Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка методу та програмних засобів автоматизованої системи моніторингу та продажу NFT на маркетплейсах OpenSea та Blur»

Виконав: студент 4 курсу
групи 1ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Коберник М.В.
(прізвище та ініціали)
Керівник: к.т.н., доцент каф. ПЗ Войтко В.В.
(прізвище та ініціали)
Рецензент: к.т.н., доцент каф. ОТ Городецька О. С.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«06» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
« 21 » березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Кобернику Михайлу Володимировичу

1. Тема роботи – «Розробка методу та програмних засобів автоматизованої системи моніторингу та продажу NFT на маркетплейсах OpenSea та Blur»

Керівник роботи: Войтко Вікторія Володимирівна, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025 р.

3. Вихідні дані до роботи: тип програми – десктопний застосунок; модель розробки – інкрементна; засоби організації даних програми – бібліотеки curl-cffi, loguru, openpyxl, pandas, eth-account, web3, aiogram; вхідні дані: приватний ключ, користувацькі налаштування; вихідні дані: інтерфейс користувача, логи та стани, транзакції ; середовище розробки – PyCharm; мова програмування – Python.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз стану розвитку систем NFT торгівлі; розробка методів та алгоритмів програмного застосунку; розробка застосунку; тестування застосунку; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; об'єкт та предмет дослідження; задачі дослідження, наукова новизна, практ цінність одержаних результатів; порівняльний аналіз аналогів, розробка ме асинхронної черги з rate-limiting на рівні HTTP-методів; розробка методу розум добору пріоритетної плати за газ при прийнятті біду; використані техно тестування системи; висновки; апробація результатів роботи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата завдання видав	Підпис, дата завдання прийняв
1-4	Войтко В.В., к.т.н., доцент кафедри ПЗ	24.03.25	01.[illegible]

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітки
1	Аналіз стану розвитку систем NFT торгівлі	25.03.25 - 05.04.25	Виконано
2	Розробка методів та алгоритмів програмного застосунку	06.04.25 - 18.04.25	Виконано
3	Варіантний аналіз та вибір мови програмування розробки	19.04.25 - 21.04.25	Виконано
4	Розробка застосунку	22.04.25 - 17.05.25	Виконано
5	Тестування застосунку	18.05.25 - 25.05.25	Виконано
6	Оформлення матеріалів до захисту БКР	26.05.25 - 30.05.25	Виконано

Студент _____ Коберник М.[illegible]
(підпис) (прізвище та [illegible])

Керівник бакалаврської кваліфікаційної роботи _____ Войтко В.В.
(підпис) (прізвище та [illegible])

АНОТАЦІЯ

УДК 004:005.9

Коберник М.В. Розробка методу та програмних засобів автоматизованої системи моніторингу та продажу NFT на маркетплейсах OpenSea та Blur : бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця : ВНТУ, 2025. 85 с.

У бакалаврській кваліфікаційній роботі були розроблені методи та програмні засоби автоматизованої системи моніторингу та продажу NFT на маркетплейсах OpenSea та Blur. Розроблені засоби дозволяють отримувати розгорнуту інформацію про колекції NFT та поточні лістинги, приймати найвигідніші пропозиції на продаж з налаштовуваним підбором плати за газ, отримувати й використовувати поточні ціни газу, зберігати та відновлювати конфігурацію NFT.

Було розроблено інформаційне забезпечення системи, продемонстровано, які дані та як використовуються при автоматизованій торгівлі NFT предметами. Розроблено модель системи, наведено компоненти, що використовуються у системі. Розроблено алгоритми роботи системи, метод асинхронної черги з rate-limiting на рівні HTTP-методів та метод розумного добору пріоритетної плати за газ при прийнятті біду.

Програмне забезпечення розроблено із використанням середовища розробки PyCharm та мови програмування Python. У процесі розробки використовувались сучасні бібліотеки та інструменти, такі як curl-cffi, loguru, openpyxl, pandas, eth-account, web3, aiogram. Під час тестування було перевірено працездатність основного функціоналу системи та підтверджено коректність реалізованих методів.

Ключові слова: автоматизована система, NFT, аналіз.

ABSTRACT

UDC 004:005.9

Kobernyk M.V. Development of a method and software tools for an automated system for monitoring, buying and selling NFTs on the OpenSea and Blur marketplaces: bachelor's qualification work in the specialty 121 Software Engineering, educational program - Software Engineering. Vinnytsia: VNTU, 2025. 85 p.

In the bachelor's qualification work, methods and software tools for an automated system for monitoring, buying and selling NFTs on the OpenSea and Blur marketplaces were developed. The developed tools allow you to receive detailed information about NFT collections and current listings, accept the most profitable offers for sale with customizable selection of gas fees, receive and use current gas prices, save and restore the NFT configuration.

During the work, the information support of the system was developed, it was demonstrated what data and how it is used in automated trading of NFT items. A system model has been developed, the components used in the system are given. The algorithms of the system operation, the method of an asynchronous queue with rate-limiting at the HTTP method level and the method of intelligent selection of priority gas payment when accepting a problem have been developed.

The software was developed using the PyCharm development environment and the Python programming language. Modern libraries and tools such as curl-cffi, loguru, openpyxl, pandas, eth-account, web3, aiogram were used in the development process. During testing, the operability of the main functionality of the system was checked and the correctness of the implemented methods was confirmed.

Keywords: automated system, NFT, analysis.

ЗМІСТ

ВСТУП.....3
1 АНАЛІЗ СТАНУ РОЗВИТКУ СИСТЕМ NFT ТОРГІВЛІ.....6
1.1 Аналіз стану питання розробки систем NFT торгівлі.....6
1.2 Порівняльний аналіз аналогів.....7
1.3 Аналіз методів розв’язання задачі.....10
1.4 Постановка задач дослідження.....12
1.5 Висновки.....13
2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ АВТОМАТИЗОВАНОЇ СИСТЕМИ NFT ПРОДАЖІВ НА МАРКЕТПЛЕЙСАХ.....14
2.1 Розробка інформаційного забезпечення системи.....14
2.2 Розробка моделі системи.....15
2.3 Розробка методу асинхронної черги з rate-limiting на рівні HTTP-методів…17
2.4 Розробка методу розумного добору пріоритетної плати за газ при прийнятті біду.....19
2.5 Розробка алгоритмів роботи системи.....22
2.6 Висновки.....24
3 РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ ТОРГІВЛІ NFT.....25
3.1 Варіантний аналіз та обґрунтування вибору мови програмування.....25
3.2 Варіантний аналіз та обґрунтування вибору середовища розробки.....28
3.3 Розробка модуля бізнес-логіки та взаємодії з Blur API.....31
3.4 Розробка модуля графічного інтерфейсу та управління налаштуваннями NFT 41
3.5 Висновки.....47
4 ТЕСТУВАННЯ СИСТЕМИ.....48
4.1 Аналіз методів тестування.....48
4.2 Тестування системи.....49
4.3 Висновки.....55
ВИСНОВКИ.....56
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....58
ДОДАТКИ.....60
Додаток А. Технічне завдання.....61
Додаток Б. Протокол перевірки на наявність текстових запозичень.....64
Додаток В. Лістинг програми.....65
Додаток Г. Ілюстративна частина.....86

ВСТУП

Обґрунтування вибору теми дослідження. Торгівля NFT предметами перебуває у стадії активного розвитку з моменту свого заснування у 2021 році, та, попри певну волатильність, продовжує демонструвати значний ріст, досягши рівня обсягу торгівлі у мільярди доларів щорічно [1]. Щоденно проводяться тисячі транзакцій, що є значним обсягом торгівлі, тому, для прийняття успішних рішень щодо покупки чи продажу предметів NFT необхідний не лише якісний, а і швидкий аналіз, який стає уже неможливо провести вручну, без використання автоматизованих систем, які проводитимуть аналіз в автоматичному режимі за короткий проміжок часу. Такі автоматизовані системи моніторингу та торгівлі надають своїм користувачам значну конкурентну перевагу, дозволяючи оперативно виявляти потенційно прибуткові та цінні активи, реагувати на зміни цін та використовувати можливості арбітражу між різними платформами для торгівлі [2]. Тому актуальною є розробка такої системи. Особливо актуальною вона є з огляду на те, що ринок NFT переходить стадію розвитку з спекулятивного до екосистеми з різноманітним реально корисним застосуванням. NFT можна використовувати не лише як цифрове мистецтво, а й як функціональні активи в метавсесвітах та ігрових системах, де все більше використовуються NFT токени для залучення гравців та підвищення мотивації грати ігри, адже під час гри можна отримувати NFT предмети, які мають реальну вартість, та на продажу яких можна заробити реальні гроші, що трансформує всю сферу ігор із джерела витрат грошей на джерело їх заробітку для найуспішніших гравців.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є удосконалення процесу торгівлі NFT на маркетплейсах OpeanSea та Blur шляхом розробки спеціалізованої автоматизованої системи, яка дозволяє оптимізувати процеси моніторингу та торгівлі NFT.

Основними задачами дослідження є:

- розробити архітектуру автоматизованої системи NFT продажів на маркетплейсах;

- розробити алгоритми роботи автоматизованої системи NFT продажів на маркетплейсах;

- розробити функціонал системи;

- розробити інтерфейс користувача застосунку;

- провести тестування розробленої системи.

Об’єктом дослідження є процес розробки програмних засобів автоматизованої системи моніторингу та продажу NFT на маркетплейсах OpenSea та Blur.

Предметом дослідження є методи і засоби реалізації автоматизованої системи моніторингу та продажу NFT на маркетплейсах OpenSea та Blur.

Методи дослідження. У процесі досліджень використовувались такі методи дослідження:

- методи роботи систем NFT торгівлі для аналізу функціонування маркетплейсів, оптимізації лістингів і підпису транзакцій;

- методи реалізації архітектури програмних систем для побудови структури застосунку;

- методи розрахунки ціни «газу» для оптимізації витрат на транзакції;

- методи реалізації інтерфейсу користувача для забезпечення зручності користування;

- методи тестування програмних систем для перевірки стабільності та коректності роботи всіх модулів програми.

Наукова новизна отриманих результатів.

1. Подальшого розвитку отримав метод асинхронної черги з rate-limiting на рівні HTTP-методів, який, на відміну від відомих, гарантує, що запити одного і того ж методу не перевищують заданого ліміту на одиницю часу, що забезпечує стабільну роботу в умовах мінливих лімітів і ненадійних проксі без втручання розробника.

2. Подальшого розвитку отримав метод розумного добору пріоритетної плати за газ при прийнятті біду, який, на відміну від відомих, дозволяє обрати таку величину, при якій очікуваний чистий прибуток від продажу не опуститься нижче заданого мінімального рівня.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в можливості використання розроблених програмних засобів для автоматизованого моніторингу та продажу NFT.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. У роботі [3], виконаній у співавторстві, автору належать: модель системи, порівняльний аналіз аналогів.

Апробація результатів роботи. Описані у бакалаврській кваліфікаційній роботі положення доповідались на конференції «LIV Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2025)».

Публікації. Результати роботи були опубліковані в матеріалах конференції «LIV Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2025)» [3].

Аналіз. У пояснювальній записці до бакалаврської кваліфікаційної роботи було розглянуто 4 розділи та було використано 20 літературних джерел.

1 АНАЛІЗ СТАНУ РОЗВИТКУ СИСТЕМ NFT ТОРГІВЛІ

1.1 Аналіз стану питання розробки систем NFT торгівлі

У сучасних умовах ринок невзаємозамінних токенів (NFT) стрімко зростає: протягом останніх років загальний обсяг торгів на провідних маркетплейсах збільшився в кілька разів, а кількість активних користувачів подвоїлася [4]. При цьому волатильність цін та високий темп появи нових колекцій створюють умови, коли ручне керування лістингом і ціноутворенням втрачає ефективність. Саме тому розробка автоматизованої системи NFT-продажів стає вкрай актуальною — вона дозволяє власникам токенів оперативно реагувати на ринкові зміни, підтримувати конкурентоспроможний рівень ціни та мінімізувати втрати внаслідок простою активів.

Першим етапом створення такої системи є глибока інтеграція з API обраних маркетплейсів (OpenSea, Magic Eden, Blur тощо). Наявність REST- та WebSocket-інтерфейсів дає змогу оперативно отримувати дані про нові поступлення колекцій, зміни флор-пресів і статуси транзакцій. За рахунок використання Web3-бібліотек (web3.js, ethers.js, web3.py) система здатна підписувати транзакції й безпосередньо взаємодіяти зі смартконтрактами — що критично для автоматичного лістингу, релістингу чи delisting-операцій згідно з заздалегідь заданими умовами.

Система NFT продажів на маркетплейсах повинна володіти таким функціоналом:

1. Моніторинг ринку — агрегування інформації з API маркетплейсів та зовнішніх аналітичних платформ для прогнозування коливань попиту.

2.Прийняття рішень — реалізація адаптивних стратегій ціноутворення (floor-tracking, динамічна корекція ставки залежно від активності інших учасників ринку).

3.Виконання транзакцій — формування, оптимізація вартості газу й підпис транзакцій через безпечні Web3-гаманці.

4.Логування та сповіщення — зберігання історії операцій у базах даних і

надсилання повідомлень про статус кожної дії.

Незважаючи на переваги, розробка стикається з низкою технічних і регуляторних викликів:

1. Затримки підтвердження транзакцій через конкуренцію за включення до блоку можуть призводити до переплати газу або до втрати вигідних позицій лістингу.

2. Нестабільність API — періодичні оновлення інтерфейсів маркетплейсів вимагають гнучкої архітектури й швидкого реагування на зміни.

3. Правові обмеження — у деяких юрисдикціях автоматизована торгівля NFT може підпадати під вимоги фінансового регулювання, що вимагає додаткових процедур звітності та ліцензування.

Таким чином, актуальність розробки автоматизованої системи NFT-продажів обумовлена не лише стрімким зростанням ринку та високою динамікою цін, а й потребою в ефективному управлінні активами, здатному забезпечити швидкість, безпеку та конкурентні переваги в умовах насиченого і мінливого середовища маркетплейсів.

1.2 Порівняльний аналіз аналогів

Розглянемо існуючі рішення автоматизованих систем для моніторингу та торгівлі NFT:

1. NFT Sniper Bot.

2. Nansen Portfolio.

3.Auto Listing Engine від NFT Butler.

NFT Sniper Bot [5] - це бот, що реалізований в інтерфейсі командного рядка, який дозволяє автоматично виявляти та купувати NFT предмети одразу після їх виходу на продаж. Бот підключається до блокчейнів Ethereum, Solana та інших через API інтерфейси, виконує транзакції на основі встановлених користувачем параметрів. Користувач може налаштувати бота для пошуку конкретних колекцій, цінових діапазонів або рідкісних характеристик.

Інтерфейс NFT Sniper Bot наведено на рисунку 1.1.

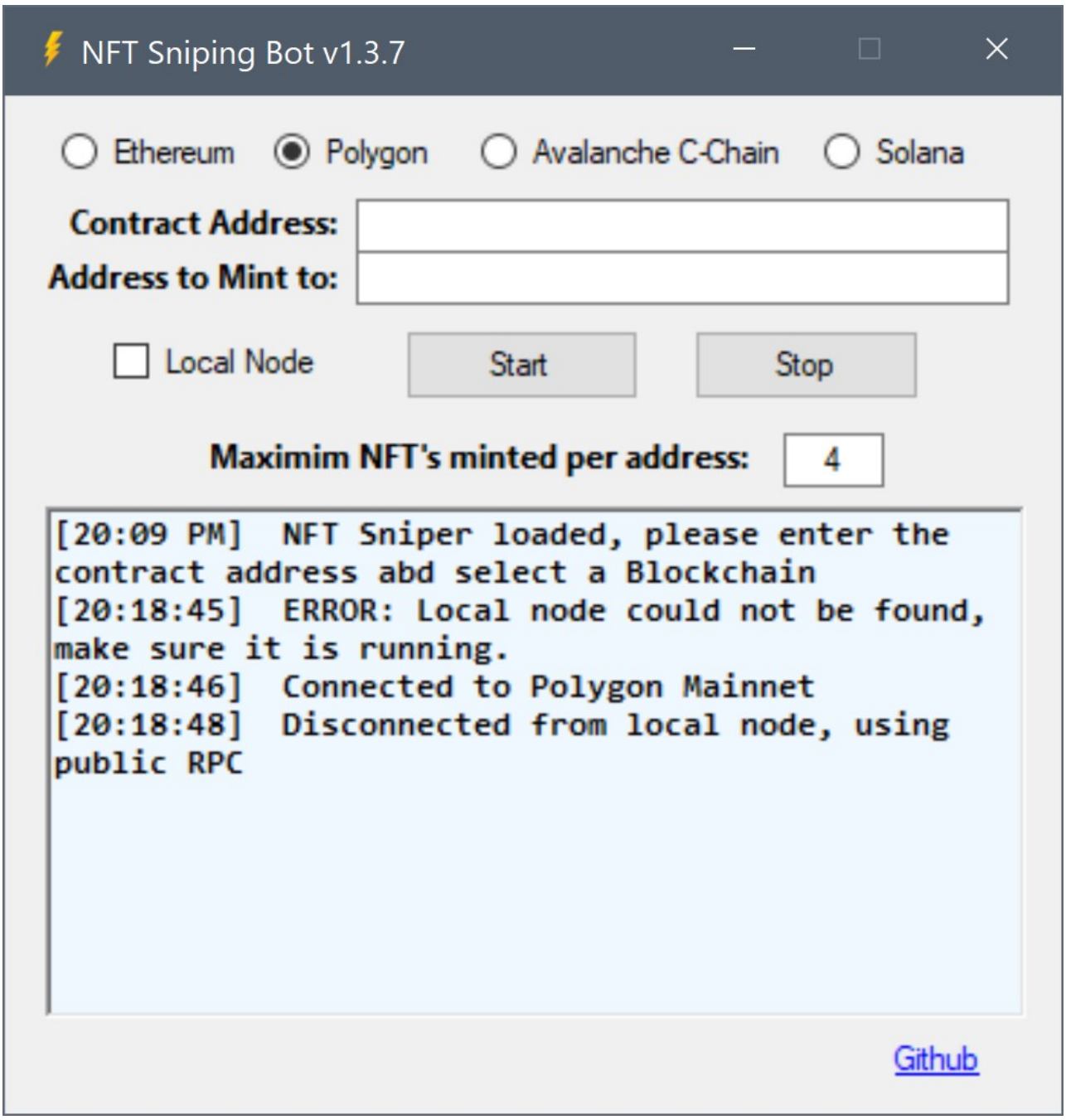


Рисунок 1.1 — Інтерфейс NFT Sniper Bot

Nansen Portfolio [6] - аналітичний інструмент з функціями автоматизованої торгівлі. Окрім відстеження активності відомих гаманців та загальних ринкових тенденцій, Nansen пропонує функціонал для автоматичного виконання транзакцій на основі виявлених трендів. Система інтегрується з основними NFT маркетплейсами, включаючи OpenSea та Blur, і дозволяє користувачам створювати складні торгові стратегії на основі on-chain даних та аналізу соціальних мереж.

Інтерфейс Nansen Portfolio наведено на рисунку 1.2.

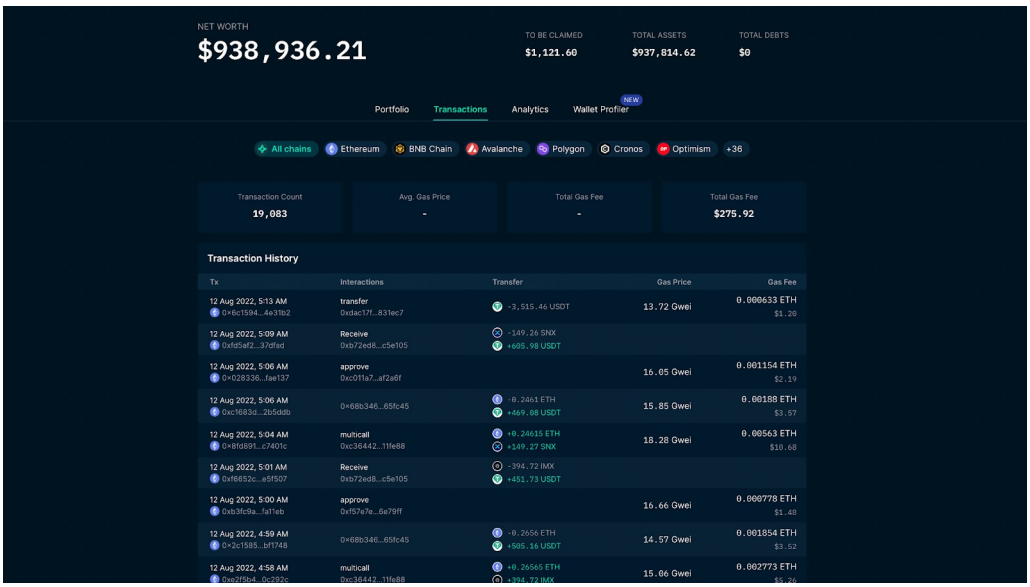


Рисунок 1.2 — Інтерфейс Nansen Portfolio

Auto Listing Engine від NFT Butler [7] — це високопродуктивне рішення для автоматизації процесу публікації та управління NFT-активами на провідних маркетплейсах. Інтеграція з OpenSea, Magic Eden та Blur забезпечує єдину точку керування всім життєвим циклом лістингу: від налаштування цінової політики до коригування роялті та метаданих.

Інтерфейс Auto Listing Engine від NFT Butler наведено на рисунку 1.3.

