArray** та **SellQuoteArray**.

Обидва класи **BuyQuoteArray** та **SellQuoteArray** визначають атрибут **WORSE** як словник з однією парою ключ-значення, що представляє гірший курс купівлі або продажу відповідно. Також вони реалізують метод **_transform()**, що додає до котирування ключ **myRate** зі значенням на основі кількості вхідного та цільового токенів у даних котирування, і метод **_update_best()** для оновлення найкращого котирування купівлі/продажу

Як уже було сказано, метод **_transform()** викликається під час додавання котирування до списку котирувань. У реалізації цього методу в класі **BuyQuoteArray** ключ котирування **myRate** зберігається зі значенням частки від ділення кількостей вхідного і цільового токену, а в класі **SellQuoteArray** навпаки — зі значенням частки від ділення кількостей цільового і вхідного токену.

Схожа ситуація і з методом **_update_best()**, що викликається всередині методу **deleteExpired()** для кожного досі дійсного котирування. У класі **BuyQuoteArray** метод **_update_best()** оновлює атрибут **best** переданим котируванням, якщо курс купівлі в цьому котируванні нижчий за існуючий найкращий курс купівлі, а в класі **SellQuoteArray** — якщо курс продажу в котируванні вищий за існуючий найкращий курс продажу.

Отже, ці класи забезпечують функціонал для управління списками котирувань на купівлю й продаж BNB. Класи **BuyQuoteArray** і **SellQuoteArray** розширюють функціональність класу **QuoteArray**, реалізуючи специфічну

поведінку для додавання котирувань і пошуку найкращого цінового котирування на основі певних критеріїв. Зв’язки між класами та їх структуру можна побачити на діаграмі класів нижче (див. рис. 10).

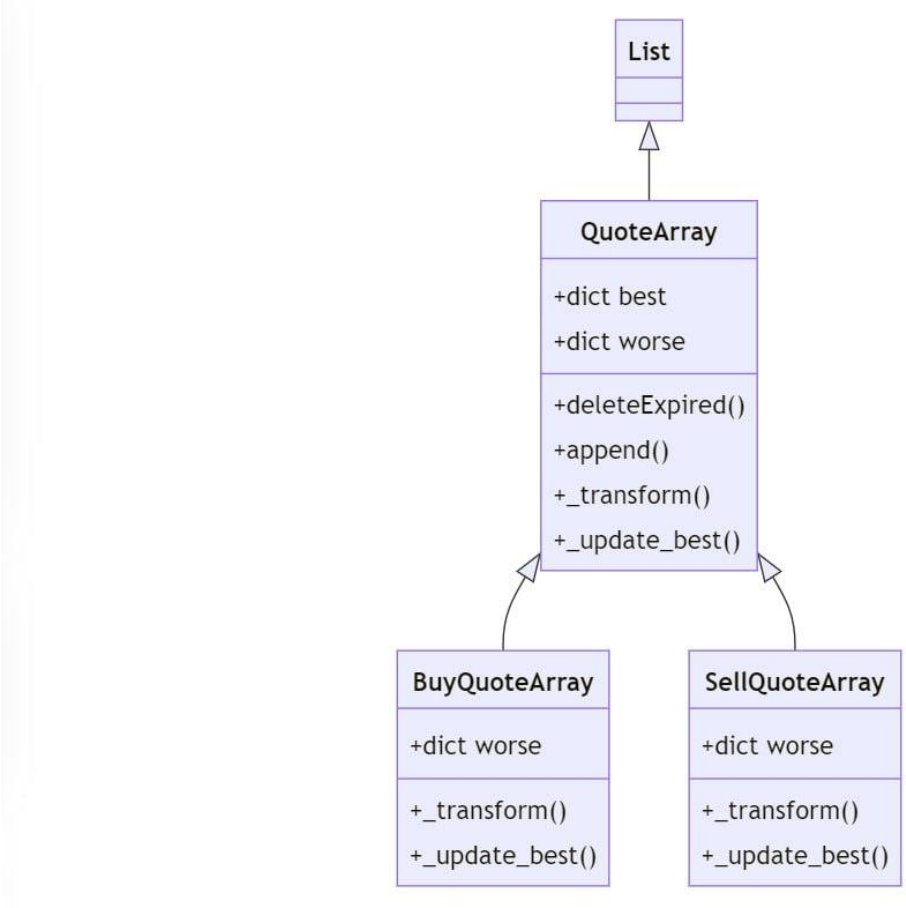


Рисунок 10 — Діаграма класів

Після створення об’єктів **buys** і **sells** запускається нескінченний цикл. Це означає, що алгоритм буде працювати нескінченно, поки його не зупинять вручну.