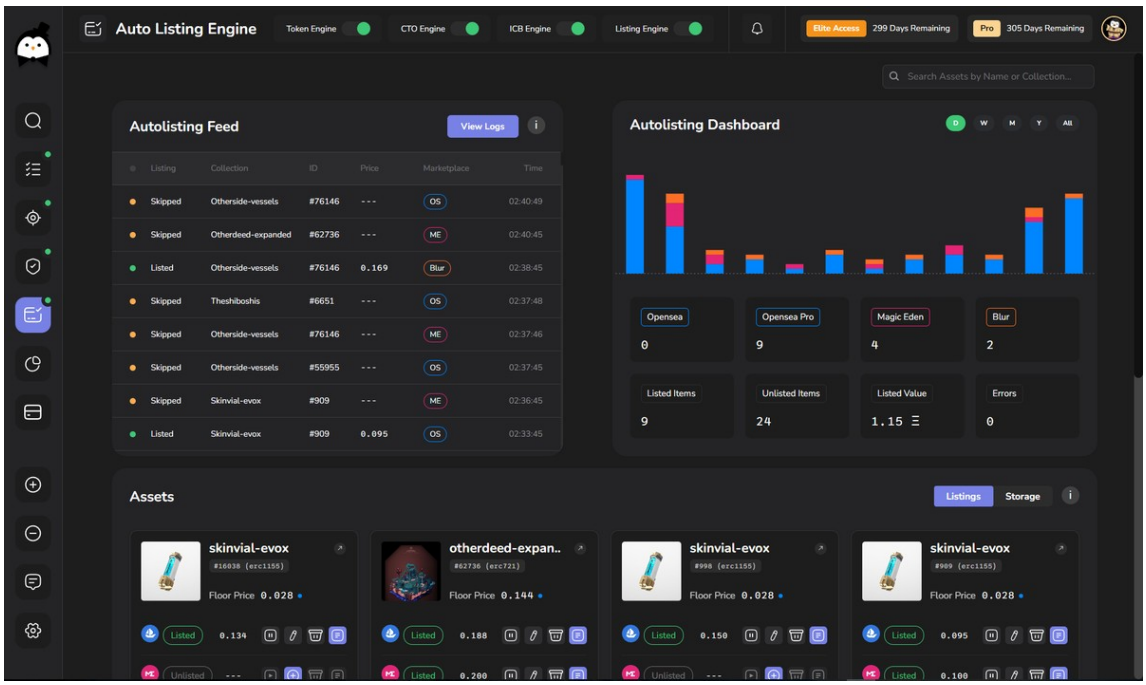


Рисунок 1.3 — Інтерфейс Auto Listing Engine від NFT Butler

Для демонстрації основних можливостей та відмінності між розглянутими застосунками, їхніх переваг та недоліків, було сформовано таблицю порівняння (таблиця 1.1).

Таблиця 1.1 – Порівняльний аналіз аналогів

Характеристика	NFT Sniper Bot	Nansen Portfolio	Auto Listing Engine	Власна розробка
Інтеграція з OpenSea	+	+	-	+
Інтеграція з Blur	+	+	+	+
Автоматичне виконання транзакцій	+	-	+	+
Підтримка найпопулярніших блокчейнів	+	+	-	+
Налаштування ризиків	-	-	-	+
Звітність	-	+	+	+
Аналіз рідкісних NFT предметів	+	+	+	+
Сумарний бал	5	5	4	7

Таким чином, власна розробка має вищий сумарний бал порівняно з NFT Sniper Bot та Nansen Portfolio на ~28.5% (100%-5/7*100% = 28.5%), за Blur Bidder на ~42.9% (100%-4/7*100% = 42.9%). Отже, власна розробка є актуальною. Її перевагами є більш комплексний функціонал порівняно з аналогами, що надає ширші можливості для якісного аналізу ринку NFT предметів та успішної торгівлі, а також наявність функціоналу для налаштування ризиків, які вберігають від втрати значних обсягів коштів за умови невдалої покупки.

1.3 Аналіз методів розв’язання задачі

Основними є такі підходи до розробки автоматизованих систем NFT-торгівлі:

1. Готові API інтерфейси з використанням OpenSea та Blur, що значно спрощує розробку, адже таким чином використовуються майже готові рішення [8]. Однак, таке рішення залежить від сторонніх рішень, які розробляються іншими розробниками, внесення змін в які можуть спричинити неправильну роботу системи. Також значним обмеженням є кількість запитів на використання, адже розробники таких API інтерфейсів витрачають ресурси на те, що інші користувачі використовують їхні API, тому встановлюють обмеження на використання.

2. Прямий моніторинг блокчейнів, який передбачає відстеження транзакцій безпосередньо з блокчейнів Ethereum, Polygon та Solana, що дозволяє отримувати необхідну інформацію в режимі онлайн [9]. Недоліком такого рішення є необхідність значних навичок у сфері блокчейну задля того, аби розробити правильно працюючий застосунок.

3. Індексація та агрегація даних через сторонні провайдери. Використання таких сервісів, як The Graph (субграфи), Alchemy, Covalent чи Moralis для побудови власних індексаторів дозволяє отримувати структурувані дані про NFT (токени, події продажу, зміну власника) без необхідності обробляти кожен блок вручну [10]. До переваг цього методу можна віднести миттєвий доступ до історії івентів, фільтрація за будь-якими параметрами (колекція, адреса, ціна), та зручні SDK та GraphQL-інтерфейс, що скорочує час на розробку.

Недоліками цього методу є залежність від сторонніх провайдерів — можливі обмеження на безкоштовний тариф, ризик зміни API та потенційні затримки оновлення даних при відставанні індексації.

4. Автоматизація на рівні смарт-контрактів із Gelato / Chainlink Keepers Замість клієнт-сайд ботів можна писати власні смарт-контракти, які за розкладом або при настанні певної умови (наприклад, ціна впала нижче заданої) самостійно викликають функції маркетплейсу. Для цього підключають оркестраторів Gelato чи Chainlink Keepers, що моніторять блокчейн і тригерять транзакції.

До переваг цього методу належать:

- повністю децентралізована логіка торгівлі — немає необхідності тримати свій сервер у мережі 24/7;

- мінімальний час реакції (транзакція ініціюється безпосередньо в мережі).

До недоліків можна віднести складність розробки та відладки смарт-контрактів, ризик багів і вразливостей та додаткові витрати на газ при кожному виклику автоматизації.

5. Flashbots [11] та MEV-стратегії для уникнення фронт-ранінгу Flashbots дозволяють об’єднати вашу транзакцію з іншими в приватному пулі, щоб запобігти передбачуваному фронт-ранінгу чи цензурі в публічному мемпулі. Автоматизовані боти можуть формувати bundle-транзакції для купівлі або снайпінгу лістингу без ризику, що інші боти випередять вас.

До переваг цього методу входить захист від передбачуваного фронт-ранінгу, більш стабільні цінові умови, можливість конкурентніших «снайперських» стратегії.

Недоліком же є обмежена підтримка блокчейнів та маркетплейсів, а також необхідність у додатковій інфраструктурі, такій як налаштування приватних RPC, та робота з Flashbots RPC.

Метод із готовими API інтерфейсами, такими як OpenSea чи Blur, забезпечує найшвидший шлях від ідеї до робочого рішення. Завдяки готовим SDK, документації та підтримці спільноти можна зосередитися на бізнес-логіці й унікальних функціях системи NFT-торгівлі, а не витрачати час на низькорівневий парсинґ блокчейну чи конфігурування інфраструктури. Тому було прийнято рішення реалізувати систему з використанням цього методу.

1.4 Постановка задач дослідження

Провівши аналіз методів розв’язання поставленої задачі, аналіз стану автоматизованих систем NFT продажів на маркетплейсах, порівняння аналогів було поставлено такі задачі дослідження:

- розробити архітектуру автоматизованої системи NFT продажів на маркетплейсах;

- розробити алгоритми роботи автоматизованої системи NFT продажів на маркетплейсах;

- розробити метод асинхронної черги з rate-limiting на рівні HTTP-методів;
- розробити метод розумного добору пріоритетної плати за газ при прийнятті біду;
- розробити функціонал системи;
- розробити інтерфейс користувача застосунку;
- провести тестування розробленої системи.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі було проведено аналіз стану питання розробки застосунків для автоматизованої торгівлі NFT, проведено порівняльний аналіз з такими застосунками: NFT Sniper Bot, Nansen Portfolio, Auto Listing Engine від NFT Butler. Було продемонстровано переваги власної розробки. Проведено аналіз методів розв’язання задачі, обрано метод з використанням готових API інтерфейсів. Проведено постановку задач дослідження.

2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ АВТОМАТИЗОВАНОЇ СИСТЕМИ NFT ПРОДАЖІВ НА МАРКЕТПЛЕЙСАХ

2.1 Розробка інформаційного забезпечення системи

Інформаційне забезпечення систему побудоване навколо взаємодії з двома головними зовнішніми джерелами даних: REST-API Blur (сендвіч запитів /auth, /v1/collections, /v1/orders тощо) та Ethereum-нода через AsyncWeb3. Вся комунікація з Blur відбувається через уніфіковану функцію _send_request, яка абстрагує HTTP-методи, параметри запитів, підписані cookies і проксі. Це гарантує, що всі необхідні для бізнес-логіки дані (інформація про колекції, біди, лістинги, токени, позики) спочатку отримуються у вигляді JSON від API, а потім трансформуються у внутрішні об’єкти (Bid, Listing).

Дані, що сильно не змінюються в межах одного запуску, кешуються локально в пам’яті:

- у словниках self.collection_addresses і self.collection_fees зберігаються адреси і мінімальні роялті колекцій, щоб уникнути дублювання запитів після першого отримання;

- у self._cookies['authToken'] тримається JWT-токен для аутентифікації; перевірка його валідності й автоматичне оновлення через login() гарантують, що виклики приватних ендпоінтів завжди мають коректний токен.

Для організації інформаційного потоку запити ранжуються у пріоритетні черги за HTTP-методами, а між відправками контролюється інтервал sleep_time. Це захищає від перевищення лімітів Blur API та допомагає акуратно розподілити навантаження між наявними проксі. Одночасно ProxyQueue стежить за «вантажем» кожного proxy-серверу й передає в запит найменш використовуваний, що підвищує стійкість до блокувань чи збоїв окремих проксі.

Усі дані, отримані у відповідь на запити, проходять агрегацію й нормалізацію:

- у методі _get_bids початковий масив priceLevels перетворюється на список Bid, потім групується за ціною з підсумовуванням обсягів;

- у get_listings відбувається пагінація: токени з кожної сторінки додаються в

єдиний масив Listing з інформацією про власника, ціну та підозрілість.

Завдяки такому інформаційному забезпеченню інформація в аплікації постійно оновлюється «по вимозі», але без зайвого дублювання мережевих викликів, — при цьому забезпечуються механізми автоматичного повтору й самоодужання (retry при 401/403/429/timeout), що робить клієнт стійким до коливань мережі та зовнішніх API.

2.2 Розробка моделі системи

На основі обраного функціоналу, було обрано інструменти для реалізації системи, та розроблено перелік модулів системи, які були зведені до діаграми компонентів.

Для роботи з API-маркетплейсами Blur та OpenSea обрано бібліотеку curl-cffi, що забезпечує асинхронні HTTP-запити з автоматичним керуванням сесіями, повторними спробами та таймаутами для стабільної та ефективної взаємодії. Модуль Logging покладено на loguru, яка підтримує багаторівневу фільтрацію повідомлень, ротацію лог-файлів за розміром або часом та зручний формат виводу для швидкого аналізу подій.

Модуль Analytics поєднує можливості pandas для обробки та попереднього аналізу даних із openpyxl для експорту зведених звітів у форматі Excel: завдяки цьому можна формувати адаптивні таблиці та легко підключати їх до BI-систем. Для безпосередньої роботи з блокчейном Ethereum використовуємо компонент web3, що дозволяє здійснювати виклики смарт-контрактів, опитувати стан мережі та відправляти транзакції з урахуванням gas-налаштувань.

Щоб інформувати користувачів про ключові події (успішні угоди, помилки при викликах API тощо), реалізовано Telegram-бота на базі aiogram, який підтримує inline-клавіатури та обробку команд у реальному часі. Нарешті, модуль Trading координує стратегії купівлі/продажу NFT: він формує заявки, аналізує книгу ордерів і забезпечує своєчасне виконання угод відповідно до обраної торгової стратегії.

Розроблену діаграму компонентів наведено на рисунку 2.1.

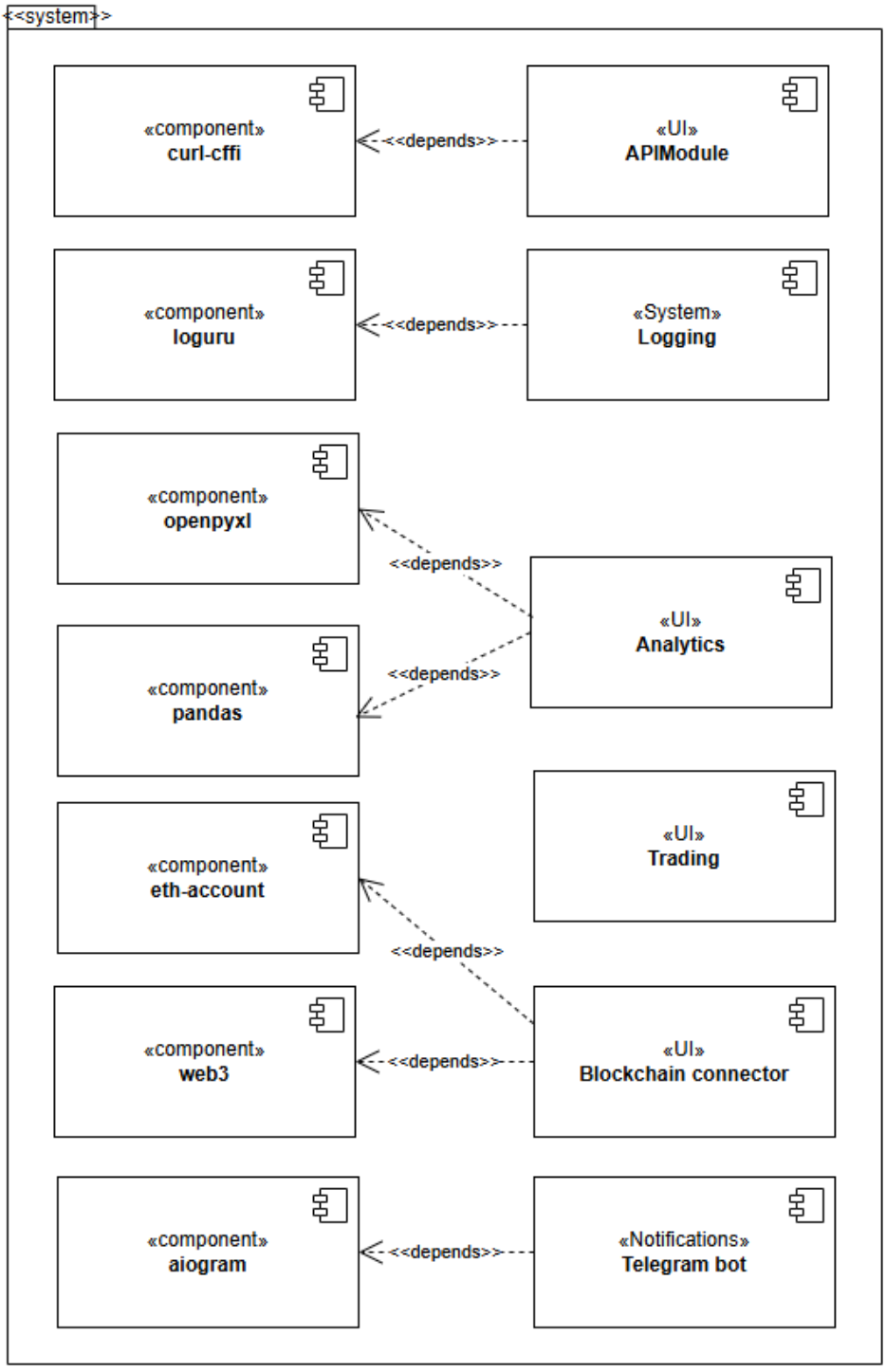


Рисунок 2.1 – Діаграма компонентів системи

Таким чином, розроблена модель системи дозволяє використовувати API для роботи автоматизованої системи NFT-торгівлі, збирати аналітику, отримувати сповіщення про угоди та власне проводити торгівлю NFT.

2.3 Розробка методу асинхронної черги з rate-limiting на рівні HTTP-методів

Метод асинхронної черги з rate-limiting на рівні HTTP-методів гарантує, що запити одного і того ж методу (GET, POST тощо) не перевищують заданого ліміту на одиницю часу, при цьому автоматично розподіляти навантаження між кількома проксі. Для кожного HTTP-методу створюється окрема пріоритетна черга запитів (PriorityQueue) і розраховується інтервал затримки (sleep_time) як обернена величина максимальної дозволеної швидкості запитів (з урахуванням кількості проксі й опціонального глобального ліміту). Перш ніж відправити запит, клієнт перевіряє, скільки часу минуло з моменту попереднього запиту цього ж методу, і, якщо треба, асинхронно «спить» необхідний час.

Ротація проксі відбувається прозоро: з ProxyQueue дістається найменш використовуваний сервер, і запит надсилається через нього. Якщо запит відпрацьовує успішно — черги оновлюються: запит вилучається з черги, час останнього запиту запам’ятовується. У разі помилок мережі (Timeout, ConnectionError) або відповіді з кодом 429/403 клієнт автоматично повторює — і при потребі переключається на інший проксі. А якщо приходить 401, спочатку виконується перезапит аутентифікації, і лише потім повторюється оригінальний запит. Це забезпечує стабільну роботу в умовах мінливих лімітів і ненадійних проксі без втручання розробника.

Вхідні дані:

- Для кожного HTTP-методу відома максимальна кількість запитів за секунду (з врахуванням кількості проксі та опціонального max_requests).

- Об’єкт запиту з полем method та пріоритетом priority (звичайно — мітка часу).

Робота цього методу складається з таких кроків:

1. Ініціалізація. На початку для кожного методу створюється власна порожня PriorityQueue і встановлюється sleep_time = 1 / фактичний_ліміт (секунд між запитами). Фактичний ліміт визначається за формулою:

фактичний_ліміт = min(rate_limit_for_method × кількість_проксі,

max_requests).

2. Додавання запиту в чергу. Перед виконанням запит вкладається в чергу відповідного методу у вигляді пари (priority, Request).

3. Очікування часу. Обчислення diff за формулою:

$$diff = час\_зараз - час\_останнього\_відправленого\_запиту.$$

Якщо diff < sleep_time, чекаємо ще sleep_time − diff секунд (або дробово спимо кілька разів по sleep_time/10, поки умова не виконається).

4. Вибір проксі. Якщо проксі завантажені, з черги ProxyQueue дістається найменш використовуваний проксі. Алгоритм вибору проксі та формування відповіді наведено на рисунку 2.2.

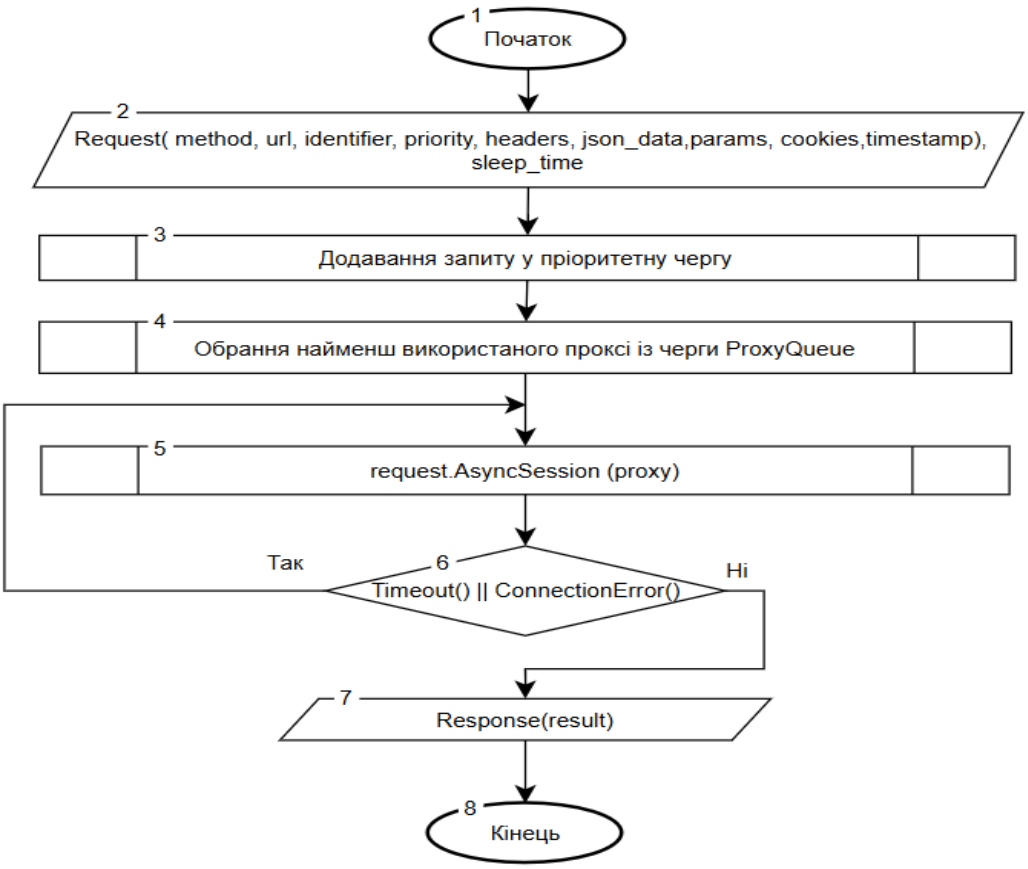


Рисунок 2.2 — Блок-схема алгоритму вибору проксі та формування відповіді

5. Виконання HTTP-запиту. Через асинхронну сесію надсилається запит із заданим методом, URL, параметрами та обраним проксі.

6. Оновлення стану черги. Після відправки та перед обробкою відповіді

видаляємо запит з черги та оновлюємо її.

7. Обробка помилок. Якщо сталося Timeout чи ConnectionError → рекурсивно повторюємо кроки 2–7 (той самий об’єкт Request). Якщо відповідь має статус 429 або 403 → логимо, робимо паузу й повторюємо те саме. Якщо 401 → виконуємо логін (login()), оновлюємо cookies і знову запускаємо повтор.

8. Повернення. Після успішного отримання відповіді (код ≠ 429/403/401, немає таймаутів) повертаємо об’єкт Response користувачу.

Інтегрована асинхронна черга з rate-limiting і прозорою ротацією проксі забезпечує надійну й передбачувану швидкість запитів, автоматично підлаштовуючись під зовнішні ліміти та стан проксі-мережі, без потреби додаткової логіки з боку розробника.

2.4 Розробка методу розумного добору пріоритетної плати за газ при прийнятті біду

Метод розумного добору пріоритетної плати за газ при прийнятті біду дозволяє обрати таку величину maxPriorityFeePerGas, при якій очікуваний чистий прибуток від продажу не опуститься нижче заданого мінімального рівня.

У контексті Ethereum (і сумісних мереж EIP-1559) «газ» — це одиниця обчислювальної роботи, яку виконує мережа для обробки транзакції [12]. При підписі і відправці транзакції (наприклад, щоб прийняти біду), потрібно вказати два взаємозалежні параметри: gas limit та gas price.

gas limit (межа газу) — максимально дозволена кількість газових одиниць, яку може «витратити» транзакція. Для простих трансферів ETH зазвичай беруть 21 000 одиниць, для викликів контрактів — більше (наприклад, ~250 000 для складних сценаріїв із затвердженнями). Якщо транзакція споживатиме більше газу, ніж gas limit, вона зупиниться із помилкою «out of gas», а витрачений газ все рівно сплатиться.