На кожній ітерації створюється список **tasks** для зберігання асинхронних завдань для виконання запитів. У цій програмі в нього додаються два завдання до списку завдань, кожне з яких використовує функцію **get_hashflow()** для відправки запиту до Hashflow API з відповідними даними. Також у межах циклу налаштовуються тіла запитів до Hashflow API: **dataHashflowBUSDBNBBSC** і **dataHashflowBNBBUSDBSC**.

Далі використовується **await**-виклик **asyncio.gather** для асинхронного

виконання всіх завдань у списку завдань і очікування їх результатів.

По завершенню виконання завдань програма за допомогою методу **append()** додає відповіді на запити котирування купівлі й продажу до списків об’єктів **buys** і **sells** відповідно. Під час додавання до списків перевіряється статус відповіді й обчислюється ключ котирувань **myRate**. Далі ці списки ітераційно переглядаються, використовуючи метод **deleteExpired()**, щоб відфільтрувати прострочені котирування і знайти найкращу пропозицію за курсом купівлі й продажу. Змінним **bestBuyRateQuote** і **bestSellRateQuote** присвоюються значення атрибутів **best** об’єктів **buys** і **sells** відповідно**.**

Після цього програма перевіряє, чи найкращий курс купівлі нижчий за найкращий курс продажу. Також є перевірка на те, що котирування на продаж отримано пізніше, ніж котирування на купівлю. У іншому випадку смарт-контракт біржі не дасть здійснити обміни. Якщо так, то програма переходить до алгоритму, що відповідає за відправку транзакцій на обмін та отримання нових котирувань.

Отримується поточне значення параметру **nonce** для адреси гаманця користувача за допомогою **w3.eth.get_transaction_count**, викликається функція **buildAndSendTx()** для побудови і відправки транзакції за найкращим котируванням курсу покупки. Потім видаляється оброблене котирування курсу покупки зі списку об’єкту **buys** і оновлюється **baseTokenAmount** в **dataHashflowBNBBUSDBSC** округленою сумою отриманих цільових токенів. Програма входить у цикл, який триває до тих пір, поки не буде досягнутий час експірації найкращого котирування курсу продажу. Під час циклу він надсилає запит до Hashflow API для отримання нового котирування на продаж і оновлює котирування найкращого курсу продажу, якщо нове котирування має вищий курс. Після виходу з циклу програма знову викликає функцію **buildAndSendTx()**, щоб побудувати і відправити транзакцію за найкращим курсом продажу на обмін. Оброблене котирування курсу продажу видаляється зі списку **sells**.

Програма також друкує відповідну інформацію, таку як довжина списків об’єктів **buys** та **sells**, час, що залишився до закінчення терміну дії найкращих котирувань на купівлю та продаж, а також курси найкращих котирувань на купівлю та продаж. Перед початком наступної ітерації циклу є пауза, що складає чотири секунди.

5.4 Оцінка роботи програми

Програма була протестована й змогла зафіксувати й використати можливості для автоматичного обміну BNB з ціллю прибутку. В якості прикладу я наведу дві транзакції, відправлені програмою: одна на купівлю BNB [8] за 10000 BUSD (див. рис. 11), друга — на продаж тієї ж кількості BNB [24] за 10045 BUSD (див. рис. 12).

Рисунок 11 — Купівля BNB

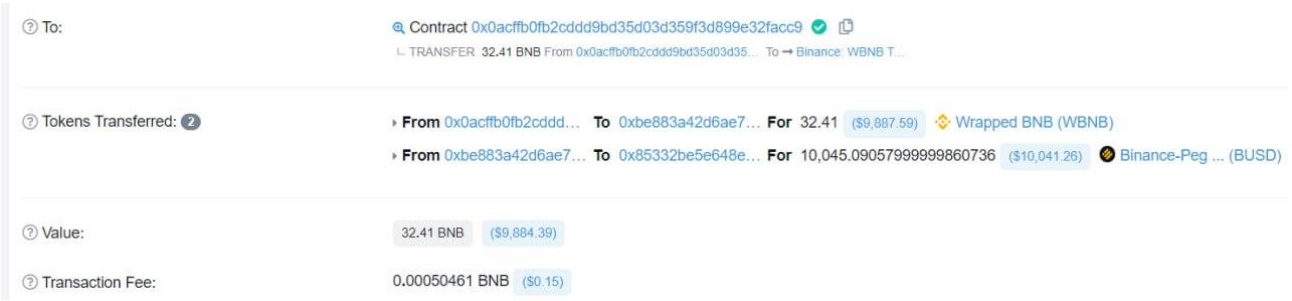


Рисунок 12 — Продаж BNB із прибутком

Отже, програмний засіб зміг використати вразливість біржі Hashflow, а саме можливість частого збирання підписаних котирувань та їх виконання за певних умов.