Ціна газу (gas price) за одиницю газу — це похідна від двох полів у EIP-1559: baseFeePerGas та maxPriorityFeePerGas.

baseFeePerGas — «базова» ціна, що автоматично змінюється з кожним блоком

залежно від завантаженості мережі. Цю частину плати мережа «спалює».

При прийнятті пропозиції на покупку NFT (біду) важливо знайти баланс між швидкістю обробки транзакції (чому сприяє вища пріоритетна плата за газ) та розміром чистого прибутку (що зменшується із зростанням плати). У цьому методі передбачено два режими: фіксований — коли користувач одразу задає точне значення maxPriorityFeePerGas, і динамічний — коли задається словник {поріг_прибутку: плата}.

У динамічному режимі спочатку обчислюється базовий прибуток як різниця між максимальною пропонованою ціною (sale_price) та мінімально прийнятною (min_price). Якщо прибуток позитивний, алгоритм ітерує записані пари «поріг прибутку → бажана плата за пріоритет». Він підбирає найбільшу плату, яка не знизить очікуваний чистий прибуток нижче зазначеного порога. Це досягається простою перевіркою: для кожної потенційної плати рахується «прибуток мінус вартість газу»: обсяг газу × (базова плата + пріоритетна плата), і якщо результат залишається вище порога – плата приймається. Якщо жодне з варіантів не задовольняє умову, операцію скасовують.

Після вибору оптимальної пріоритетної плати вона додається до базового рівня maxFeePerGas, і отримана конфігурація використовується для відправки транзакцій процесу схвалення та прийняття біду. Такий підхід допомагає автоматизувати тонке налаштування вартості газу, щоб транзакції швидко підтверджувалися в мережі, але при цьому не “з’їдали” увесь прибуток.

maxPriorityFeePerGas (також званий «tip») — додаткова плата, яка йде прямо валідаторам (майнерам) для прискорення обробки. Вона вказується самостійно, і чим вона вища, тим швидше майнери захочуть включити транзакцію в блок.

maxFeePerGas — максимально допустима сума, яку користувач готовий заплатити за газ в одній транзакції; фактично EIP-1559 встановлює параметр maxFeePerGas = baseFeePerGas + maxPriorityFeePerGas. Якщо базова плата зросте менш ніж на різницю до maxFeePerGas, залишок повертається відправнику.

Вхідні дані: sale_price – максимальна доступна сума (ETH) з котирування; min_price – мінімальна прийнятна сума; required_gas – очікуваний обсяг газу;

gas_price – базова пропозиція плати за газ (з поля maxFeePerGas); max_priority_fee – або одне число (фіксована плата), або словник {поріг_прибутку: допустима_пріоритетна_плата}.

Послідовність виконання:

1. Обчислення базового прибутку

$$base\_profit = sale\_price - min\_price$$

Якщо base_profit ≤ 0, значить операція невигідна і виконання методу зупиняється.

2. Динамічний випадок (словник). Перебираємо словник max_priority_fee у порядку спадання порогів прибутку. Якщо жоден fee_value не забезпечує необхідний поріг чистого прибутку — відмовитися від угоди (див. рисунок 2.3).

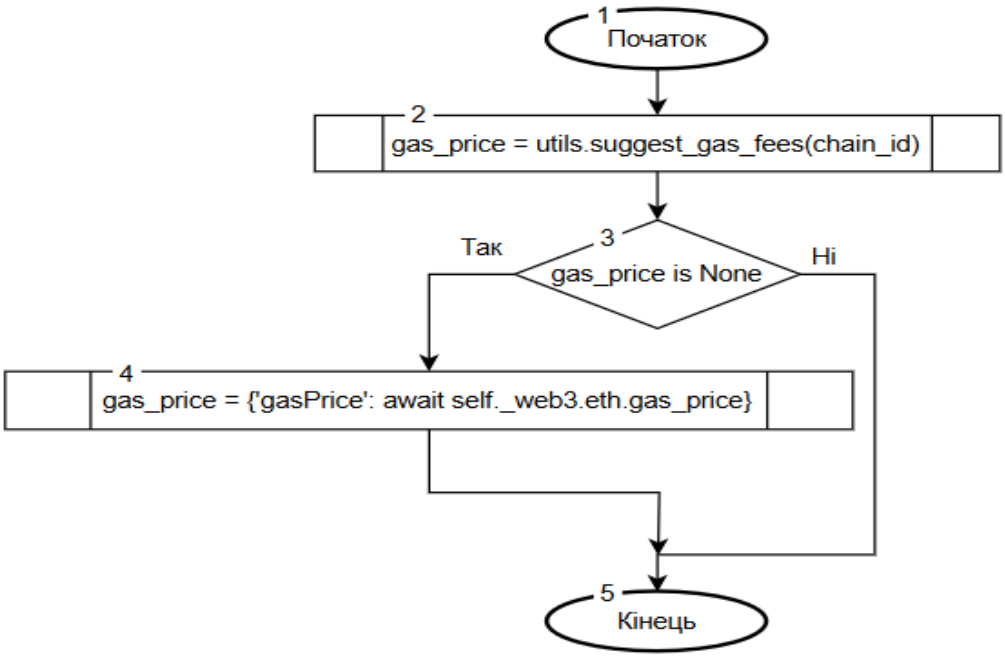


Рисунок 2.3 — Блок-схема алгоритму визначення поточної ціни газу

3. Накопичення повної пропозиції. Перетворюємо обраний maxPriorityFeePerGas в одиниці Wei та додаємо до maxFeePerGas.

4. Подальший ланцюжок. Використовуємо цей gas_price із новими значеннями в транзакції /v1/bids/accept. Далі підписуємо й надсилаємо approval- та accept-транзакції (за аналогією з алгоритмом лістингу).

Результат: maxPriorityFeePerGas підібрано так, щоб забезпечити мінімальний бажаний прибуток, або — відмова (False), якщо це неможливо.

“Розумний” підбір пріоритетної плати за газ дозволяє автоматично балансувати між швидкістю підтвердження транзакцій і розміром чистого прибутку, гарантуючи, що вартість мережевих зборів не “з’їсть” бізнес-метрику продажу.

2.5 Розробка алгоритмів роботи системи

Алгоритм завантаження проксі відповідає за зчитування списку проксі з файлу proxies.txt у каталозі BASE_DIR. Він підтримує два формати запису (IP:порт:логін:пароль і логін:пароль@IP:порт), перевіряє відповідність рядків одному з патернів, формує коректні HTTP-URL та відкидає некоректні записи з викиданням виключення.

Блок-схема алгоритму завантаження проксі наведена на рисунку 2.4.

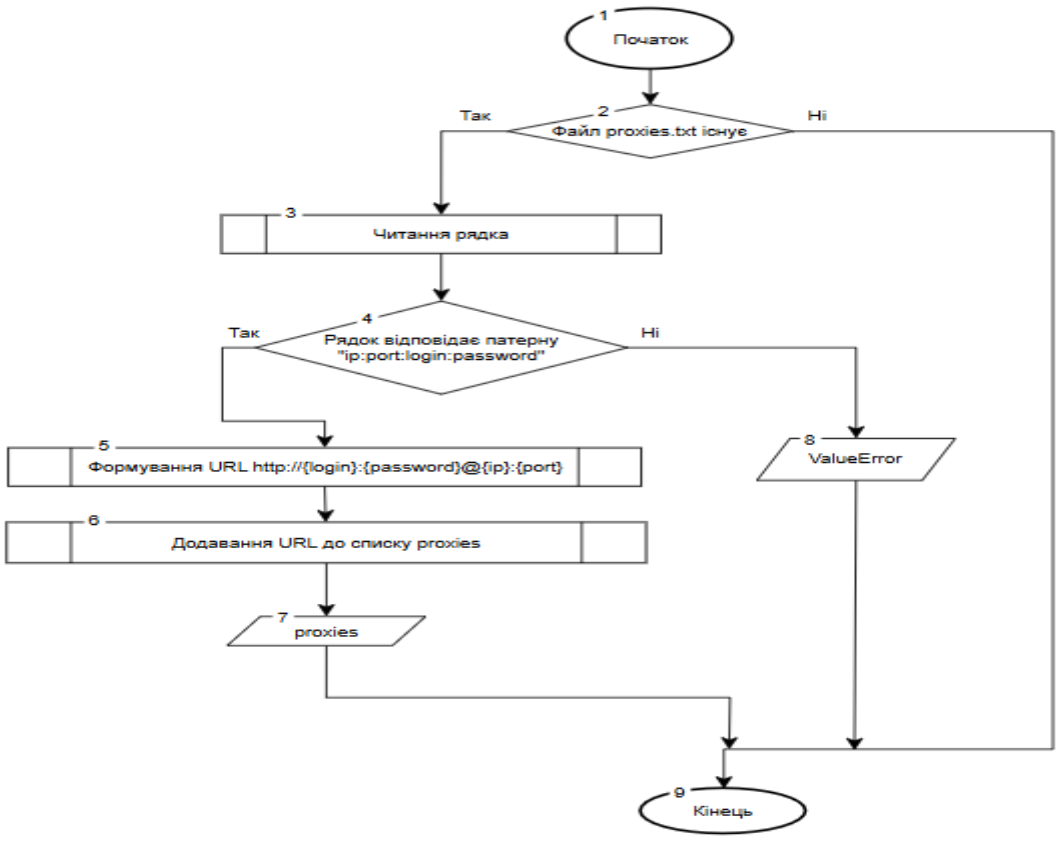


Рисунок 2.4 — Блок-схема алгоритму завантаження проксі

Після обробки кожного рядка алгоритм повертає масив готових рядкових URL-адрес проксі. Завдяки цьому наступні мережеві виклики можуть одразу брати список робочих серверів для ротації та балансування навантаження.

Для доступу до захищених ендпоінтів API застосунок зберігає JWT-токен у self._cookies['authToken']. Функція check_auth_token декодує без перевірки підпису, перевіряє, що в токені передана поточна адреса гаманця й час життя перевищує мінімальний поріг (зазвичай +7 днів).

Блок-схема алгоритму аутентифікації користувача наведена на рисунку 2.5.

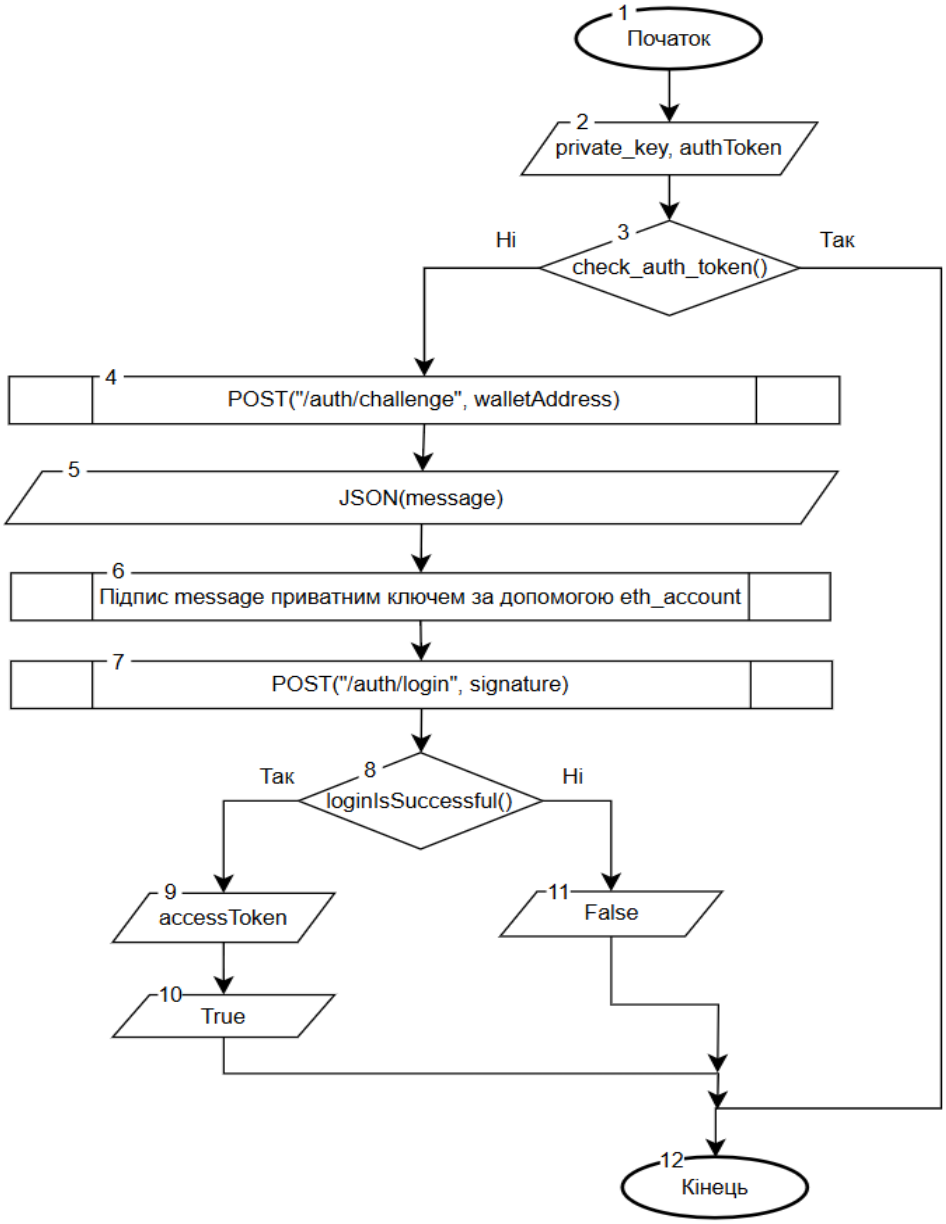


Рисунок 2.5 — Блок-схема алгоритму аутентифікації користувача

Якщо токен відсутній або прострочений, метод login виконує двоступеневу схему: спочатку POST /auth/challenge для отримання повідомлення, потім підписує його приватним ключем через EIP-191 і відправляє на /auth/login. Успішний виклик повертає accessToken, який зберігається й використовується в наступних запитах.

Таким чином, розроблені алгоритми забезпечують основний функціонал для автоматизованої торгівлі NFT предметами.

2.6 Висновки

У другому розділі було розроблено інформаційне забезпечення системи, було продемонстровано, які дані та як використовуються при автоматизованій торгівлі NFT предметами. Розроблено модель системи, наведено компоненти, що використовуються у системі. Розроблено алгоритми роботи системи, метод асинхронної черги з rate-limiting на рівні HTTP-методів та метод розумного добору пріоритетної плати за газ при прийнятті біду.

3 РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ ТОРГІВЛІ NFT

3.1 Варіантний аналіз та обґрунтування вибору мови програмування

Для вибору мови програмування для розробки автоматизованої системи NFT продажів на маркетплейсах розглянемо такі мови, як: Python, JavaScript, Rust.

Python [13] – це універсальна мова програмування з дуже зрозумілим і лаконічним синтаксисом, що пришвидшує розробку складних систем автоматизації продажів NFT. Окрім web3.py та py-opensea, вона має багатий набір інструментів для роботи з блокчейном і метаданими: eth-brownie або apeworx для розгортання і тестування смартконтрактів, ipfshttpclient для збереження артефактів у децентралізованому сховищі, а також fastapi, Flask чи Django для швидкого створення REST або GraphQL-інтерфейсів.

Архітектура Python наведена на рисунку 3.1.

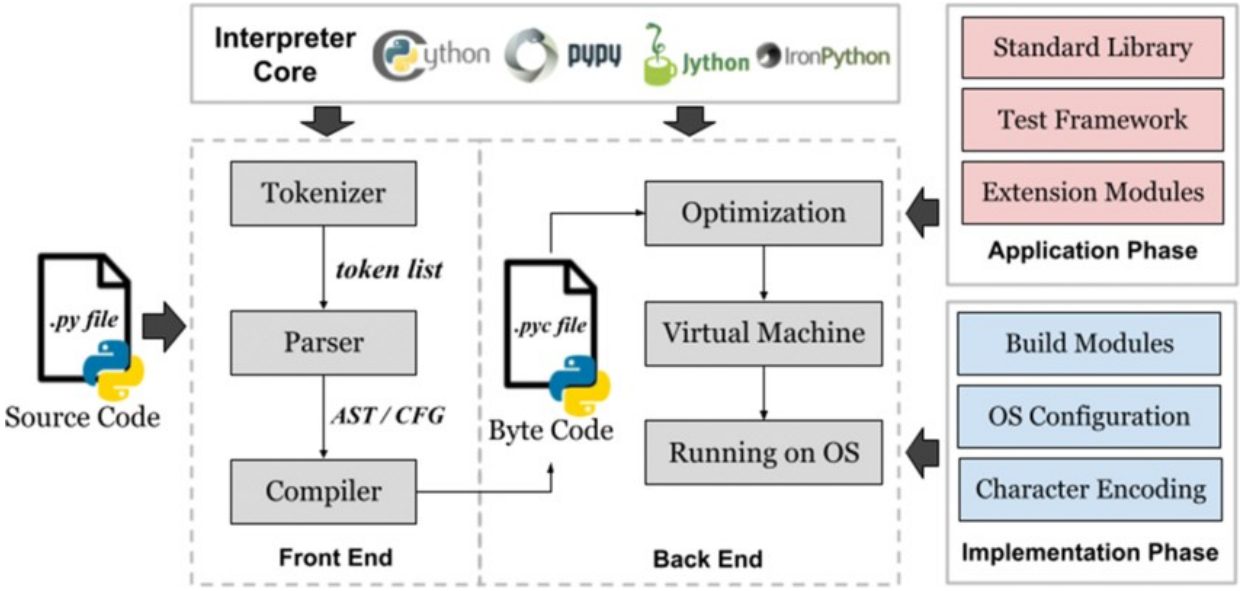


Рисунок 3.1 — Архітектура Python

Асинхронний стек на базі asyncio й websockets дозволяє обробляти події блокчейну в реальному часі без блокування головного потоку, а пакети pandas, numpy та бібліотеки машинного навчання (scikit-learn, TensorFlow, PyTorch) дають змогу аналізувати історію торгів, прогнозувати цінові тренди й автоматизувати стратегії купівлі-продажу. Крім того, велика спільнота та численні приклади коду на

GitHub значно полегшують навчання і прискорюють вирішення нетипових задач.

JavaScript [14] – це технологія для створення динамічних веб-інтерфейсів і взаємодії з гаманцями користувачів прямо в браузері. Бібліотеки Web3.js та ethers.js забезпечують повний доступ до JSON-RPC Ethereum-сумісних мереж, дозволяючи підписувати транзакції, відстежувати події смартконтрактів і працювати з метаданими NFT за допомогою OpenSea SDK або Moralis SDK. На стороні сервера Node.js ви можете використовувати фреймворки Next.js чи NestJS для розробки full-stack рішень, налаштовувати cron-таски для періодичного сканування торгів або webhook-сервери для обробки натискань користувачів. Інструменти тестування (Hardhat, Truffle, Mocha/Chai, Jest) та система типізації TypeScript підвищують надійність коду, а широке ком'юніті npm-пакетів і плагінів дає змогу швидко інтегрувати засоби авторизації (MetaMask, WalletConnect), моніторингу транзакцій та безпеки.

Архітектура JavaScript наведена на рисунку 3.2.

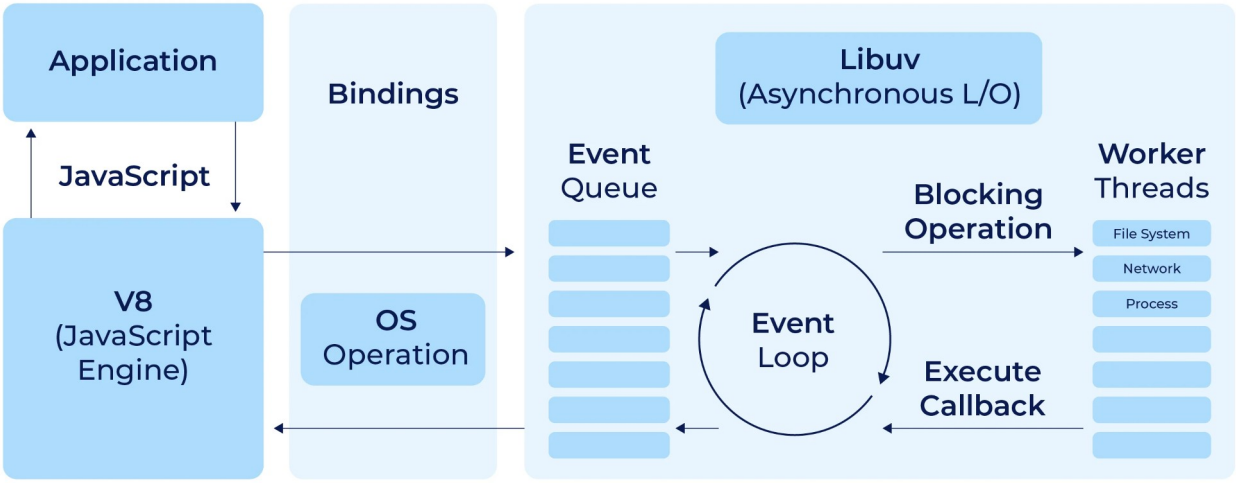


Рисунок 3.2 — Архітектура JavaScript

Rust [15] – це мова системного рівня, орієнтована на високу продуктивність, паралелізм і безпечне керування пам'яттю без runtime-компонента. Завдяки крейтам ethers-rs, web3 та substrate, а також підтримці serde JSON й reqwest для HTTP-запитів, Rust підходить для створення високопродуктивних бекендсервісів, індексерів NFT або компонентів, що компілюються у WebAssembly для

клієнтського виконання. Розвинена система Cargo і засоби статичного аналізу (Clippy, rustfmt) допомагають дотримуватися єдиного стилю та уникати помилок на етапі компіляції. Хоча крива навчання Rust є дещо вищою, його переваги – відсутність race conditions, нульова вартість абстракцій та потужна підтримка асинхронності через tokio – роблять цю мову відмінним вибором для критичних до швидкодії сервісів у екосистемі NFT.

Архітектура Rust наведена на рисунку 3.2.