5.5 Перешкоди для роботи програми, створені біржою Hashflow

Програма для моніторингу курсу та автоматичного обміну BNB демонструє вразливість біржі Hashflow, що дозволяє отримувати прибуток за рахунок здійснення обмінів за котируваннями з різною ціною. Зрозуміло, що покупка BNB, що здійснюється першою транзакцією, йде по трохи запізнілій ціні, що виливається у збиток біржі. Команда біржі була проінформована мною, що існує ризик використання архітектури їх біржі для отримання безризикової вигоди за їх рахунок. У свою чергу мене запевнили, що платформа буде боротися проти користувачів, що використовують їх API для подібних цілей. Незабаром це дійсно стало відчутно через те, що після кількох вдалих спроб отримання котирувань, подальші отримані через API котирування були дійсними лише декілька секунд, що унеможливлювало тривалу роботу програми.

ВИСНОВКИ

Під час виконання роботи я вивчив особливості API 1inch та Hashflow, інтегрувався з останнім, ознайомився з модулем web3 мови програмування Python, зміг за допомогою нього взаємодіяти зі смарт-контрактом біржі Hashflow. Далі було побудовано два незалежні програмні засоби, перший з яких відслідковує курси стейблкоїнів, шукаючі неефективності ринку, а другий — намагається використовувати вразливості біржі Hashflow, збираючи підписані котирування й автоматично виконуючи обміни за певних умов.

За допомогою програмного засобу для відслідковування курсів стейблкоїнів були помічені регулярні неефективності ринку, а засіб для автоматичного обміну BNB зміг використати вразливість біржі Hashflow. Тим не менш, команда біржі Hashflow була повідомлена про наявність вразливості і змогла досить швидко й ефективно її виправити.

Таким чином, у результаті вдалося отримати робочі програмні засоби, що наглядно демонструють те, що ринок цифрових активів іноді є неефективним через свою новизну, а деякі децентралізовані платформи містять вразливості у своїй архітектурі, і ці вразливості можуть бути використані. Подібні вразливості можуть стати наслідками фінансових збитків біржі, тому дуже важливо проводити аудит систем та швидко реагувати на повідомлення про потенційні проблеми в роботі обмінної платформи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What Is Decentralized Finance (DeFi) and How Does It Work? [Електронний ресурс] – Режим доступу до ресурсу : https://www.investopedia.com/decentralized-finance-defi-5113835

2. The Decentralized Exchange Landscape [Електронний ресурс] – Режим доступу до ресурсу : https://www.gemini.com/cryptopedia/decentralized-exchange-dex-crypto

3. What are smart contracts on blockchain? [Електронний ресурс] – Режим доступу до ресурсу : https://www.ibm.com/topics/smart-contracts

4. Hashflow: A DEX for Bridgeless Cross-chain Swaps, Zero Slippage, and MEV-protected Trades [Електронний ресурс] – Режим доступу до ресурсу : https://consensys.net/blog/cryptoeconomic-research/hashflow-a-dex-for-bridgeless-cross-chain-swaps-zero-slippage-and-mev-protected-trades/

5. What Are DEX Aggregators? A Deep Dive by 1inch Trades [Електронний ресурс] – Режим доступу до ресурсу : https://coinmarketcap.com/alexandria/article/what-are-dex-aggregators-a-deep-dive-by-1inch

6. Binance to Auto-Convert USDC, USDP, TUSD to BUSD (Binance USD) [Електронний ресурс] – Режим доступу до ресурсу : https://www.binance.com/en/support/announcement/binance-to-auto-convert-usdc-usdp-tusd-to-busd-binance-usd-e62f703604a94538a1f1bc803b2d579f

7. Hashflow Intro [Електронний ресурс] – Режим доступу до ресурсу : https://docs.hashflow.com/hashflow/

8. Blockchain Facts: What Is It, How It Works, and How It Can Be Used [Електронний ресурс] – Режим доступу до ресурсу : https://www.investopedia.com/terms/b/blockchain.asp

9. What are EVM Compatible Blockchains? A Guide to the Ethereum Virtual Machine [Електронний ресурс] – Режим доступу до ресурсу :

https://blog.thirdweb.com/evm-compatible-blockchains-and-ethereum-virtual-machine

10. Quote: Definition in Trading and Investing [Електронний ресурс] – Режим доступу до ресурсу : https://www.investopedia.com/terms/q/quote.asp

11. Stablecoins: Definition, How They Work, and Types [Електронний ресурс] – Режим доступу до ресурсу : https://www.investopedia.com/terms/s/stablecoin.asp

12. What Are Crypto Tokens, and How Do They Work? [Електронний ресурс] – Режим доступу до ресурсу : https://www.investopedia.com/terms/c/crypto-token.asp