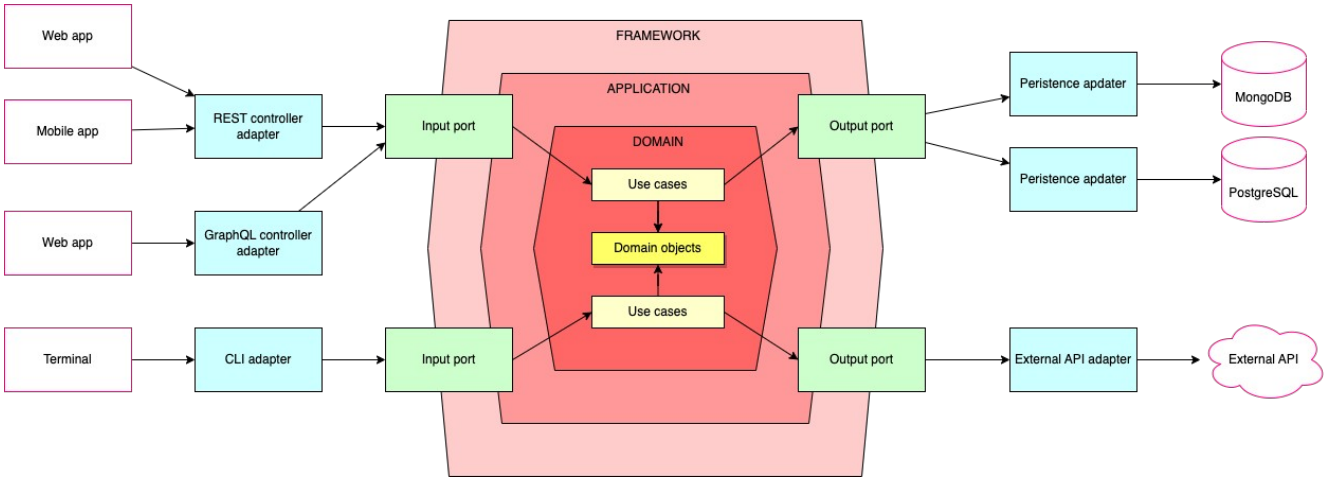


Рисунок 3.2 — Архітектура Rust

Порівняємо характеристики розглянутих мов програмування та сформуємо таблицю 3.1.

Таблиця 3.1 — Порівняльний аналіз мов програмування

Характеристика	Python	JavaScript	Rust
Екосистема NFT/блокчейн бібліотек	+	+	+
Інтеграція з маркетплейсами	+	+	+
Асинхронна обробка	+	+	+
Обробка даних і аналітика	+	-	-
Можливості машинного навчання для аналізу ринку	+	-	-
Кросплатформеність	+	+	+
Готові шаблони NFT рішень	+	-	-
Сумарний бал	7	4	4

З огляду на результати проведеного аналізу, було прийнято рішення

розробляти систему автоматизованої торгівлі NFT з використанням мови Python.

3.2 Варіантний аналіз та обґрунтування вибору середовища розробки

Розглянемо такі середовища розробки для Python: Pycharm, Visual Studio Code , Neovim.

PyCharm Professional [16] — IDE від JetBrains, спеціально оптимізована під розробку на Python. Окрім вбудованої підтримки віртуальних оточень (venv, Conda) та інтегрованого відлагоджувача з точками зупину й переглядом стеку викликів, вона пропонує розширені можливості аналізу коду: інспекції за PEP8, автоматичне виправлення помилок, комплексне рефакторинг (перейменування, екстракція методів, інлайн-зміни) і граф залежностей між модулями.

Інтерфейс PyCharm наведено на рисунку 3.4.

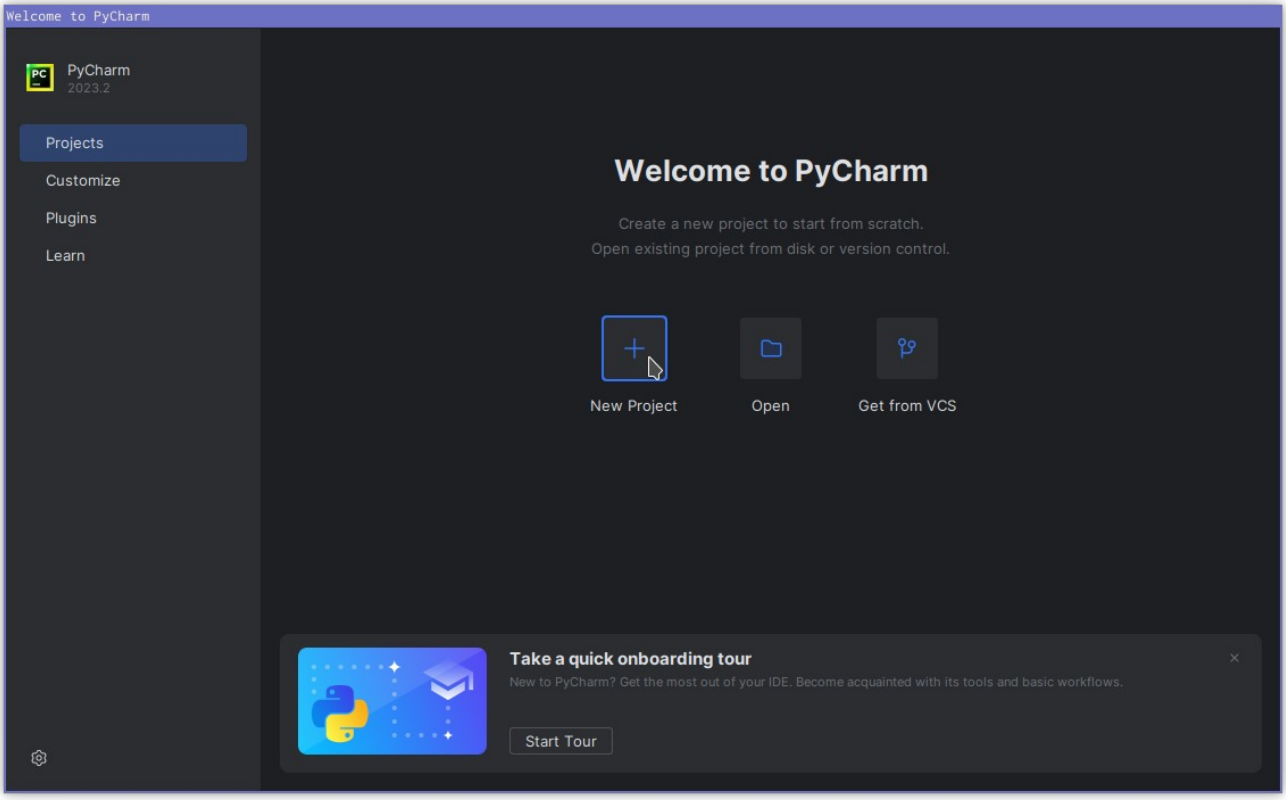


Рисунок 3.4 — Інтерфейс PyCharm

Для роботи з базами даних є окремий Database Tool Window, який підтримує підключення до PostgreSQL, MySQL, SQLite тощо, а також можливість перегляду

схем і запуску SQL-запитів. У “Scientific Mode” передбачена інтеграція з Jupyter Notebook та підтримка matplotlib, а для віддаленої розробки — SSH-інтерпретатори, Docker Compose і WSL. Спеціальні плагіни допомагають працювати з Django, Flask, FastAPI та іншими фреймворками, а профайлер і тест-ранер з покриттям коду значно спрощують оптимізацію та валідацію складних асинхронних сервісів.

VS Code [17] — легковага й надзвичайно розширювана платформа з величезним маркетплейсом розширень. Для Python є офіційне розширення Python і Pylance, які забезпечують швидке автодоповнення, інтелектуальну типізацію та миттєвий аналіз помилок. Вбудований термінал і прості налаштування для venv/Conda дозволяють миттєво переключатися між проєктами.

Інтерфейс VS Code наведено на рисунку 3.5.

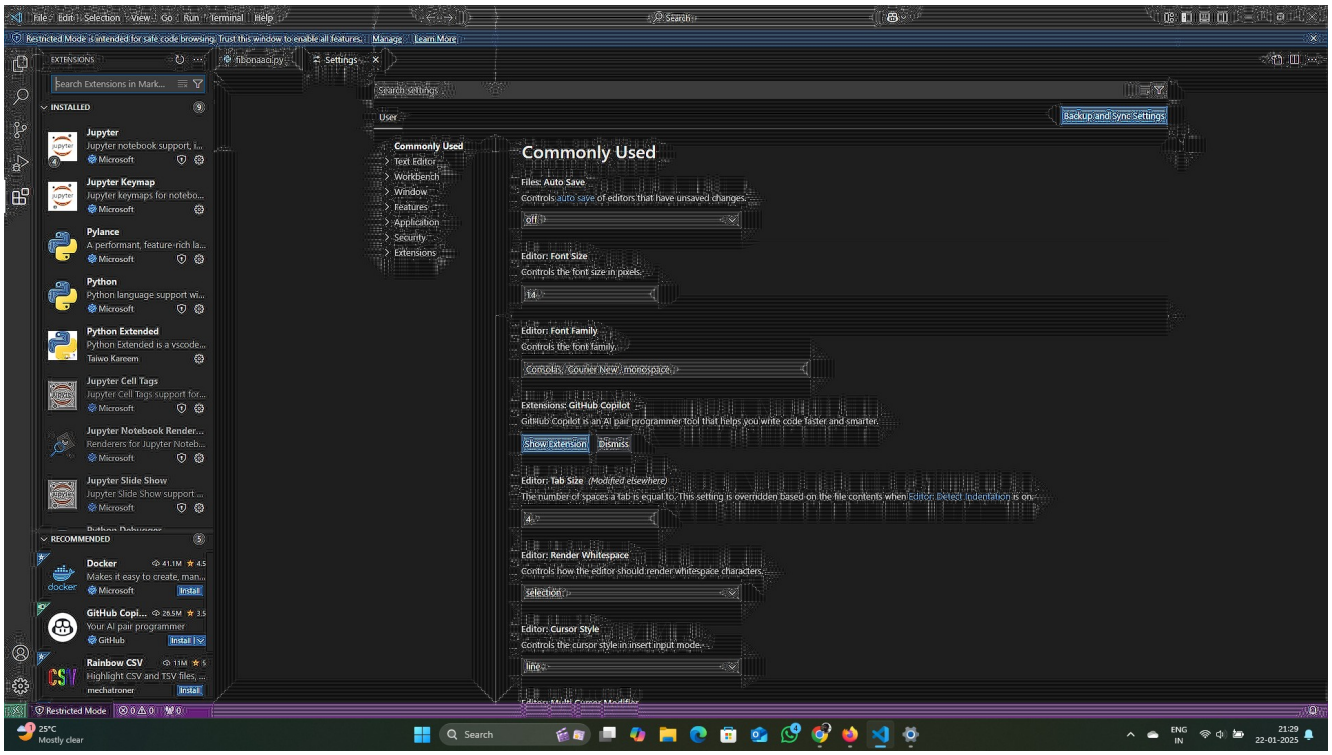


Рисунок 3.5 — Інтерфейс VS Code

Розширення Docker і Remote –Containers дають змогу працювати в контейнерах або віддалено через SSH безпосередньо з VS Code. Інструменти для роботи з Git (GitLens) і Live Share спрощують колаборацію в реальному часі, а

REST Client дозволяє відправляти HTTP-запити без виходу з редактора. Конфігурації для debugpy забезпечують гнучке відлагодження, а система задач (Tasks) і налаштовувані сервіси автоматизують запуск скриптів, тестів і збірку.

Neovim [18] — мінімалістичне середовище, яке за допомогою LSP-сервера pyright перетворюється на легку IDE з підтримкою статичної типізації, автодоповнення та навігації по проєкту. Плагіни ale або neomake інтегрують lint-інструменти (flake8, mypy), а nvim-dap або vimspector додають можливості відлагодження з точками зупину. За допомогою менеджерів плагінів (packer.nvim, vim-plug) можна швидко оновлювати та налаштовувати доповнення, а Treesitter забезпечує точний синтаксичний розбір і підсвічування більшості мов. Інтеграція з Telescope дає потужний fuzzy-файндер для файлів, функцій і символів, а lualine та which-key покращують інтерфейс та підказки. Neovim ідеальний для тих, хто цінує швидкість запуску, гнучкість налаштувань у конфігураційному файлі init.lua та абсолютний контроль над робочим простором.

Інтерфейс Neovim наведено на рисунку 3.6.

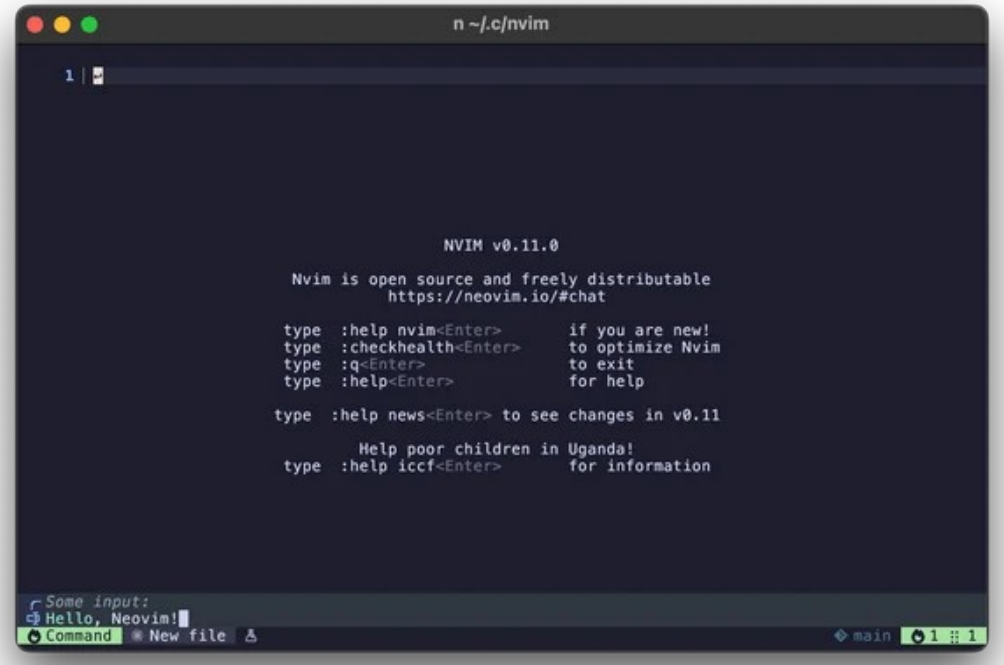


Рисунок 3.6 — Інтерфейс Neovim

Для порівняння можливостей розглянутих середовищ розробки, сформуємо

таблицю 3.2.

Таблиця 3.2 — Порівняння середовищ розробки

Характеристика	PyCharm Professional	Visual Studio Code	Neovim +coc.nvim
Вбудований відлагоджувач	+	+	+
Підтримка віртуальних оточень	+	+	+
Інтеграція з Docker/Docker Compose	+	+	–
Профайлер	+	–	–
Плагіни для linting та форматування	+	+	+
Підтримка контейнеризації (Remote − Containers)	+	+	–
Сумарний бал	6	5	3

З огляду на результати аналізу середовищ розробки, було прийнято рішення розробляти автоматизовану систему торгівлі NFT предметами з використанням Pycharm.

3.3 Розробка модуля бізнес-логіки та взаємодії з Blur API

Модуль бізнес-логіки та взаємодії з Blur API побудований навколо класу Blur , який в конструкторі викликає load_proxies() для підготовки списку HTTP-проксі та налаштування інтервалів між запитами. Функція load_proxies() читає файл proxies.t xt, перевіряє кожен рядок на відповідність двом форматам і формує список URL. Алгоритм роботи функції включає послідовну обробку кожного рядка файлу, перевірку на наявність символів '@' та ':', що визначає формат проксі (з аутентифікацією чи без), та формування відповідного URL. При виявленні помилок у форматі рядок пропускається з відповідним повідомленням у лог (див. рисунок 3.1).

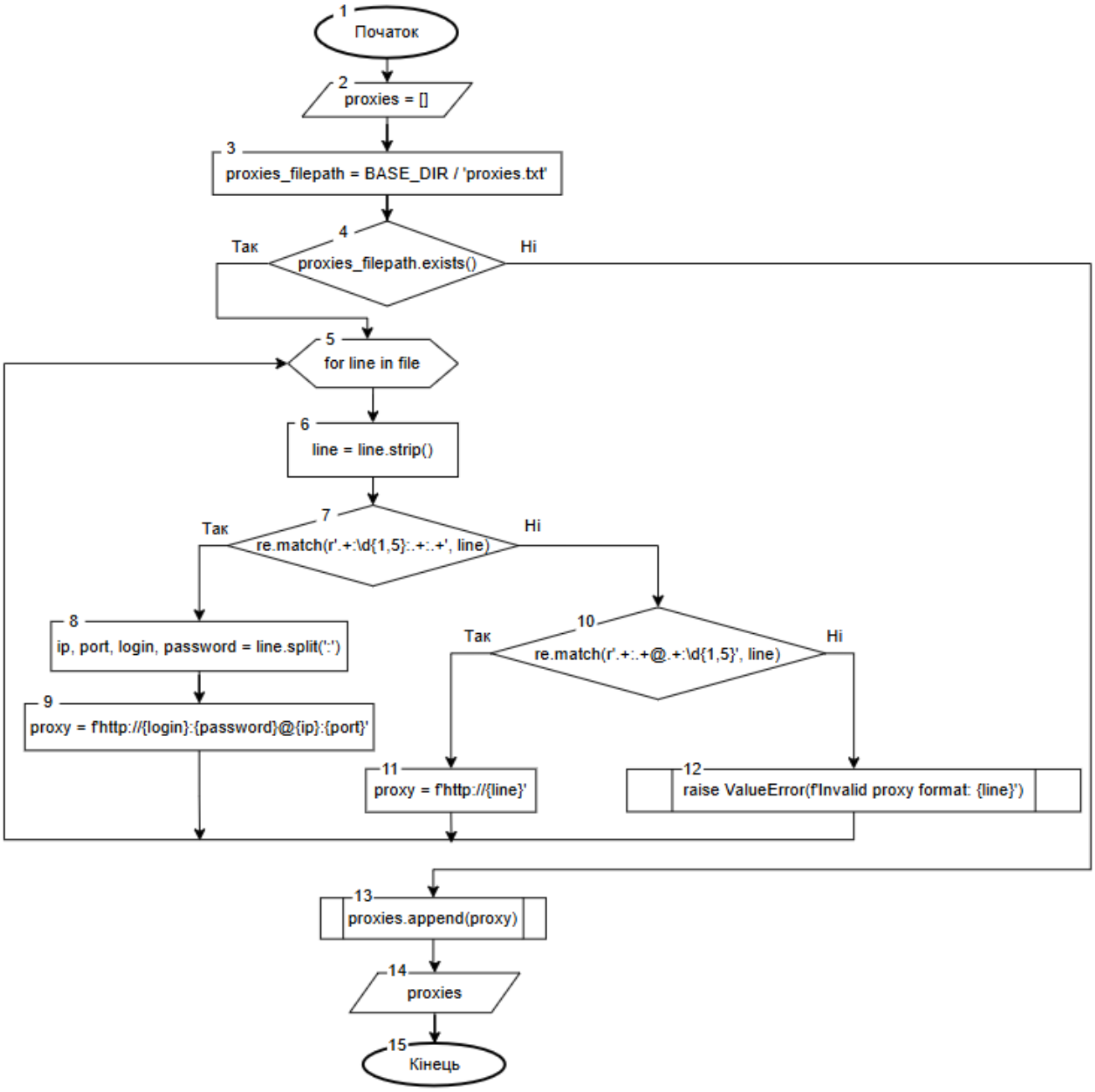


Рисунок 3.1 — Алгоритм функції load_proxies()

Ключовим у мережевій роботі є метод _send_request(), який впроваджує асинхронний rate-limit і проксі-ротацію. Спочатку запит ставиться в чергу PriorityQueue, потім чекає необхідну паузу, далі обирається найменш навантажений проксі і відправляється через AsyncSession. Помилки Timeout, ConnectionError, а також відповіді 429/403/401 обробляються автоматичним повтором (для 401 --- із викликом login()). Блок-схема на рисунку 3.2 демонструє логіку обробки запитів: початкову постановку в чергу з пріоритетом,

очікування згідно з rate-limit, вибір оптимального проксі на основі статистики навантаження, відправку запиту та комплексну обробку різних типів помилок з відповідними стратегіями повтору.

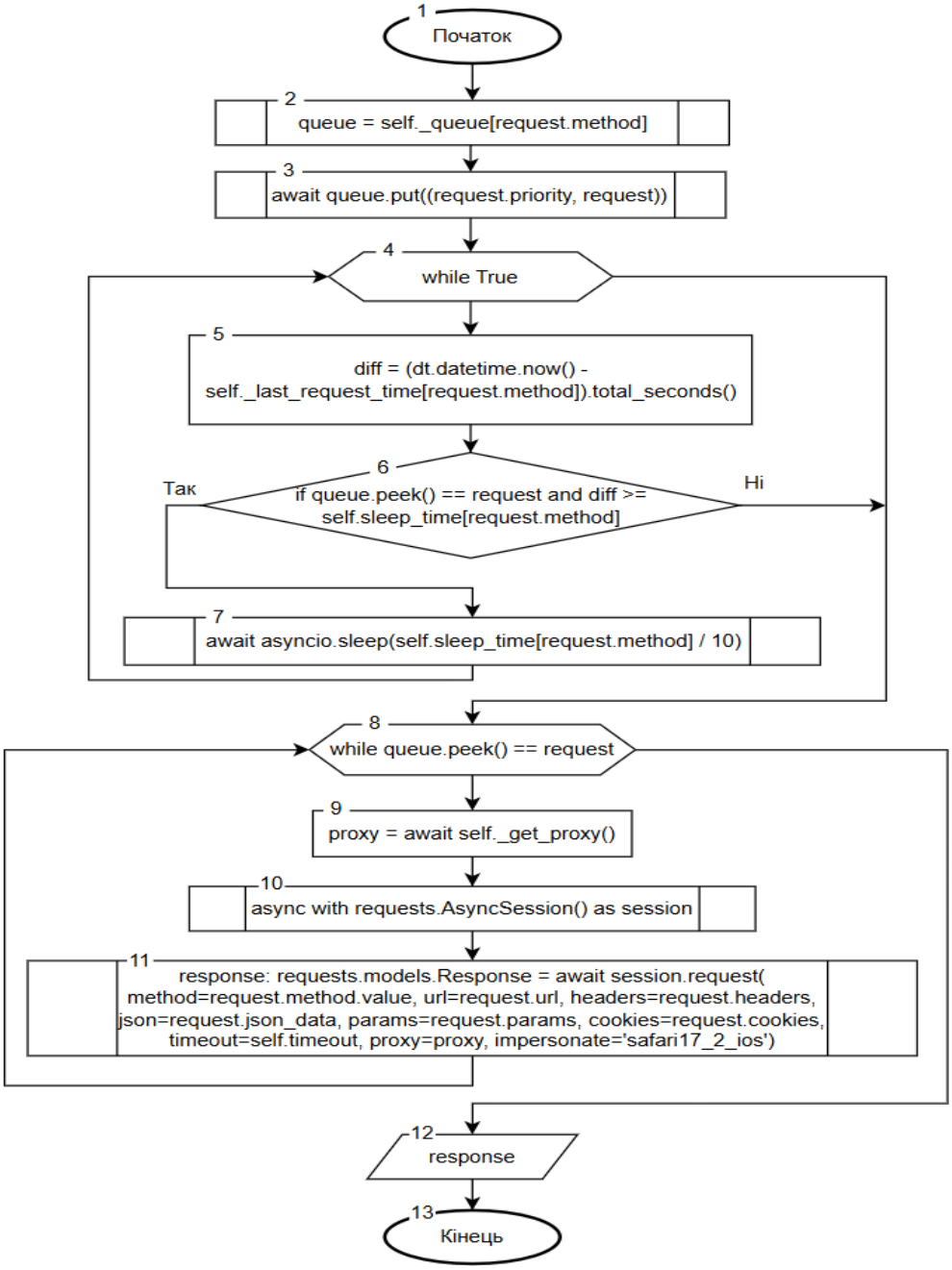


Рисунок 3.2 — Алгоритм роботи функції _send_request()

Оскільки більшість викликів до Blur API вимагають JWT-аутентифікації, клас реалізує check_auth_token() та login(). Перший перевіряє термін дії поточного токена, інший отримує new challenge, підписує його через

encode_defunct і надсилає на /auth/login. На блок-схемі на рисунку 3.3 відображено двоетапний процес аутентифікації: спершу перевірка валідності існуючого токена з урахуванням часу його життя, а при необхідності оновлення - генерація криптографічного виклику від сервера, його підпис приватним ключем користувача за стандартом EIP-191 та отримання нового JWT-токена.

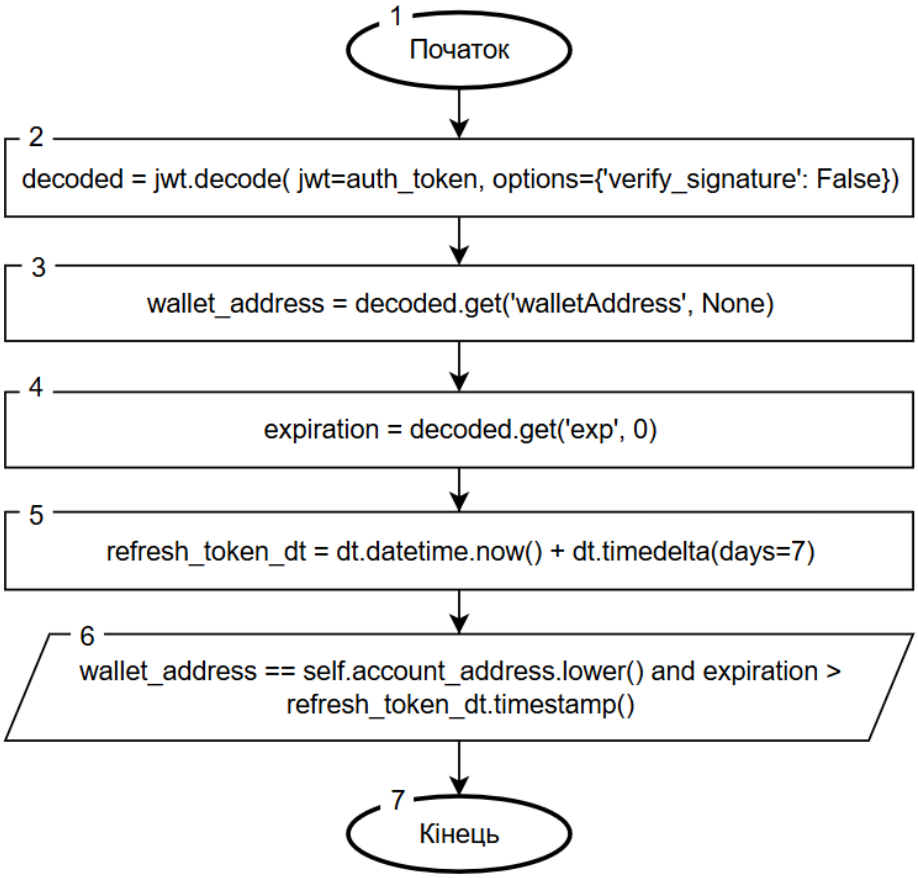


Рисунок 3.3 — Алгоритм роботи функцій check_auth_token() та login()

Для скорочення числа звернень до API дані про колекцію кешуються у get_collection_address() та get_collection_fee(). Перший підтягує адресу контракту, другий --- мінімальний роялті (у bips → ETH). Обидва результати зберігаються в пам'яті self.collection_*, щоб уникнути дублювання. Блок-схема на рисунку 3.4 ілюструє механізм кешування з перевіркою наявності даних у пам'яті перед зверненням до API: якщо дані вже завантажені, вони повертаються з кешу, в іншому випадку виконується API-запит з подальшим збереженням результату для майбутнього використання.

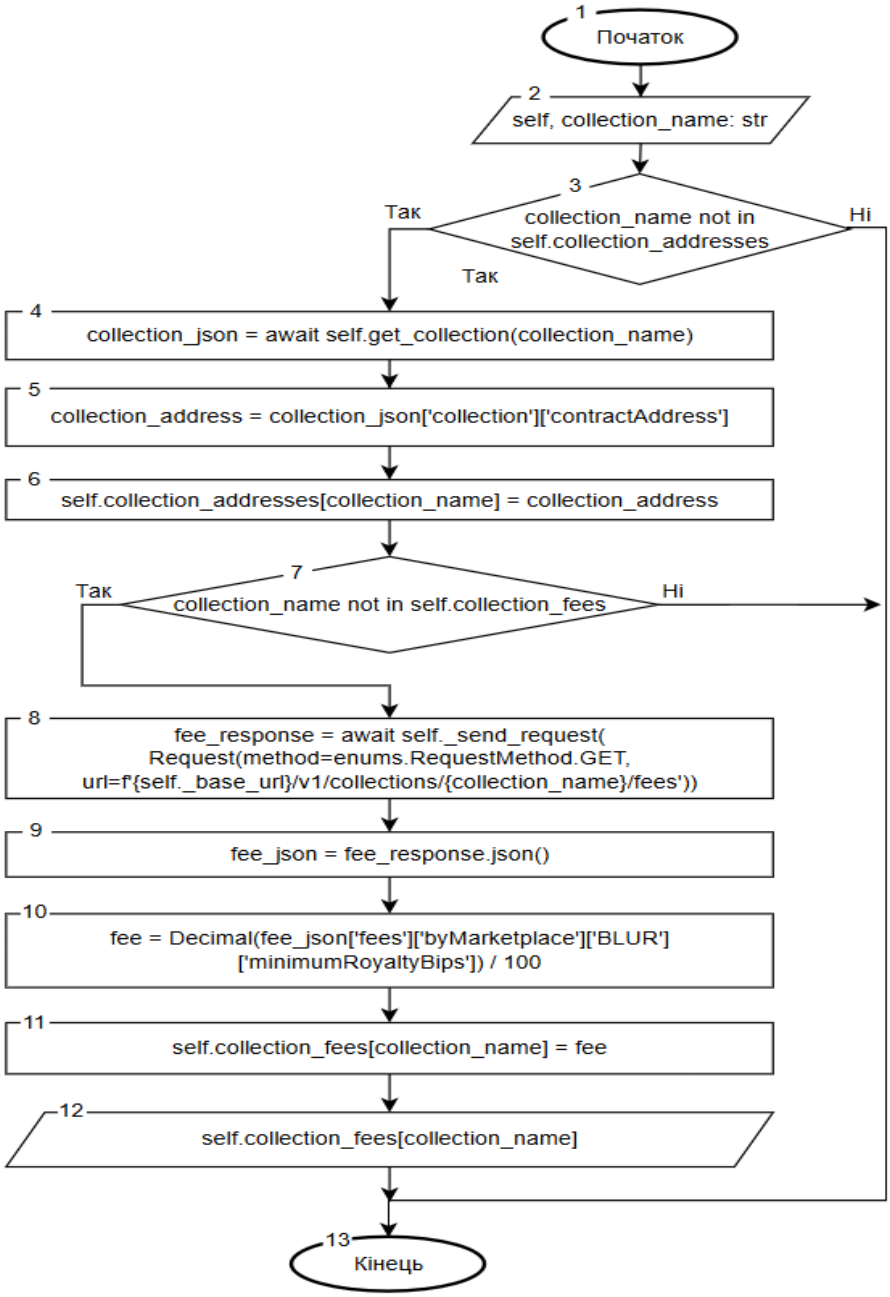


Рисунок 3.4 — Алгоритм роботи функцій get_collection_address() та get_collection_fee()

Збір ринкових даних реалізовано у _get_bids() та get_listings(). Перший будує всі комбінації traits, надсилає фільтровані виклики /executable-bids, агрегує та сортує об'єкти Bid. Другий обробляє пагінацію /tokens?filters= і будує масив Listing. Блок-схема на рисунку 3.5 демонструє процес збору ринкових даних: функція _get_bids() генерує всі можливі комбінації характеристик NFT, виконує паралельні запити до API з різними

фільтрами, агрегує отримані пропозиції та сортує їх за ціною. Функція get_listings() реалізує пагіновану обробку даних про лістинги з формуванням структурованого масиву об'єктів.

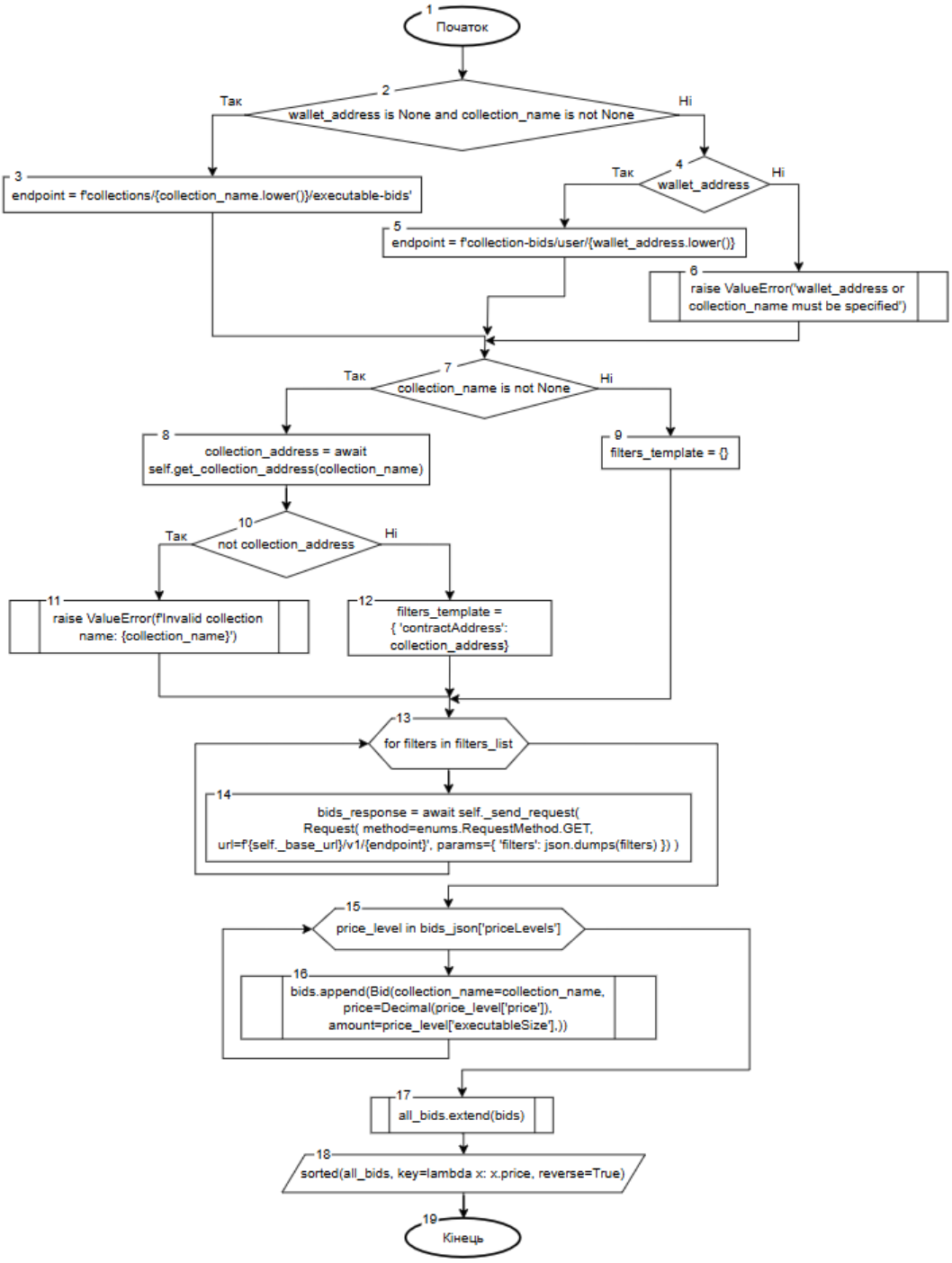


Рисунок 3.5 — Код функцій _get_bids() та get_listings()

Основні торговельні операції інкапсулюються у відповідний функціонал submit_listing() та accept_highest_bid(). Перший формує JSON для

/orders/format, підписує всі approvals транзакції через encode_typed_data та відправляє /orders/submit. Функція submit_listing() показує реалізує цикл створення лістингу: формування параметрів ордеру (ціна, термін дії, умови), отримання структури для підпису від API, криптографічне підписання транзакцій згідно зі стандартом EIP-712, включаючи approval для передачі NFT, та відправку готового ордеру на маркетплейс.(див. рисунок 3.6).

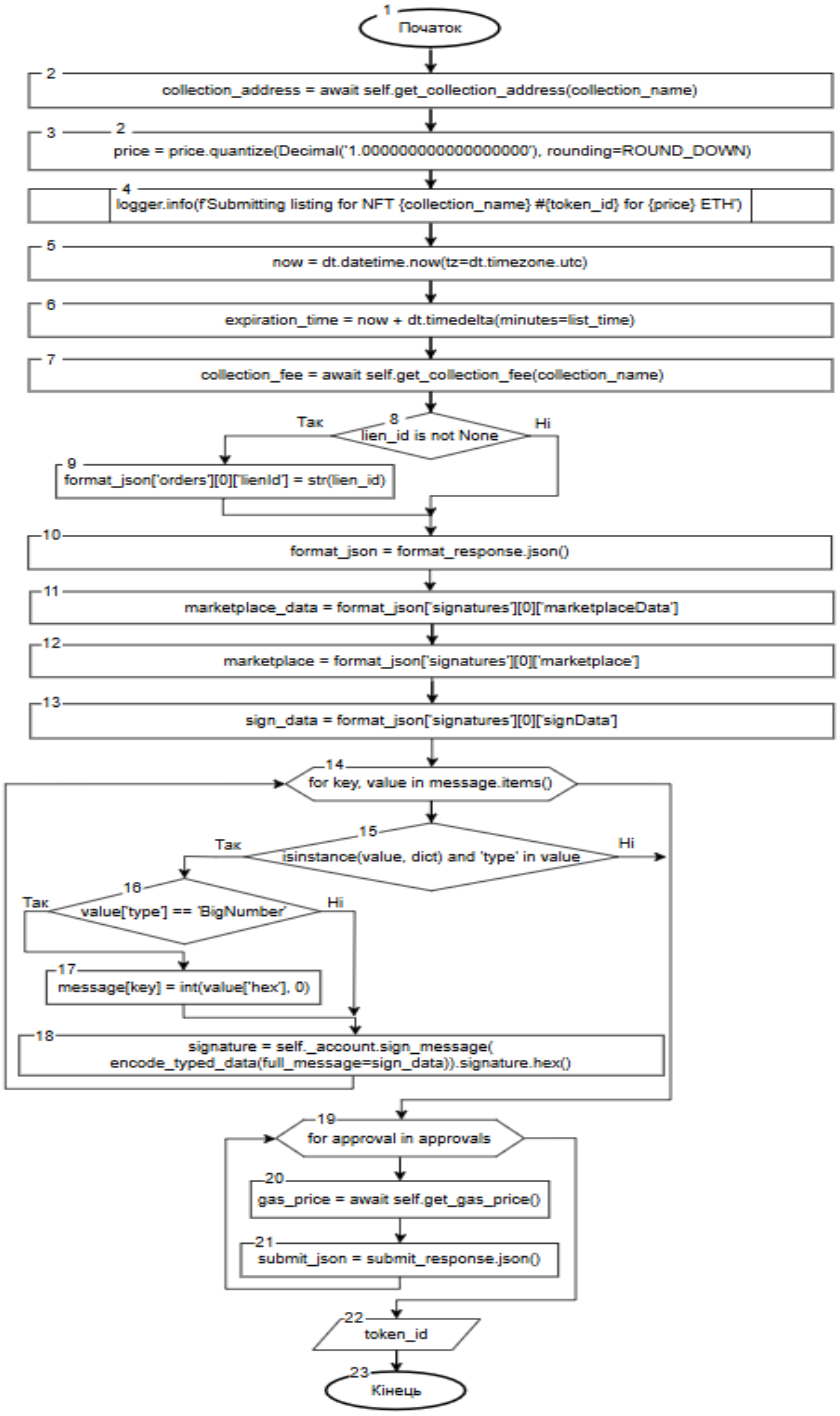


Рисунок 3.6 — Код функції submit_listing()

Другий алгоритм через /v1/bids/quote отримує ціни, рахує base_profit, динамічно підбирає maxPriorityFeePerGas під ваші пороги, а потім виконує /v1/bids/accept і підписує accept-транзакції (див. рисунок 3.7).

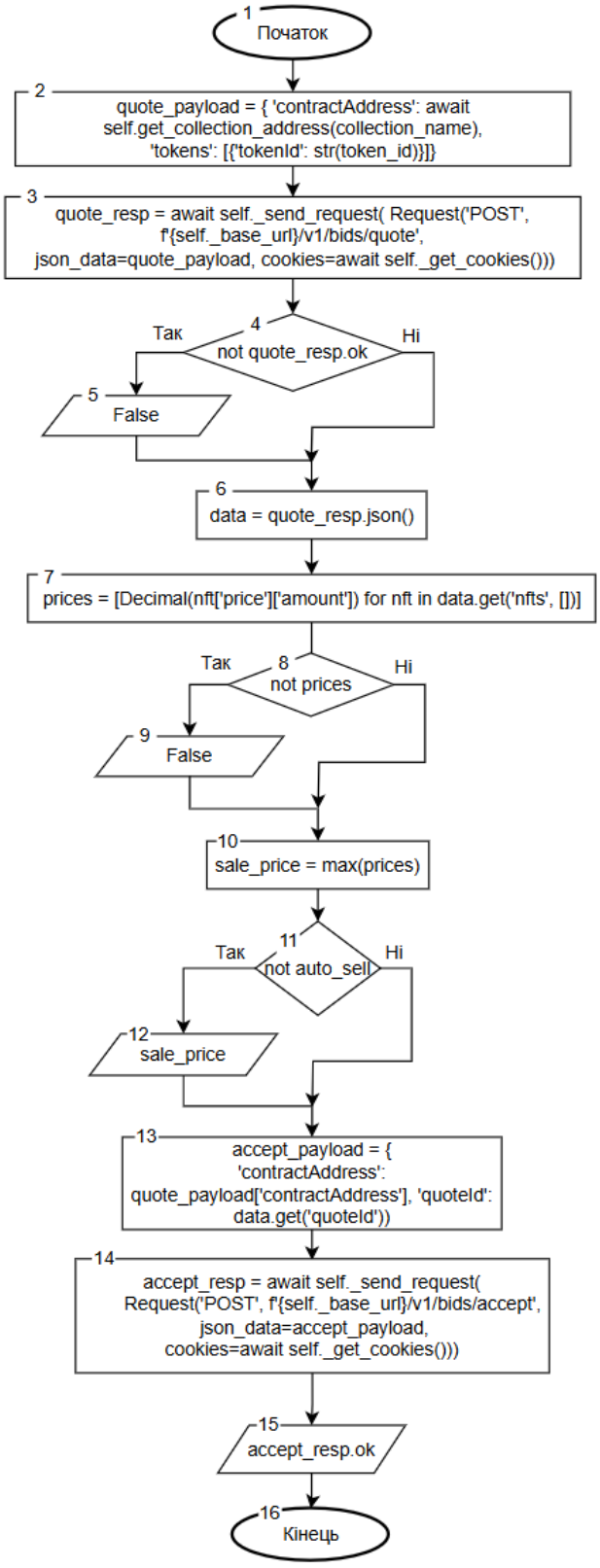


Рисунок 3.7 — Код функції accept_highest_bid()

Актуальні ціни за газ витягуються у get_gas_price(), яка намагається спершу викликати utils.suggest_gas_fees, а за відсутності результату звертається до web3.eth.gas_price. Функція реалізує багаторівневу систему отримання цін на газ: спочатку спроба використання вбудованого алгоритму оцінки комісій, який враховує поточне навантаження мережі та рекомендує оптимальні значення base fee та priority fee, а при його недоступності - використання стандартного методу web3 для отримання базової ціни газу.