13. Quote params. Find the best quote to exchange via 1inch router [Електронний ресурс] – Режим доступу до ресурсу : https://docs.1inch.io/docs/aggregation-protocol/api/quote-params

14. Binance-Peg BUSD Token [Електронний ресурс] – Режим доступу до ресурсу : https://bscscan.com/token/0xe9e7cea3dedca5984780bafc599bd69add087d56

15. Token Binance-Peg USD Coin Token [Електронний ресурс] – Режим доступу до ресурсу : https://bscscan.com/token/0x8ac76a51cc950d9822d68b83fe1ad97b32cd580d

16. Hashflow Getting Started [Електронний ресурс] – Режим доступу до ресурсу : https://docs.hashflow.com/hashflow/taker/getting-started

17. Hashflow Set up authentication [Електронний ресурс] – Режим доступу до ресурсу : https://docs.hashflow.com/hashflow/taker/getting-started#1.-set-up-authentication

18. Binance API. Introduction [Електронний ресурс] – Режим доступу до ресурсу : https://binance-docs.github.io/apidocs/spot/en/#introduction

19. Welcome to python-binance v1.0.17 [Електронний ресурс] – Режим доступу до ресурсу : https://python-binance.readthedocs.io/en/latest/

20. Why Tether and Stablecoin USDT Have Become a Big Crypto Worry [Електронний ресурс] – Режим доступу до ресурсу : https://www.bloomberg.com/news/articles/2022-12-18/understanding-tether-and-its-usdt-stablecoin-and-why-regulators-are-worried

21. Telegram a new ara of messaging [Електронний ресурс] – Режим доступу до ресурсу : https://telegram.org

22. Приклад вдалого обміну на обізрівачі блокчейну BSC Bscscan [Електронний ресурс] – Режим доступу до ресурсу : https://bscscan.com/tx/0x1560553a4792dc71d5e2e0e1e369f6e3db814f84357fd06612e25005a6460ffe

23. Приклад купівлі BNB програмою на обізрівачі блокчейну BSC Bscscan [Електронний ресурс] – Режим доступу до ресурсу : https://bscscan.com/tx/0x754c7ae0fbed66ecfab69d77db7e955fb6722a5b1e265f5e578fcf998741452e

24. Приклад продажу BNB програмою на обізрівачі блокчейну BSC Bscscan [Електронний ресурс] – Режим доступу до ресурсу : https://bscscan.com/tx/0xed02b5382ad3f735c586451b8dcb4487c1841c65233e50993829f669d6a9a377