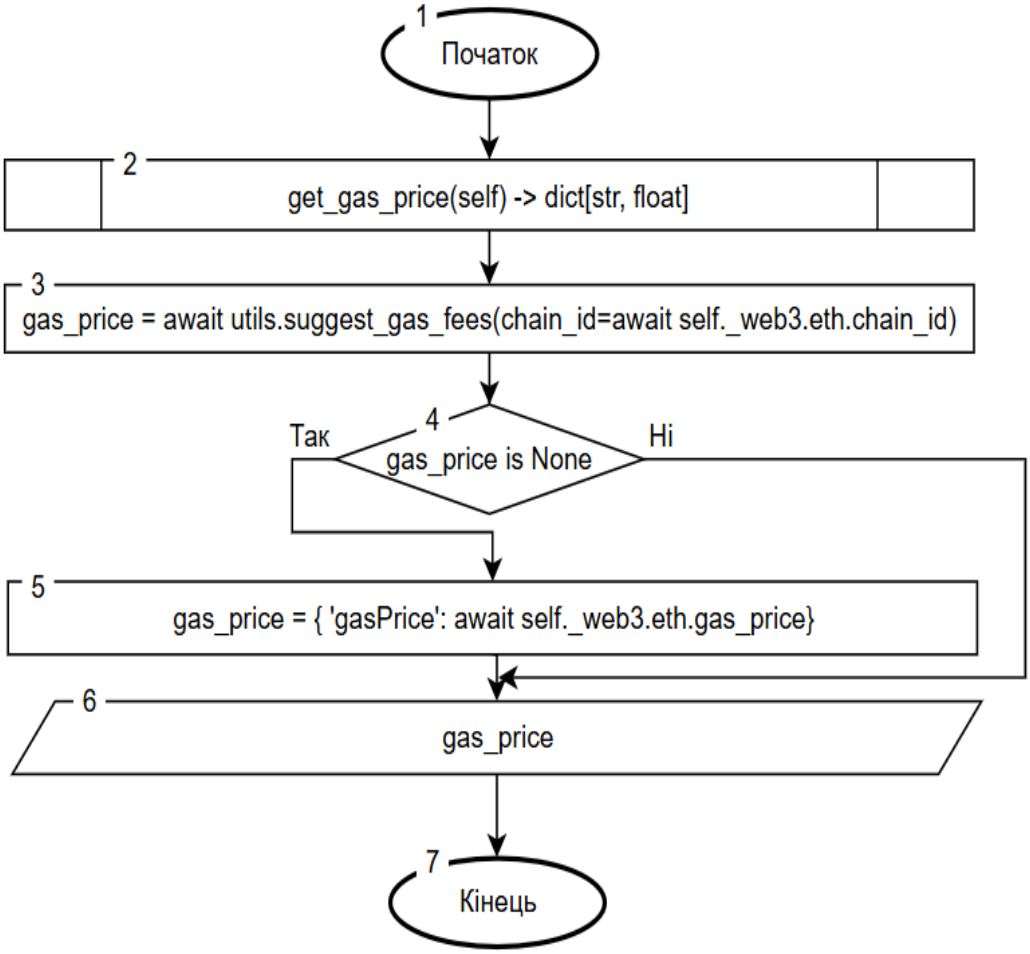


Рисунок 3.8 — Код функції get_gas_price()

Успішне завершення торгівельного циклу для кожного NFT забезпечує функція run_nft(), яка у нескінченній петлі перевіряє nft.paused, довантажує поточні позики, токен і лістинги через get_loans(), get_token(), get_listings(), рахує підрізку і прибуток, а потім --- залежно від nft.auto_sell --- викликає accept_highest_bid() або

submit_listing().

Блок-схема алгоритму функції run_nft() наведено на рисунку 3.9.

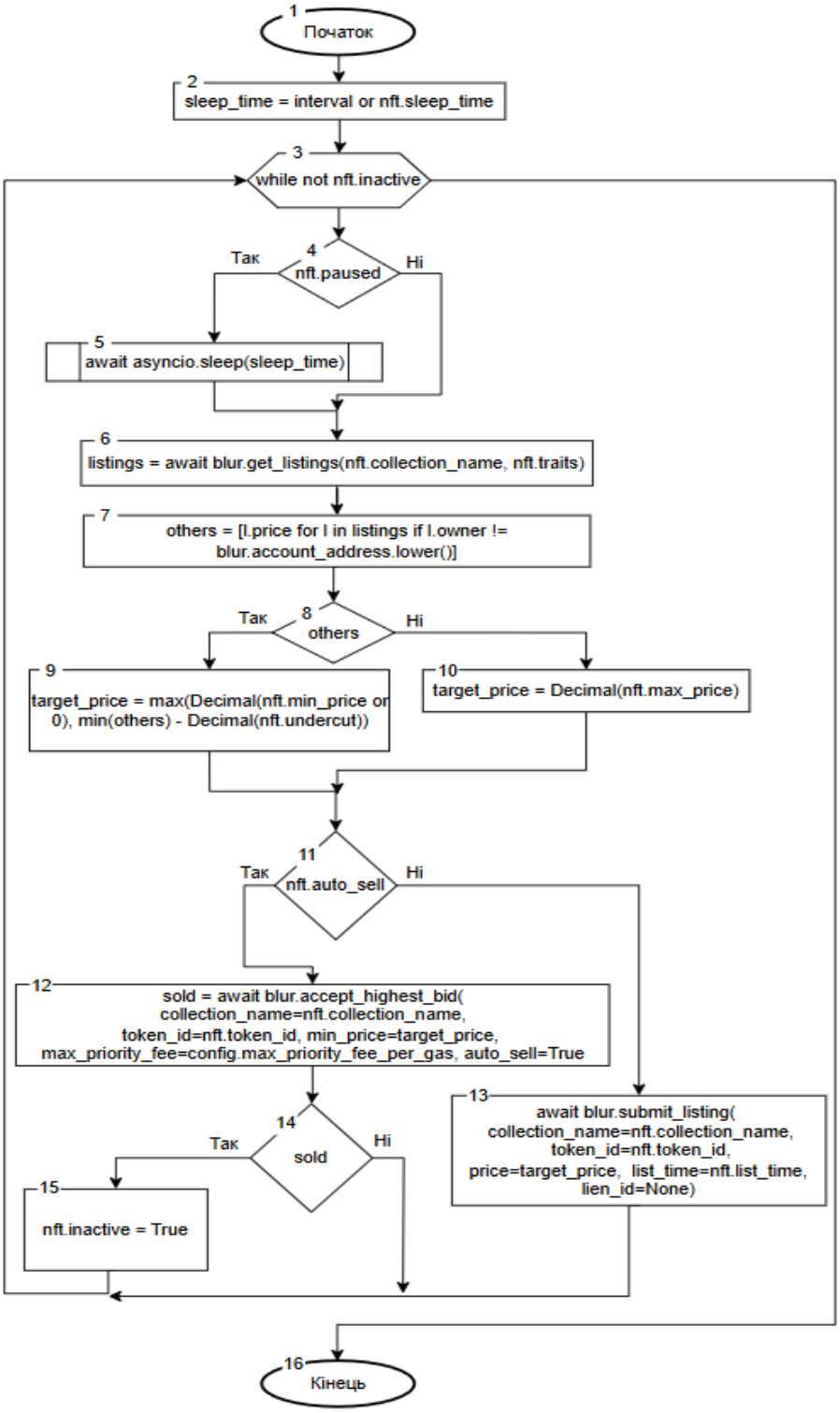


Рисунок 3.9 — Код функції run_nft()

Пауза між ітераціями формується на основі nft.sleep_time та результатів

попереднього циклу. Функція реалізує основний цикл автоматизованої торгівлі: перевірку стану паузи, завантаження актуальних даних про позики та ринкову ситуацію, розрахунок потенційного прибутку з урахуванням підрізки, прийняття рішення про тип операції (продаж за найвищою пропозицією або створення власного лістингу) на основі налаштувань користувача, та динамічне налаштування інтервалів між ітераціями.

Отже, розроблений модуль забезпечує функціонал для торгівлі NFT предметами, виконання мережевих запитів, налаштування параметрів торгівлі.

3.4 Розробка модуля графічного інтерфейсу та управління налаштуваннями NFT

Модуль графічного інтерфейсу та управління налаштуваннями NFT відповідає за всю взаємодію з користувачем і збереження початкових налаштувань для кожного токена. Він дозволяє вводити базові дані: посилання на NFT, цінові межі, параметри підрізання й прибутку, режими Auto sell і Flagged mode, та редагувати вже існуючі записи. У його завдання входить валідація вводу, перетворення користувацьких налаштувань у внутрішні об'єкти а також управління локальним файлом конфігурації --- зчитуванням і записом списку NFT для подальшої обробки. Модуль складається з трьох основних класів: AddWindow, NFTSettingsWindow та SelectionWindow.

Клас AddWindow відповідає за діалог додавання нового NFT до списку. При ініціалізації він приймає поточний список nfts та створює поля вводу для всіх ключових параметрів. Метод save_nft() збирає значення полів, перевіряє їх, викликає конструктор NFT.from_raw(...) і додає новий об'єкт до self.nfts. Фрагмент коду демонструє реалізацію ключових методів класу: save_nft() виконує збір та валідацію даних з форми, включаючи перевірку числових значень та URL, створення об'єкта NFT з введених параметрів та його додавання до списку; методи switch_flagged() та switch_auto_sell() обробляють перемикання відповідних булевих налаштувань через зміну стану чекбоксів інтерфейсу. Фрагмент класу AddWindow наведено на рисунку 3.10.

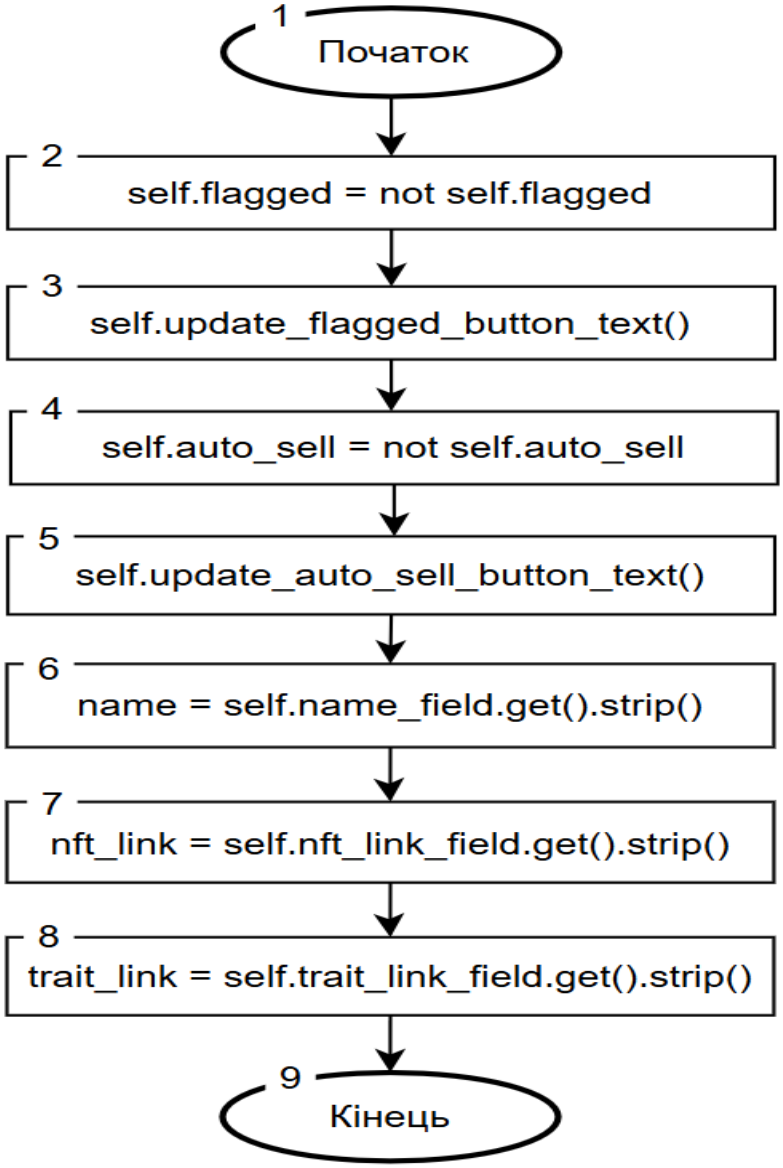


Рисунок 3.10 – Фрагмент класу AddWindow із методами save_nft, switch_flagged, switch_auto_sell

Структура інтерфейсу вікна додавання NFT наведена на рисунку 3.11.

Складові елементи вікна додавання NFT:

1. Назва.
2. Посилання на NFT.
3. Адреса метаданих NFT.
4. Мінімальна ціна.
5. Максимальна ціна.
6. Підрізання ціни.

7. Мінімальний прибуток.
8. Тривалість лістингу.
9. Інтервал затримки між запитами.
10. Режим перевірки лістингів.
11. Прапорець для активації режиму автоматичної продажі.
12. Кнопка «Зберегти NFT».
13. Кнопка «Скасувати».

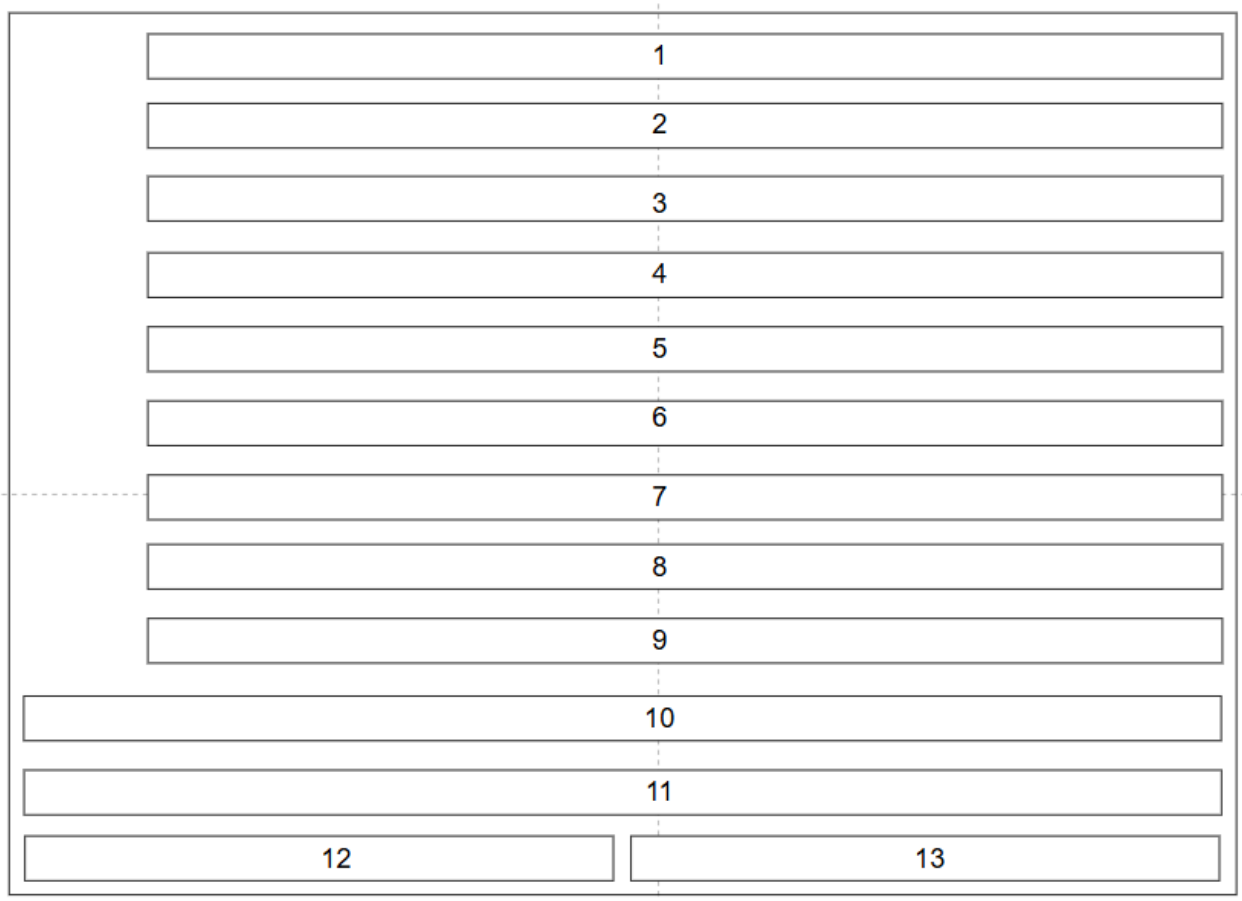


Рисунок 3.11 — Структурна схема вікна додавання NFT

Після подвійного кліку на будь-якому вже доданому елементі у головному списку відкривається вікно налаштувань для конкретного NFT. Поля вікна попередньо заповнюються значеннями з об’єкта, а метод save_changes() читає їх назад, перетворює у потрібні типи (Decimal, float, рядки) і записує в атрибути self.nft. При цьому перевірки (ValueError) гарантують, що користувач не введе

некоректні дані (наприклад, недозволена змінна min_profit у дешевшому плані). Фрагмент класу NFTSettingsWindow наведено на рисунку 3.12.

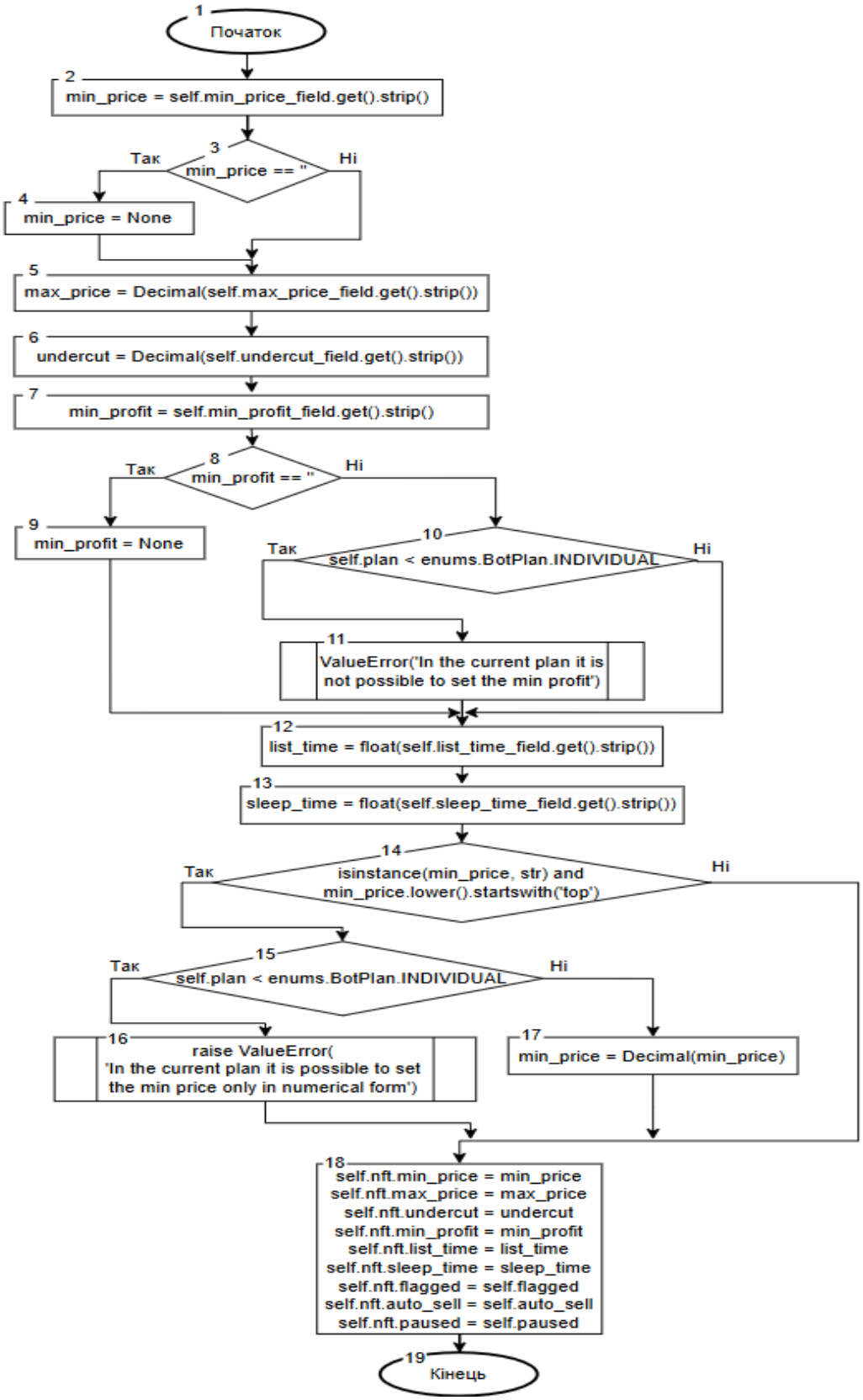


Рисунок 3.12 – Фрагмент класу NFTSettingsWindow із методом save_changes

Структурна схема інтерфейсу цього вікна наведена на рисунку 3.13.

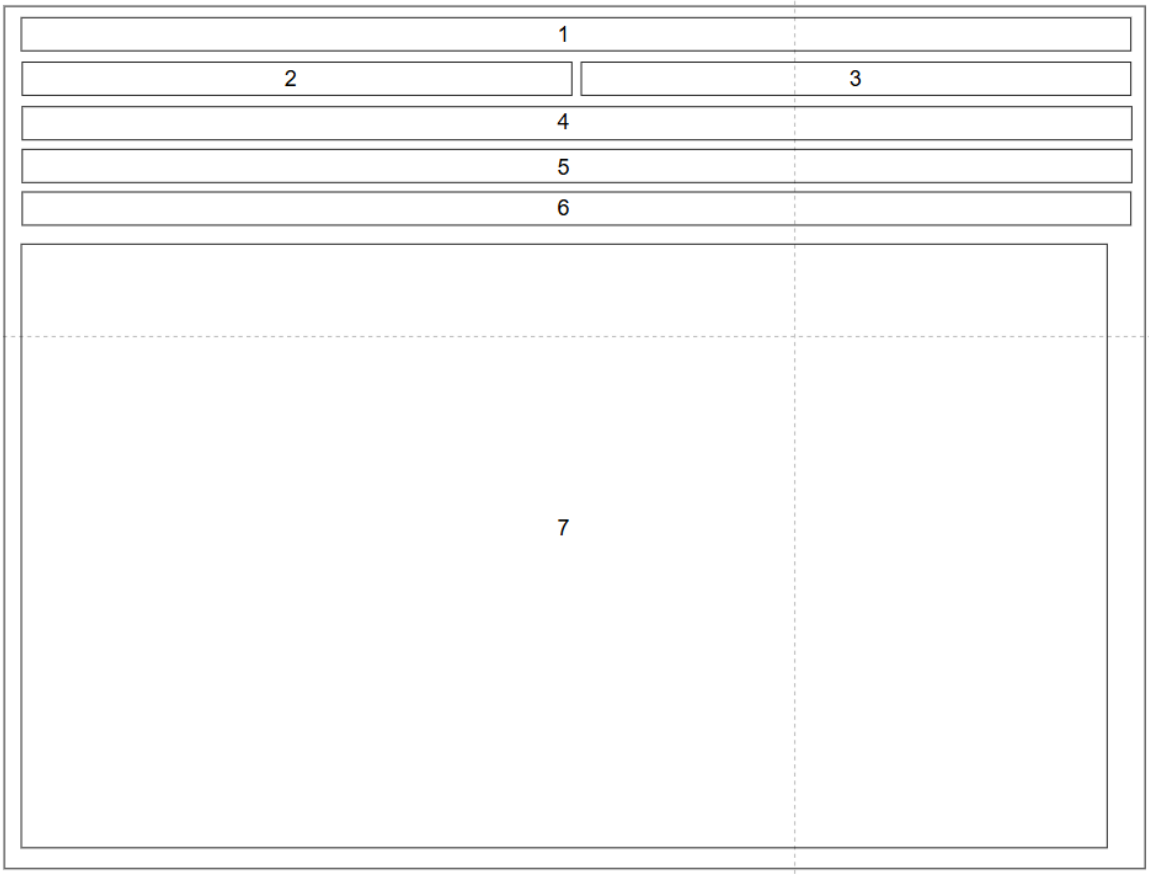


Рисунок 3.13 — Структурна схема вікна налаштування торгівлі

Складові вікна налаштування торгівлі:

1. Кнопка «Додати NFT».
2. Кнопка «Поставити на паузу».
3. Кнопка «Відновити операції».
4. Кнопка «Зберегти NFT у файл».
5. Кнопка «Отримати нові NFT та зберегти усі у файли».
6. Поле для пошуку NFT.
7. Список NFT.