ДОДАТОК А
КОД ДОДАТКУ, ЩО ВИКОНУЄ МОНІТОРИНГ КУРСІВ СТЕЙБЛКОЇНІВ

```
import time
import aiohttp
import asyncio
from datetime import datetime
import telebot
from binance.client import Client

api_key = '*Токен telegram боту*'
user_id = '*Ідентифікатор користувача Telegram*'

BAPI_KEY = '*публічний ключ Binance API*'
BSECRET_KEY = '*приватник ключ Binance API*'
bot = telebot.TeleBot(token=api_key)
client = Client(BAPI_KEY, BSECRET_KEY)

#INPUT
usdInputBSC1inchAndHashflow = int(input("Stablecoin num for BSC : ") or
"102000")
usdInputPolygonHashflow = int(input("Stablecoin num for Crosschain (Hashflow) :
") or "30000")
usdcInputPolygon1inch = int(input("Stablecoin num for Polygon (1inch) : ") or
"20000")



#HASHFLOW
urlHashflow = 'https://api.hashflow.com/taker/v2/rfq'

headersHashflow = {
  "Authorization": "*Авторизаційний заголовок*"
  }

#1INCH
```

```
_url = 'https://pathfinder.1inch.io/v1.4/chain/56/router/v5/quotes'
_ends= '&walletAddress=*Адреса мого гаманця*&preset=maxReturnResult'
url1inchBscBusdUsdc =
f'{_url}?fromTokenAddress=0xe9e7cea3dedca5984780bafc599bd69add087d56&toTokenAdd
ress=0x8ac76a51cc950d9822d68b83fe1ad97b32cd580d&amount={usdInputBSC1inchAndHash
flow*pow(10,18)}&gasPrice=5000000000{_ends}'

...

url1inchBscUsdtBusd =
f'{_url}?fromTokenAddress=0x55d398326f99059ff775485246999027b3197955&toTokenAdd
ress=0xe9e7cea3dedca5984780bafc599bd69add087d56&amount={usdInputBSC1inchAndHash
flow*pow(10,18)}&gasPrice=5000000000{_ends}'

_url = 'https://pathfinder.1inch.io/v1.4/chain/137/router/v5/quotes'
url1inchPolygonUsdcBusd =
f'{_url}?fromTokenAddress=0x2791bca1f2de4661ed88a30c99a7a9449aa84174&toTokenAdd
ress=0x9c9e5fd8bbc25984b178fdce6117defa39d2db39&amount={usdcInputPolygon1inch*p
ow(10,6)}&gasPrice=624908927597{_ends}'

...

url1inchPolygonUsdtUsdc =
f'{_url}?fromTokenAddress=0xc2132d05d31c914a87c6611c10748aeb04b58e8f&toTokenAdd
ress=0x2791bca1f2de4661ed88a30c99a7a9449aa84174&amount={usdcInputPolygon1inch*p
ow(10,6)}&gasPrice=52777592881{_ends}'

headers1inch = {'if-none-
match':'W/"4140e0fbffd5dcb8aeb2866d6335923c"','Accept': 'application/json,
text/plain, */*', 'Referer': 'https://app.1inch.io/', 'User-Agent':
'Mozilla/5.0 (Linux; U; Android 6.0; zh-CN; Letv X501 Build/DBXCNOP5902303111S)
AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/40.0.2214.89
UCBrowser/11.5.2.942 Mobile Safari/537.36NULL', 'sec-ch-ua':'"Google
Chrome";v="107", "Chromium";v="107", "Not=A?Brand";v="24"', 'sec-ch-ua-
mobile':'?0','sec-ch-ua-platform':'"Windows"', 'origin':'https://app.1inch.io'}

#BSC
```

```
dataHashflowUsdcBusdBSC = {
"networkId":56,"source":"api","baseToken":"0x8ac76a51cc950d9822d68b83fe1ad97b32
cd580d",
"quoteToken":"0xe9e7cea3dedca5984780bafc599bd69add087d56","baseTokenAmount":str
(usdInputBSC1inchAndHashflow*pow(10,18)),"trader":"*Адреса мого
гаманця*","rfqType":0
  }

...

dataHashflowUsdtUsdtAvaxPolygon = {
"networkId":43114,"source":"api","baseToken":"0x9702230a8ea53601f5cd2dc00fdbc13
d4df4a8c7",
"quoteToken":"0xc2132d05d31c914a87c6611c10748aeb04b58e8f","baseTokenAmount":str
(usdInputPolygonHashflow*pow(10,6)), "trader":"*Адреса мого
гаманця*","dstNetworkId":137,"rfqType":0
}

#TEST
async def get_1inch(session_, url_, outputCurrencyDecimals_):
  async with session_.get(url = url_, headers = headers1inch) as response:
    try:
      jsonResponse_ = await response.json()
      return
float(jsonResponse_['bestResult']['toTokenAmount'])/10**outputCurrencyDecimals_
    except Exception as e:
      print(e)
      return 1.337

async def get_hashflow(session_, data_, outputCurrencyDecimals_):
  async with session_.post(url = urlHashflow, json = data_, headers =
headersHashflow) as response:
    try:
      jsonResponse_ = await response.json()
      return
float(jsonResponse_['quoteData']['quoteTokenAmount'])/10**outputCurrencyDecimal
s_
```

```
    except Exception as e:
      print(jsonResponse_['error']['message'])
      return 1.337

def difference(inputAmount_, outputAmountFormatted_):
  difference_ = round(float(outputAmountFormatted_) - float(inputAmount_),2)
  if difference_ > 0:
     return f" +{difference_}"
  return f" {difference_}"

def process_hashflow(network_, inputCurrency_ ,outputCurrency_, inputAmount_,
outputAmountFormatted_):
  flag_ = ''
  difference_ = difference(inputAmount_, outputAmountFormatted_)
  if float(difference_) > 2:
    flag_ = '!!!'
    bot.send_message(chat_id=user_id, text=f'⚠⚠⚠⚠⚠⚠⚠\nHashflow
{network_} {difference_}\n\n {inputCurrency_} ->
{outputCurrency_}\n\n{inputAmount_} {inputCurrency_} for
{outputAmountFormatted_} {outputCurrency_}')
  print(f'Hashflow {network_:15}| {inputAmount_:6} {inputCurrency_} ->
{outputAmountFormatted_:9} {outputCurrency_:3} |{difference_:10} |  {flag_:3}  |')

def process_1inch(network_, inputCurrency_ ,outputCurrency_, inputAmount_,
outputAmountFormatted_):
  flag_ = ''
  difference_ = difference(inputAmount_, outputAmountFormatted_)
  if float(difference_) > 3:
    flag_ = '!!!'
    bot.send_message(chat_id=user_id, text=f'⚠⚠⚠⚠⚠⚠⚠\n1inch
{network_} {difference_}\n\n {inputCurrency_} ->
{outputCurrency_}\n\n{inputAmount_} {inputCurrency_} for
{outputAmountFormatted_} {outputCurrency_}')
  print(f'1inch {network_:18}| {inputAmount_:6} {inputCurrency_} ->
{outputAmountFormatted_:9} {outputCurrency_:3} |{difference_:10} |  {flag_:3}  |')

async def main():
```

```
  async with aiohttp.ClientSession() as session:
    while True:
      try:
        bidBusdUsdt, askBusdUsdt, bidUsdcUsdt, askUsdcUsdt = 1, 1, 1, 1
        tickers = client.get_orderbook_tickers()
        for i in tickers:
          if i['symbol'] == 'BUSDUSDT':
            bidBusdUsdt = float(i['bidPrice'])
            askBusdUsdt = float(i['askPrice'])
          if i['symbol'] == 'USDCUSDT':
            bidUsdcUsdt = float(i['bidPrice'])
            askUsdcUsdt = float(i['askPrice'])

        now = datetime.now()
        print('\n', now.strftime("%H:%M:%S"))

        gatheredResultsFormatted = []
        tasks = []

        tasks.append(asyncio.ensure_future(get_1inch(session,
url1inchBscBusdUsdc, 18)))
      ...
        tasks.append(asyncio.ensure_future(get_1inch(session,
url1inchPolygonUsdcUsdt, 6)))

        #B
        tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowUsdcBusdBSC, 18)))
        ...
        tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowUsdtUsdcBSC, 18)))

        #B-P
        tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowBusdUsdcBSCPolygon, 6)))
        ...
        tasks.append(asyncio.ensure_future(get_hashflow(session,
```

```
dataHashflowUsdtUsdtBSCPolygon, 6)))

        #B-A
        tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowBusdUsdcBSCAvax, 6)))
        ...
        tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowUsdtUsdtBSCAvax, 6)))

        #P
        tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowUsdcUsdtPolygon, 6)))
        tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowUsdtUsdcPolygon, 6)))
        #P-B
        tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowUsdcUsdcPolygonBSC, 18)))
        ...
        tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowUsdtUsdtPolygonBSC, 18)))

        #P-A
        tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowUsdcUsdcPolygonAvax, 6)))
      ...
        tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowUsdtUsdtPolygonAvax, 6)))

        #A
        tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowUsdcUsdtAvax, 6)))
        tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowUsdtUsdcAvax, 6)))
        #A-B
        tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowUsdcBusdAvaxBsc, 18)))
        ...
```

```
        tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowUsdtUsdtAvaxBsc, 18)))

        #A-P
        tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowUsdcUsdcAvaxPolygon, 6)))
        ...
        tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowUsdtUsdtAvaxPolygon, 6)))

        gatheredResults = await asyncio.gather(*tasks)

        for result in gatheredResults:
          gatheredResultsFormatted.append(round(result,2))

        print('-----------------------------------------------------')
        print('{protocolNetworkColumn:24}| {inputAmount:6} {inputCurrency:5}
{outputAmountFormatted:7} {outputCurrency:5} |  {difference:5}  | {flag:9} |'
           .format(protocolNetworkColumn = "Protocol & Network", inputAmount =
"Input", inputCurrency = "Asset", outputAmountFormatted = "Output",
          outputCurrency = "Asset", difference = "Delta", flag = "Attention"))
        print('-----------------------------------------------------\n')
        print('-----------------------------------------------------')
        process_1inch('BSC', 'BUSD', 'USDC', usdInputBSC1inchAndHashflow,
gatheredResultsFormatted[0])

        ...
        process_hashflow('A-P Ut*b -> U/a', 'USDT', 'USDT',
int(round(usdInputPolygonHashflow/bidUsdcUsdt, 2)),
round(gatheredResultsFormatted[51]/askUsdcUsdt, 2))

        print('-----------------------------------------')
        time.sleep(6)
      except Exception as e:
        print(e)
        time.sleep(6)
asyncio.run(main())
```