Вікно налаштувань торгівлі служить для подальшого редагування вже доданого NFT, дозволяючи коригувати торговельну стратегію, умови ціноутворення та способи автоматизації, не потребуючи повторного створення запису.

Структурна схема інтерфейсу цього вікна наведена на рисунку 3.14.

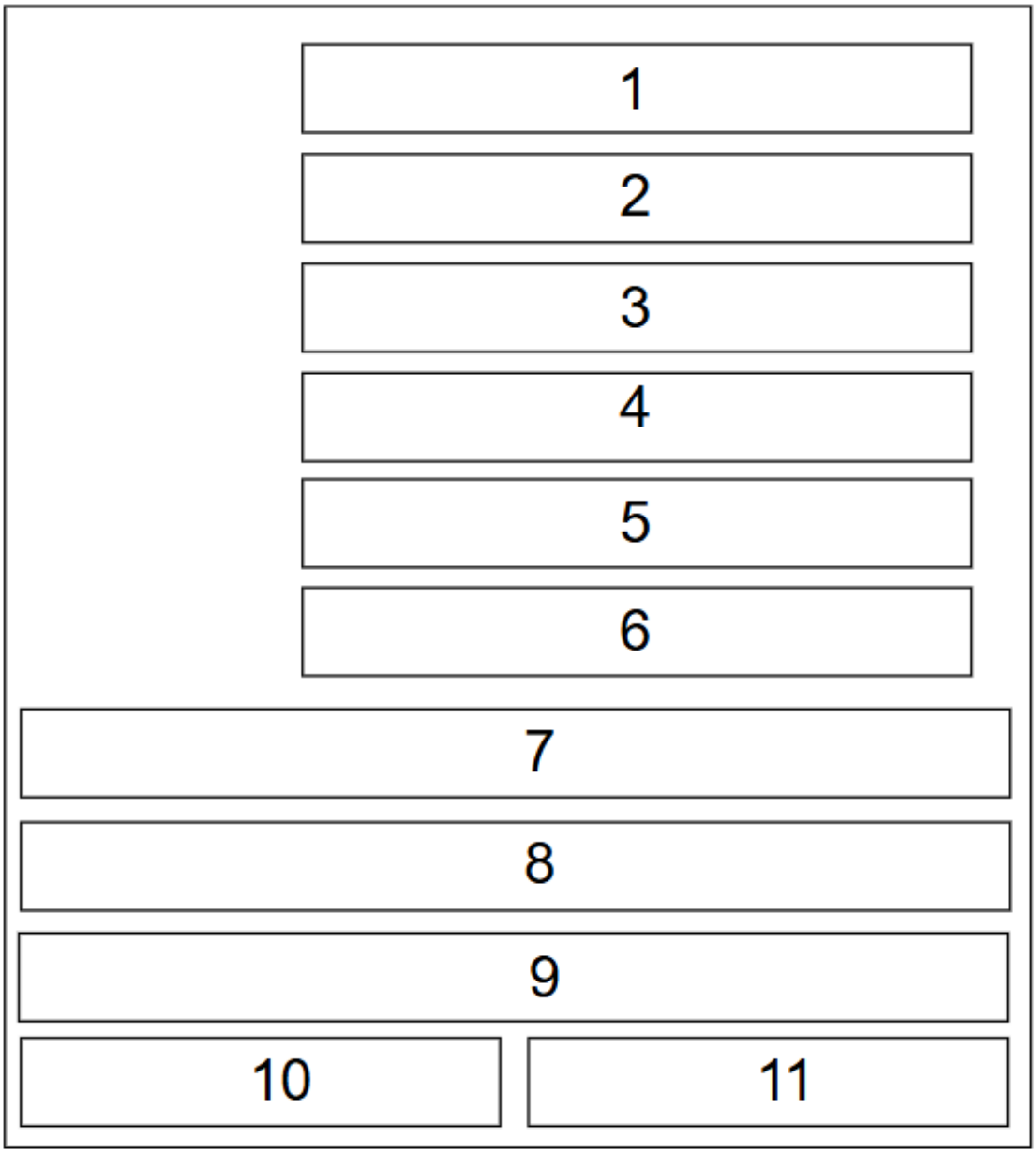


Рисунок 3.14 — Структурна схема вікна налаштувань торгівлі

Складові елементи вікна налаштувань торгівлі:

1. Мінімальна ціна.
2. Максимальна ціна.
3. «Підрізання» ціни.
4. Мінімальний прибуток
5. Тривалість лістингу.

6. Інтервал затримки між запитами.

7. Режим перевірки лістингів.

8. Прапорець для режиму автоматичного продажу.

9. Кнопка «Продовжити».

10. Кнопка «Зберегти».

11. Кнопка «Скасувати».

Таким чином, розроблений інтерфейс дозволяє зручно ініціювати та налаштовувати торгівлю, відстежувати результати торгівлі.

3.5 Висновки

У третьому розділі було проведено порівняльний аналіз мов програмування для розробки, було обрано Python з огляду на можливість її екосистеми бібліотек при роботі автоматизованої торгівлі NFT предметів, роботи з web3. Проведено порівняльний аналіз середовищ розробки, обрано PyCharm. Розроблено модуль бізнес-логіки та взаємодії з Blur API та модуль графічного інтерфейсу та управління налаштуваннями NFT.

4 ТЕСТУВАННЯ СИСТЕМИ

4.1 Аналіз методів тестування

Можна виділити чотири базові підходи до тестування автоматизованої системи NFT продажів на маркетплейсах: модульне, інтеграційне, end-to-end (E2E) та ручне. Кожен із них має свої сильні та слабкі сторони, але з огляду на специфіку взаємодії з асинхронними API, проксі-рутингом та блокчейн-транзакціями, найбільш доцільним виявляється саме ручне тестування.

Модульне тестування дозволяє ізолювати окремі функції та класи (наприклад, load_proxies(), _sign_challenge() чи логіку агрегації bids) і перевіряти їх поведінку на заздалегідь заданих вхідних даних. Воно забезпечує швидкий зворотний зв’язок і простоту регресій, але в цьому проєкті велика частина коду залежить від асинхронних викликів до віддалених сервісів та побічних ефектів (HTTP-запити, підпис транзакцій). Щоб покрити такі місця юнітами, доведеться маскувати (mock) багато зовнішніх залежностей, що ускладнить написання та підтримку тестів.

Інтеграційне тестування перевіряє взаємодію декількох компонентів разом — наприклад, відправлення запиту через _send_request() із реальним проксі та перевірка обробки статусів 401/403/429, або обробку ланцюжка EIP-712 підписів[19]. Воно ближче до «живої» роботи коду, але потребує або тестової інфраструктури Blur-сервера та Ethereum-вузла, або складного емулятора. Крім того, зміни у віддалених API можуть ламати інтеграційні тести, навіть якщо логіка клієнта не змінилася.

End-to-End (E2E) тестування намагається симулювати повний користувацький сценарій: від аутентифікації до публікації лістингу чи прийняття біду. Воно дає найвищу ступінь впевненості, але вимагає підготовки тестових акаунтів, коштів на гаманці та контролю середовища (щоб не створювати справжні транзакції у мережі Ethereum). Через асинхронність та зовнішні затримки та ризик непередбачуваних помилок у мережі такі тести будуть крихкими й повільними.

Ручне тестування передбачає, що розробник або тестувальник виконує ключові сценарії через скрипти: налаштовує проксі, виконує функції і візуально

перевіряє логи та результати в реальному середовищі. Цей підхід дозволяє:

- Оперативно реагувати на нетипові відповіді API та ліміти.

- Перевіряти справжню поведінку ротації проксі й обробку помилок у різних мережах.

- Гнучко змінювати параметри (ціни, час експірації) і одразу бачити результат без потреби переписувати мок-конфігурації.

- Фіксувати скриншоти логів або консолі для документування проблем.

З огляду на асинхронні та зовнішні залежності, різноманіття умов виконання (мережеві затримки, проксі-блокування, динамічні ліміти), ручне тестування в кожному релізі дозволить найшвидше виявляти реальні помилки, перевіряти поведінку в умовах, близьких до реальних, і уникати накладних витрат на підтримку великого набору емуляцій чи мок-тестів.

4.2 Тестування системи

Тестування системи розпочинається з головного її вікна, у якому міститься список NFT у власності, можливі для зберігання та додавання нових NFT, а також виставлення NFT на продаж. Головне вікно системи наведено на рисунку 4.1.

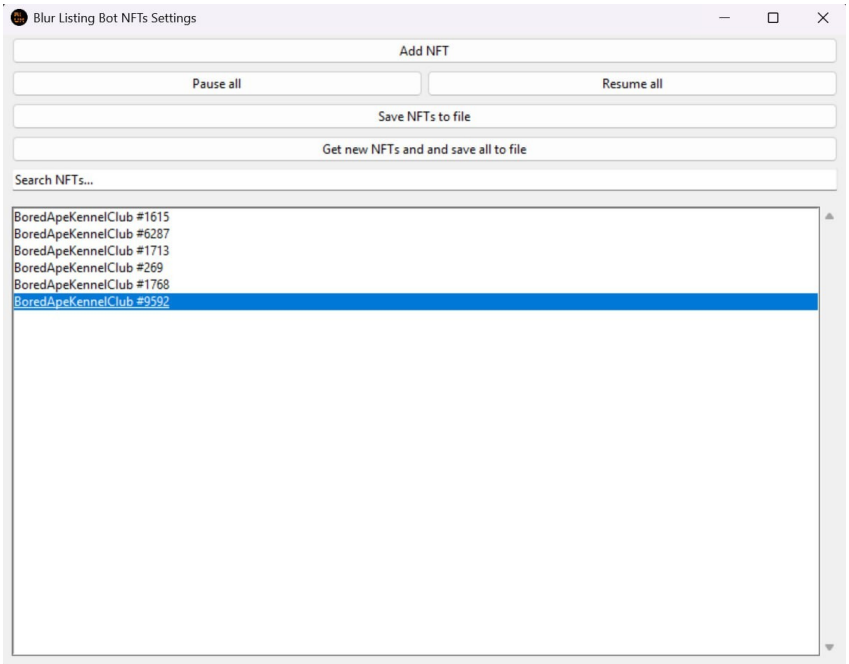


Рисунок 4.1 — Головне вікно застосунку

Оберемо NFT зі списку та відкриємо вікно його налаштувань для перегляду

детальної інформації. Результат наведено на рисунку 4.2.

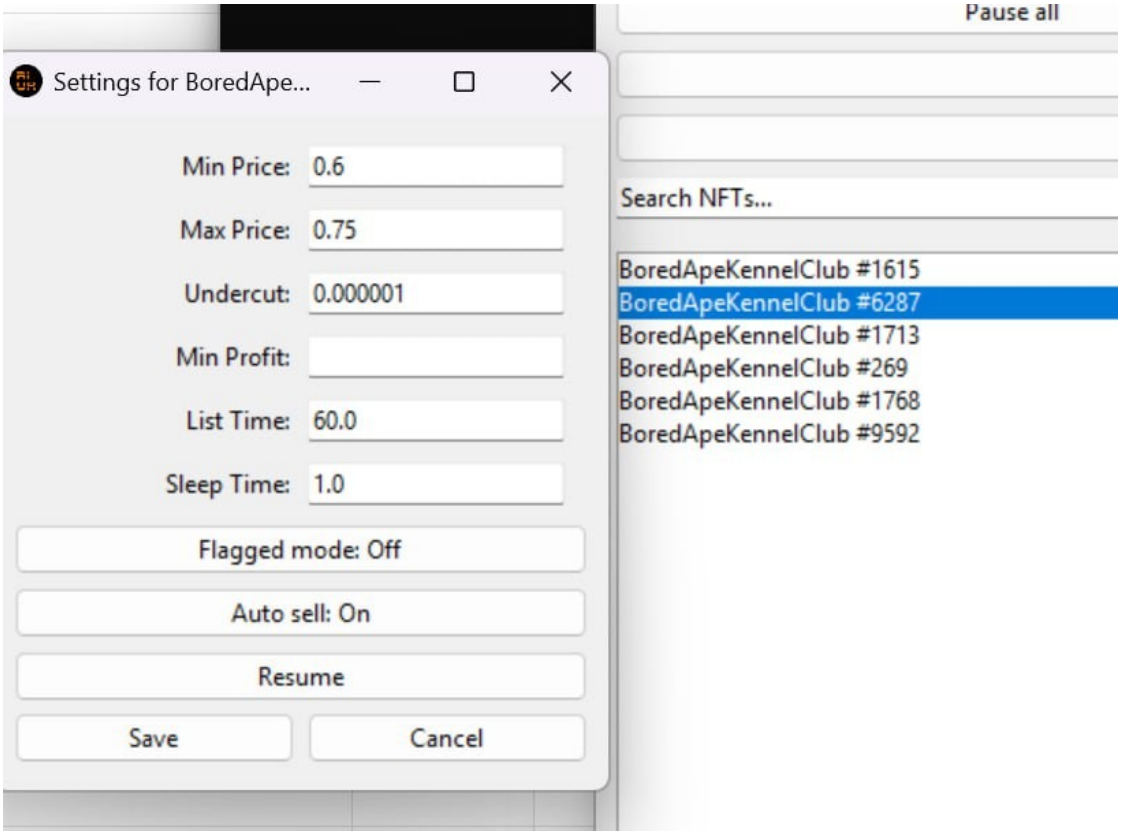


Рисунок 4.2 — Вікно налаштування торгівлі

З головного вікна застосунку натиснемо Get new NFTs and save all to file. У результаті буде збережено нові NFT. Результат продемонстровано на рисунку 4.3.

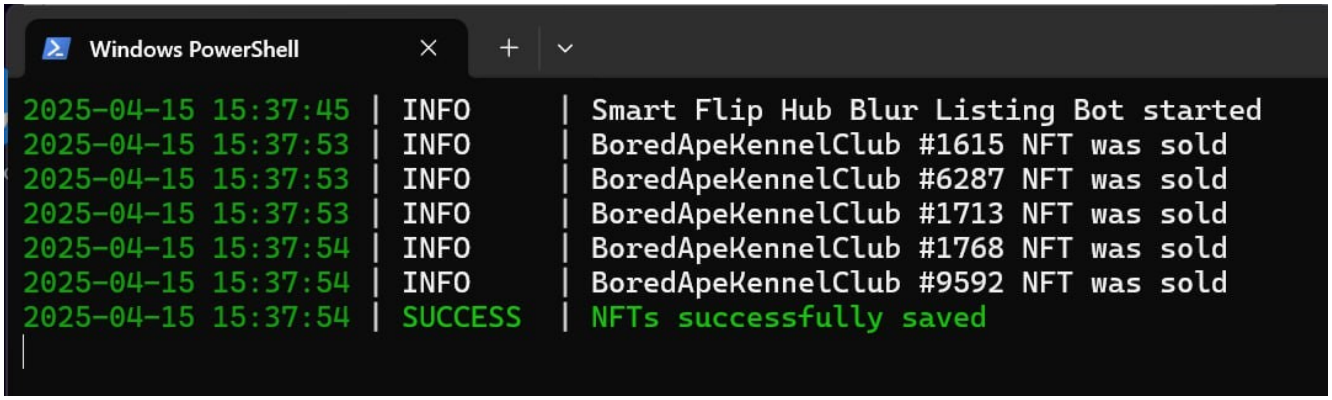
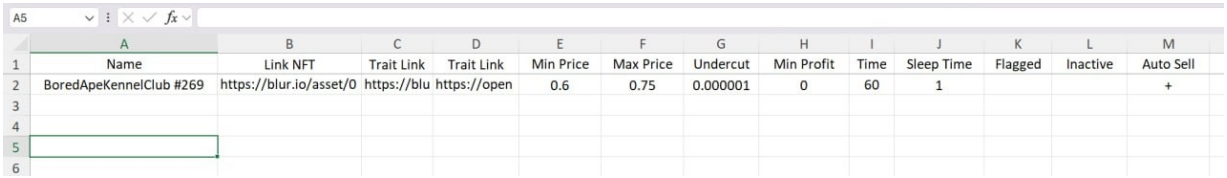


Рисунок 4.3 — Результат роботи кнопки Get new NFTs and save all to file

На рисунку 4.4 продемонстровано, що у таблиці для зберігання наявних NFT

відображається щойно додана NFT.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Name	Link NFT	Trait Link	Trait Link	Min Price	Max Price	Undercut	Min Profit	Time	Sleep Time	Flagged	Inactive	Auto Sell
2	BoredApeKennelClub #269	https://blur.io/asset/0	https://blu	https://open	0.6	0.75	0.000001	0	60	1			+
3													
4													
5													
6													

Рисунок 4.4 — Оновлена таблиця із наявними NFT

Виставимо це NFT на продаж. Результат продемонстровано на рисунку 4.5. У консолі відображається відповідне повідомлення про успішне виставлення на продаж.

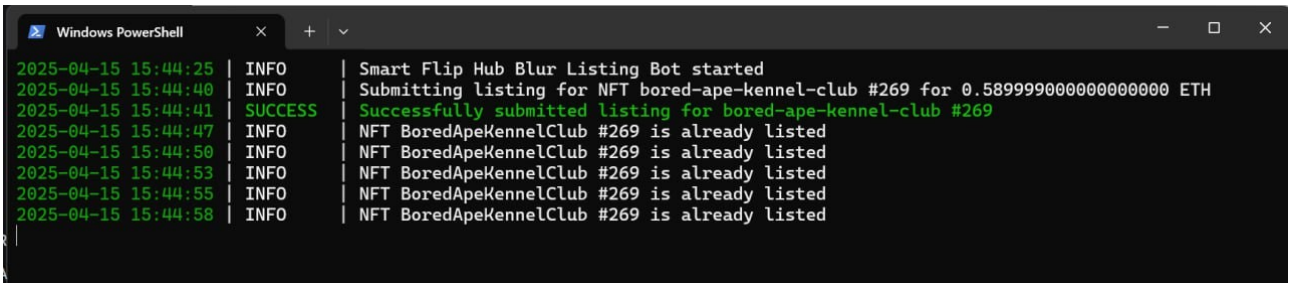


Рисунок 4.5 — Лістинг NFT на Blur

Переконаємось, що NFT справді виставлене на продаж, перейшовши на сайт Blur (див. рисунок 4.6).

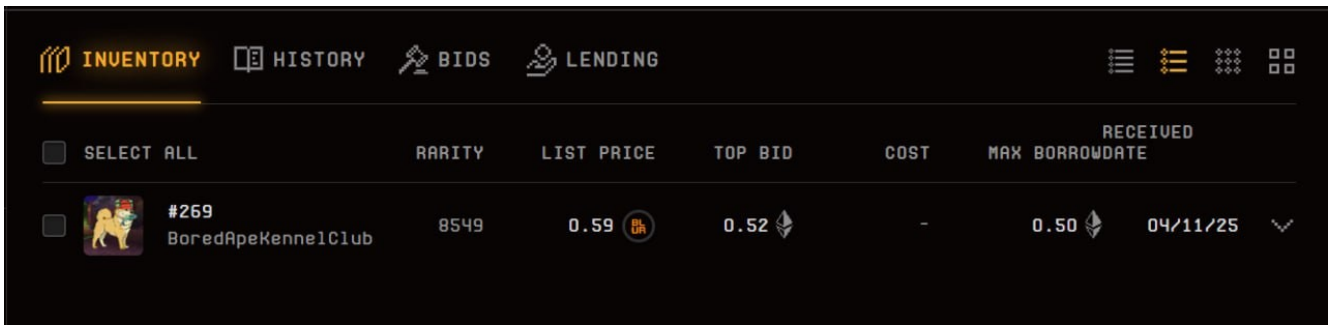


Рисунок 4.6 — Лістинг NFT на сайті Blur

Як ми можемо переконатися, NFT справді стоїть на продажі.

Для ефективної торгівлі реалізовано Telegram бота, який надсилає важливі

сповіщення, за допомогою яких можна відстежувати активність інших гравців ринку та оперативно реагувати для більш успішної торгівлі. Приклад таких сповіщень наведено на рисунку 4.7.

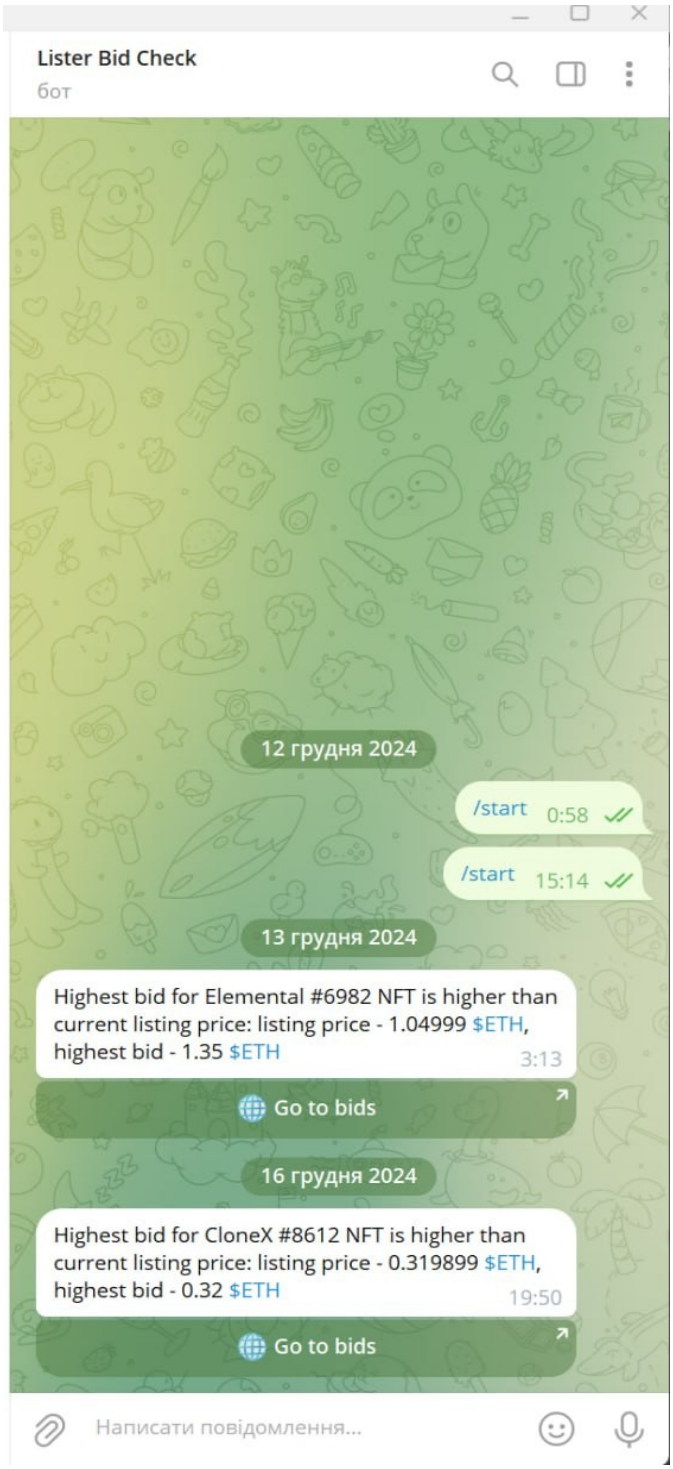


Рисунок 4.7 — Робота Telegram бота для сповіщень

На рисунку 4.8 продемонстровано вікно додавання нової NFT. Воно дозволяє

вводити його ціну, посилання, мінімальну та максимальну ціну, мінімальний прибуток.