ДОДАТОК Б
КОД ДОДАТКУ, ЩО ВИКОНУЄ МОНІТОРИНГ КУРСУ ТА АВТОМАТИЧНИЙ ОБМІН BNB

```
import time
import json
import aiohttp
import asyncio
from datetime import datetime
import hexbytes

from web3 import Web3
from web3.middleware import geth_poa_middleware

privKey = str(input("Input the private key from the EVM wallet with 10000 BUSD
and some BNB for gas fee: "))

w3 = Web3(Web3.HTTPProvider('*URL-адреса для підключення до провайдеру
інфраструктури блокчейну*'))
w3.middleware_onion.inject(geth_poa_middleware, layer=0)

hftContract = w3.eth.contract(
    address=w3.toChecksumAddress('0x0ACFFB0fb2cddd9BD35d03d359F3D899E32FACc9'),
    abi=[{"inputs": [{"components": [
        {"internalType": "address", "name": "pool", "type": "address"},
        {"internalType": "address", "name": "externalAccount", "type":
"address"},
        {"internalType": "address", "name": "trader", "type": "address"},
        {"internalType": "address", "name": "effectiveTrader", "type":
"address"},
        {"internalType": "address", "name": "baseToken", "type": "address"},
        {"internalType": "address", "name": "quoteToken", "type": "address"},
        {"internalType": "uint256", "name": "effectiveBaseTokenAmount", "type":
"uint256"},
        {"internalType": "uint256", "name": "maxBaseTokenAmount", "type":
"uint256"},
```

```
        {"internalType": "uint256", "name": "maxQuoteTokenAmount", "type":
"uint256"},
        {"internalType": "uint256", "name": "quoteExpiry", "type": "uint256"},
        {"internalType": "uint256", "name": "nonce", "type": "uint256"},
        {"internalType": "bytes32", "name": "txid", "type": "bytes32"},
        {"internalType": "bytes", "name": "signature", "type": "bytes"}],
        "internalType": "struct IQuote.RFQTQuote", "name": "quote", "type":
"tuple"
    }
    ], "name": "tradeSingleHop", "outputs": [], "stateMutability": "payable",
"type": "function"}])

urlHashflow = 'https://api.hashflow.com/taker/v2/rfq'

headersHashflow = {
    "Authorization": "*Авторизаційний заголовок*"
}


class QuoteArray(list):

    def __init__(self):
        self.best = self.WORSE.copy()

    def deleteExpired(self, maxSecondsLeft: float):
        listLen = len(self)
        i = 0
        j = 0
        nowTimestamp = datetime.now().timestamp()
        self.best = self.WORSE.copy()
        while i < listLen:
            if self[i]['quoteData']['quoteExpiry'] - maxSecondsLeft >=
nowTimestamp:
                self[j] = self[i]
                self._findBestPriceQuote(j)
                j += 1
```

```
            i += 1
        del self[j:]

    def append(self, el):
        if el['status'] == 'success':
            self._transform(el)
            super().append(el)

    @staticmethod
    def _transform(el):
        raise NotImplementedError

    def _findBestPriceQuote(self, i):
        raise NotImplementedError


class BuyQuoteArray(QuoteArray):
    WORSE = {'myRate': 10000}

    @staticmethod
    def _transform(el):
        el['myRate'] = float(el['quoteData']['baseTokenAmount']) /
float(el['quoteData']['quoteTokenAmount'])

    def _findBestPriceQuote(self, j):
        if self[j]['myRate'] < self.best['myRate']:
            self.best = self[j]


class SellQuoteArray(QuoteArray):
    WORSE = {'myRate': 0}

    @staticmethod
    def _transform(el):
        el['myRate'] = float(el['quoteData']['quoteTokenAmount']) /
float(el['quoteData']['baseTokenAmount'])
```

```
    def _findBestPriceQuote(self, j):
        if self[j]['myRate'] > self.best['myRate']:
            self.best = self[j]


def buildAndSendTx(answer, effectiveAmount, traderNonce):
    eoa = '0x0000000000000000000000000000000000000000'
    if 'eoa' in answer['quoteData']:
        eoa = w3.toChecksumAddress(answer['quoteData']['eoa'])

    value = 0
    if answer['quoteData']['baseToken'] ==
'0x0000000000000000000000000000000000000000':
        value = int(effectiveAmount)

    tx = hftContract.functions.tradeSingleHop(
        {
            'pool': w3.toChecksumAddress(answer['quoteData']['pool']),
'externalAccount': eoa,
            'trader': w3.toChecksumAddress(answer['quoteData']['trader']),
            'effectiveTrader':
w3.toChecksumAddress(answer['quoteData']['trader']),

            'baseToken':
w3.toChecksumAddress(answer['quoteData']['baseToken']),
            'quoteToken':
w3.toChecksumAddress(answer['quoteData']['quoteToken']),

            'effectiveBaseTokenAmount': int(effectiveAmount),
            'maxBaseTokenAmount': int(answer['quoteData']['baseTokenAmount']),
            'maxQuoteTokenAmount':
int(answer['quoteData']['quoteTokenAmount']),

            'quoteExpiry': int(answer['quoteData']['quoteExpiry']), 'nonce':
int(answer['quoteData']['nonce']),

            'txid': hexbytes.HexBytes(answer['quoteData']['txid']),
```

```
            'signature': hexbytes.HexBytes(answer['signature'])
        }
    ).build_transaction(
        {'nonce': traderNonce, 'from':
w3.toChecksumAddress(answer['quoteData']['trader']), 'gas': 170000,
         'gasPrice': Web3.toWei(5, 'gwei'), 'value': value})
    print(tx)
    tx_create = w3.eth.account.sign_transaction(tx, privKey)
    tx_hash = w3.eth.send_raw_transaction(tx_create.rawTransaction)
    tx_receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
    print(f"Transaction successful with hash: {tx_hash.hex()}")



async def get_hashflow(session_, data_):
    async with session_.post(url=urlHashflow, json=data_,
headers=headersHashflow) as response:
        try:
            jsonResponse_ = await response.json()
            if 'error' in jsonResponse_:
                print(jsonResponse_)
                return {'status': 'fail'}
            return jsonResponse_

        except Exception as e:
            print(jsonResponse_)
            return {'status': 'fail'}


async def main():
    async with aiohttp.ClientSession() as session:
        buys = BuyQuoteArray()
        sells = SellQuoteArray()
        while True:
            try:
                dataHashflowBUSDBNBBSC = {
                    "networkId": 56, "source": "api", "baseToken":
```

```
"0xe9e7cea3dedca5984780bafc599bd69add087d56",
                    "quoteToken": "0x0000000000000000000000000000000000000000",
                    "baseTokenAmount": "1000000000000000000000", "trader":
"*Адреса мого гаманця*", "rfqType": 0
                }

                dataHashflowBNBBUSDBSC = {
                    "networkId": 56, "source": "api", "baseToken":
"0x0000000000000000000000000000000000000000",
                    "quoteToken": "0xe9e7cea3dedca5984780bafc599bd69add087d56",
                    "baseTokenAmount": "3300000000000000000", "trader":
"*Адреса мого гаманця*", "rfqType": 0
                }

                tasks = []
                tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowBUSDBNBBSC)))  # BUSD --> BNB
                tasks.append(asyncio.ensure_future(get_hashflow(session,
dataHashflowBNBBUSDBSC)))  # BNB  --> BUSD

                res = await asyncio.gather(*tasks)

                buys.append(res[0])
                sells.append(res[1])

                buys.deleteExpired(8)
                sells.deleteExpired(8)

                bestBuyRateQuote = buys.best
                bestSellRateQuote = sells.best

                if bestBuyRateQuote['myRate'] < bestSellRateQuote['myRate']:
                    if int(bestBuyRateQuote['quoteData']['quoteExpiry']) < int(
                            bestSellRateQuote['quoteData']['quoteExpiry']):
                        myNonce = w3.eth.get_transaction_count(

w3.toChecksumAddress(bestBuyRateQuote['quoteData']['trader']))
```

```
                        buildAndSendTx(bestBuyRateQuote,
bestBuyRateQuote['quoteData']['baseTokenAmount'], myNonce)
                        buys.remove(bestBuyRateQuote)
                        sells.remove(bestSellRateQuote)

                        buyQTA =
bestBuyRateQuote['quoteData']['quoteTokenAmount']
                        dataHashflowBNBBUSDBSC['baseTokenAmount'] = str(
                            int(buyQTA) // (10 ** (len(buyQTA) - 4)) * (10 **
(len(buyQTA) - 4)))

                        while bestSellRateQuote['quoteData']['quoteExpiry'] - 5
>= datetime.now().timestamp():
                            tasks = []
                            tasks.append(
                                asyncio.ensure_future(get_hashflow(session,
dataHashflowBNBBUSDBSC)))  # BNB  --> BUSD
                            res = await asyncio.gather(*tasks)
                            newSellQuote = res[0]
                            if newSellQuote['status'] == 'success':
                                newSellQuote['myRate'] =
float(newSellQuote['quoteData']['quoteTokenAmount']) / float(

newSellQuote['quoteData']['baseTokenAmount'])
                                if newSellQuote['myRate'] >
bestSellRateQuote['myRate'] and newSellQuote['quoteData'][
                                        'quoteExpiry'] - 5 >=
datetime.now().timestamp():
                                    bestSellRateQuote = newSellQuote
                            time.sleep(2)

                        buildAndSendTx(bestSellRateQuote,
                                       str(int(buyQTA) // (10 ** (len(buyQTA) -
4)) * (10 ** (len(buyQTA) - 4))),
                                       myNonce + 1)

                nowTimetamp = datetime.now().timestamp()
```

```
                print(len(buys), bestBuyRateQuote['quoteData']['quoteExpiry'] -
nowTimetamp)
                print(len(sells), bestSellRateQuote['quoteData']['quoteExpiry']
- nowTimetamp)
                print("Best buying rate:", bestBuyRateQuote['myRate'])
                print("Best selling rate:", bestSellRateQuote['myRate'], "\n")
                time.sleep(1)
            except Exception as e:
                print(e)
                time.sleep(4)


asyncio.run(main())
```