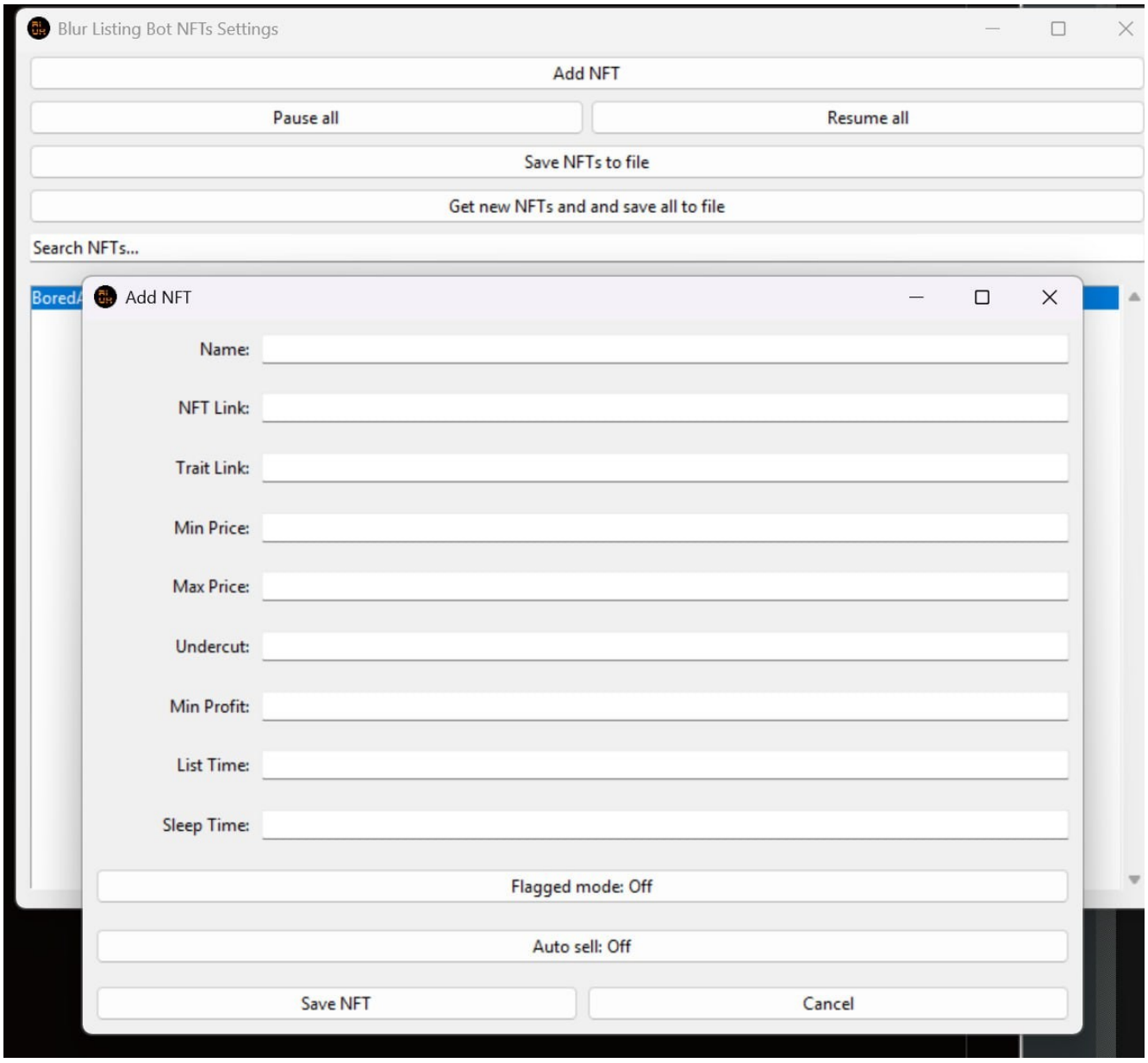


Рисунок 4.8 — Вікно додавання в бота нової NFT

Заповнимо це вікно необхідними даними про NFT, після чого додамо його до системи для подальшої роботи. Результат наведено на рисунку 4.9. Як ми можемо побачити, відображається відповідне повідомлення про успішне збереження NFT.

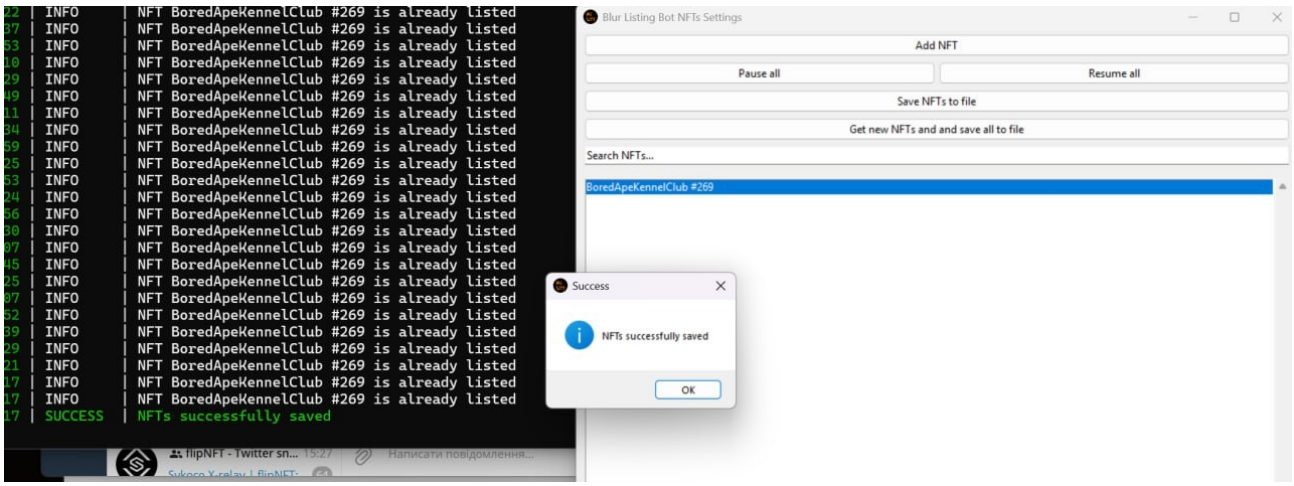


Рисунок 4.9 — Результат додавання нової NFT

Активність системи на сайті Blur продемонстровано на рисунку 4.10.

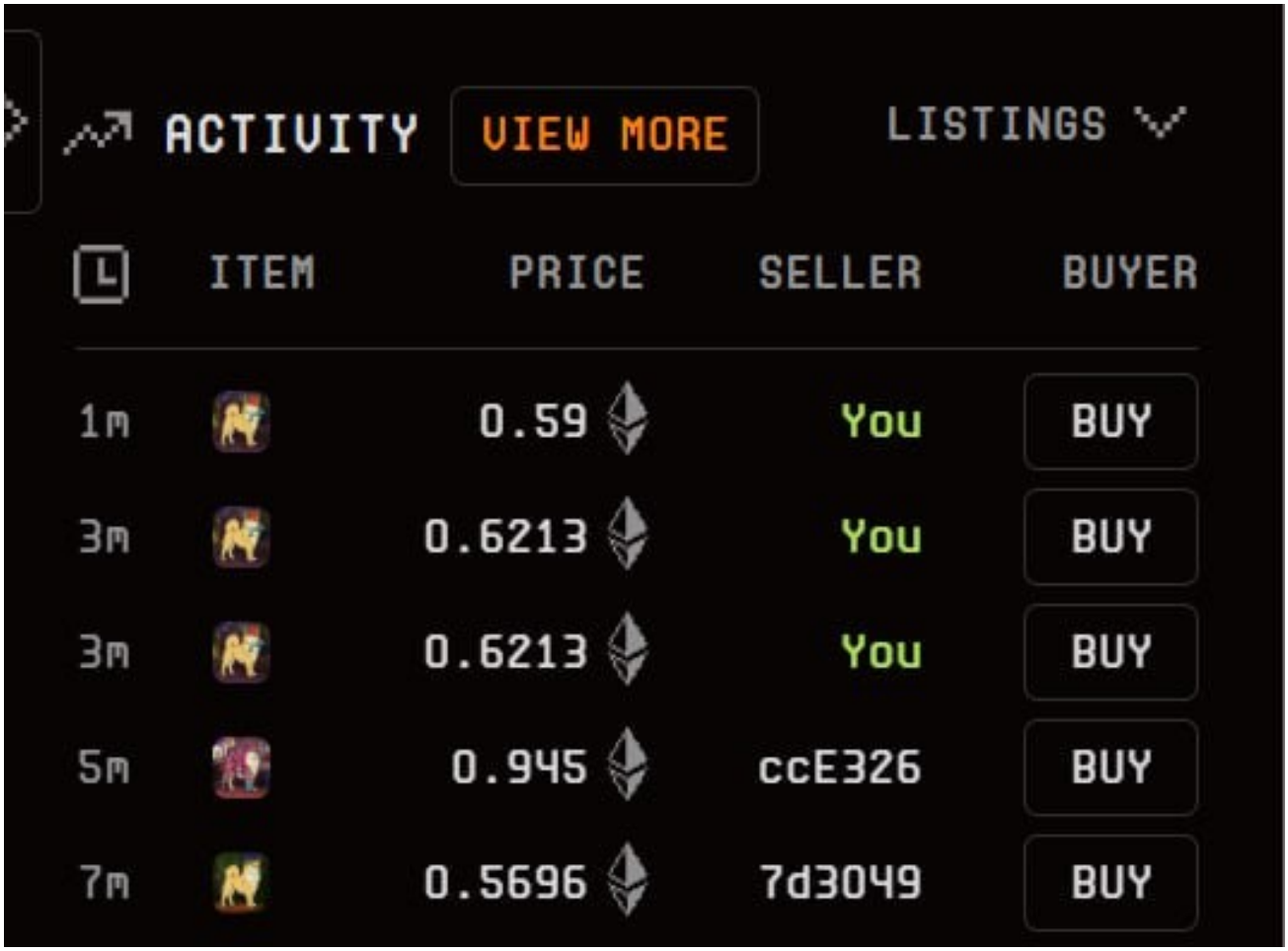


Рисунок 4.10 — Активність системи на сайті Blur

Активність системи на сайті OpenSea продемонстровано на рисунку 4.11.

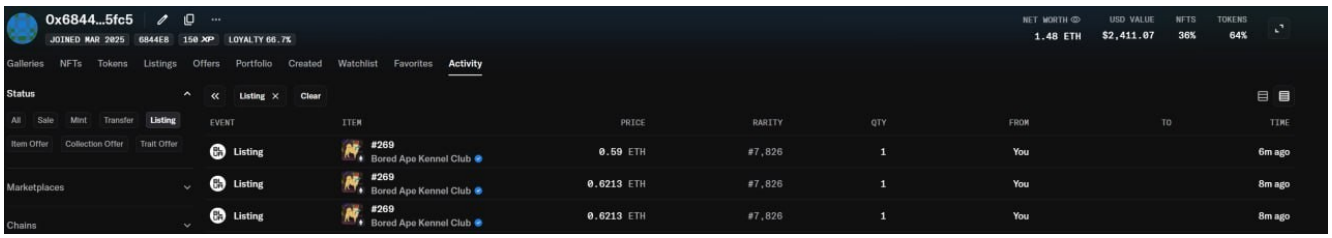


Рисунок 4.11 — Активність системи на сайті OpenSea

Таким чином, було проведено тестування автоматизованої системи торгівлі NFT предметами. Продемонстровано її функціонал, інтерфейс, приклади використання та доведено працездатність цієї системи. Розроблена система виконує поставлені на початку розробки завдання.

4.3 Висновки

У четвертому розділі було проведено аналіз методів тестування автоматизованої системи для NFT торгівлі, обрано метод ручного тестування. Було проведено тестування розробленої системи, продемонстровано її функціонал, доведено її працездатність та те, що розроблена система виконує поставлені на початку розробки завдання.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи були розроблені методи та програмні засоби автоматизованої системи моніторингу та продажу NFT на маркетплейсах OpenSea та Blur. Розроблені засоби дозволяють отримувати розгорнуту інформацію про колекції NFT та поточні лістинги, приймати найвигідніші пропозиції на продаж з налаштовуваним підбором плати за газ, отримувати й використовувати поточні ціни газу, зберігати та відновлювати конфігурацію NFT. Для розробки використовувалося середовище програмування PyCharm. Бакалаврську кваліфікаційну роботу реалізовано згідно методичних вказівок [20].

Під час виконання роботи було проведено аналіз стану питання розробки застосунків для автоматизованої торгівлі NFT, проведено порівняльний аналіз аналіз аналогів, продемонстровано переваги власної розробки. Проведено аналіз методів розв’язання задачі та постановку задач дослідження.

Під час аналізу технологій розробки було обґрунтовано вибір мови програмування Python.

У першому розділі було проведено аналіз стану питання розробки застосунків для автоматизованої торгівлі NFT, проведено порівняльний аналіз з такими застосунками: NFT Sniper Bot, Nansen Portfolio, Auto Listing Engine від NFT Butler. Було продемонстровано переваги власної розробки. Проведено аналіз методів розв’язання задачі, обрано метод з використанням готових API інтерфейсів. Проведено постановку задач дослідження.

У другому розділі було розроблено інформаційне забезпечення системи, було продемонстровано, які дані та як використовуються при автоматизованій торгівлі NFT предметами. Розроблено модель системи, наведено компоненти, що використовуються у системі. Розроблено алгоритми роботи системи, метод асинхронної черги з rate-limiting на рівні HTTP-методів та метод розумного добору пріоритетної плати за газ при прийнятті біду.

У третьому розділі було проведено порівняльний аналіз мов програмування

для розробки, було обрано Python з огляду на можливість її екосистеми бібліотек при роботі автоматизованої торгівлі NFT предметів, роботи з web3. Проведено порівняльний аналіз середовищ розробки, обрано PyCharm. Розроблено інтерфейс користувача системи.

У четвертому розділі було проведено аналіз методів тестування автоматизованої системи для NFT торгівлі, обрано метод ручного тестування. Було проведено тестування розробленої системи, продемонстровано її функціонал, доведено її працездатність та те, що розроблена система виконує поставлені на початку розробки завдання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. How Many NFTs Are Sold Every Day? [Електронний ресурс] — Режим доступу до ресурсу: https://playtoday.co/blog/stats/how-many-nfts-are-sold-every-day

2. State of Blockchain Gaming in Q1 2025 [Електронний ресурс] — Режим доступу до ресурсу: https://dappradar.com/blog/state-of-blockchain-gaming-in-q1-2025

3. Войтко В.В. Розробка засобів автоматизованої системи моніторингу, купівлі та продажу NFT на Маркетплейсах OPENSEA та BLUR / В.В. Войтко, Г.О. Черноволик, О.В. Гаврилюк, Н.Є. Барчук, М.В. Коберник // Матеріали LIV Всеукраїнської науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2025) : збірник доповідей [Електронний ресурс]. — Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24402/20094

4. NFT trading volume in 2024 [Електронний ресурс] — Режим доступу до ресурсу: https://www.theblock.co/post/334700/nft-trading-volume-2024

5. NFT Sniper Bot [Електронний ресурс] — Режим доступу до ресурсу: https://github.com/freesparrowrob/nft-sniper-bot

6. Nansen Portfolio [Електронний ресурс] — Режим доступу до ресурсу: https://app.nansen.ai/portfolio

7. Auto Listing Engine [Електронний ресурс] — Режим доступу до ресурсу: https://docs.nftbutler.io/nft-butler/features/auto-listing-engine

8. S. He, W. Ren, T. Zhu and K. -K. R. Choo, "BoSMoS: A Blockchain-Based Status Monitoring System for Defending Against Unauthorized Software Updating in Industrial Internet of Things," in IEEE Internet of Things Journal, vol. 7, no. 2, pp. 948-959, Feb. 2020, doi: 10.1109/JIOT.2019.2947339.

9. OpenSea API overview [Електронний ресурс] — Режим доступу до ресурсу: https://docs.opensea.io/reference/api-overview

10. Alchemy [Електронний ресурс] — Режим доступу до ресурсу: https://www

.alchemy.com/

11. Flashbots [Електронний ресурс] — Режим доступу до ресурсу: https://www.flashbots.net/

12. Understanding Ethereum Gas Fees [Електронний ресурс] — Режим доступу до ресурсу: https://www.kucoin.com/learn/web3/understanding-ethereum-gas-fees

13. Dilyan Grigorov. Introduction to Python and Large Language Models. Нью-Йорк: Apress Berkeley, 2024. 380с.

14. David Flanagan. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language. Farnham: O'Reilly Media, 2020. 704с.

15. Jim Blandy. Programming Rust: Fast, Safe Systems Development. Farnham: O'Reilly Media, 2021. 735с.

16. PyCharm [Електронний ресурс] — Режим доступу до ресурсу: https://www.jetbrains.com/pycharm/

17. VS Code [Електронний ресурс] — Режим доступу до ресурсу: https://code.visualstudio.com/

18.Neovim [Електронний ресурс] — Режим доступу до ресурсу: https://neovim.io/

19. V.Khorikov. Unit Testing Principles, Practices, and Patterns. Нью-Йорк: Manning, 2020. 304 с.

20. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О.Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 52с.

ДОДАТКИ

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«_21_» __березня__ 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка методу та програмних засобів автоматизованої системи моніторингу та продажу NFT на маркетплейсах OpenSea та Blur»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. каф. ПЗ Войтко В.В.
«_21_» __березня__ 2025 р.

Виконав:
студент гр. 1ПІ-21б Коберник М.В.
«_21_» __березня__ 2025 р.

м. Вінниця – 2025 рік

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка методу та програмних засобів автоматизованої системи моніторингу та продажу NFT на маркетплейсах OpenSea та Blur».

Галузь застосування – декстоп-технології.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 року про закріплення тем БКР.

3. Мета та призначення розробки.

Метою роботи є удосконалення процесу торгівлі NFT на маркетплейсах OpeanSea та Blur шляхом розробки спеціалізованої автоматизованої системи, яка дозволяє оптимізувати процеси моніторингу та торгівлі NFT.

4. Вихідні дані для проведення НДР

1. How Many NFTs Are Sold Every Day? [Електронний ресурс] — Режим доступу до ресурсу: https://playtoday.co/blog/stats/how-many-nfts-are-sold-every-day

2. OpenSea API overview [Електронний ресурс] — Режим доступу до ресурсу: https://docs.opensea.io/reference/api-overview

3. Understanding Ethereum Gas Fees [Електронний ресурс] — Режим доступу до ресурсу: https://www.kucoin.com/learn/web3/understanding-ethereum-gas-fees

4. Dilyan Grigorov. Introduction to Python and Large Language Models. Нью-Йорк: Apress Berkeley, 2024. 380с.

5. Технічні вимоги

Вхідні дані — приватний ключ, користувацькі налаштування.

Вихідні дані — інтерфейс користувача, логи та стани, транзакції.

6. Конструктивні вимоги.

Інтерфейс програми поовинен відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

–пояснювальна записка до бакалаврської кваліфікаційної роботи;

–технічне завдання;

–лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку систем NFT торгівлі	25.03.25 - 05.04.25
2	Розробка методів та алгоритмів програмного застосунку	06.04.25 - 18.04.25
3	Варіантний аналіз та вибір мови програмування розробки	19.04.25 - 21.04.25
4	Розробка застосунку	22.04.25 - 17.05.25
5	Тестування застосунку	18.05.25 - 25.05.25
6	Оформлення матеріалів до захисту БКР	26.05.25 - 30.05.25

10. Порядок контролю та прийняття.

Усі етапи бакалаврської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

Додаток Б. Протокол перевірки на наявність текстових запозичень

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка методу та програмних засобів автоматизованої системи моніторингу та продажу на маркетплейсах OpenSea та Blur»

Тип роботи: БКР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доц. каф. ПЗ Войтко В.В.

Оригінальність	95,3 %
Схожість	4,8 %

Аналіз звіту подібності

Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою StrikePlagiarism.

_____ Коберник М.В.

Автор роботи

_____ Войтко В.В.

Керівник роботи

ДОДАТОК В. Лістинг програми

```
import asyncio
import datetime as dt
import itertools
import json
import re
import typing
from collections import defaultdict
from decimal import ROUND_DOWN, Decimal

import jwt
from curl_cffi import requests
from curl_cffi.requests.exceptions import ConnectionError, Timeout
from eth_account.messages import encode_defunct, encode_typed_data
from eth_account.signers.local import LocalAccount
from jwt.exceptions import ExpiredSignatureError, InvalidTokenError
from web3 import AsyncWeb3

import enums
import utils
from bids import Bid
from config import BASE_DIR, RateLimit
from listing import Listing
from logger import logger
from my_queue import PriorityQueue, ProxyQueue
from request import Request


def load_proxies() -> list[str]:
    proxies = []

    proxies_filepath = BASE_DIR / 'proxies.txt'

    if proxies_filepath.exists():
        with open(proxies_filepath) as file:
            for line in file:
                line = line.strip()

                if re.match(r'.+:\d{1,5}:.+:.+', line):
                    ip, port, login, password = line.split(':')
                    proxy = f'http://{login}:{password}@{ip}:{port}'
                elif re.match(r'.+:.+@.+:\d{1,5}', line):
                    proxy = f'http://{line}'
                else:
                    raise ValueError(f'Invalid proxy format: {line}')
```

```
            proxies.append(proxy)

    return proxies


class Blur:
    def __init__(
        self,
        private_key: str,
        rate_limit: RateLimit,
        api_host: str,
        timeout: int,
        ethereum_node_url: str,
        max_requests: int
    ):
        self._private_key = private_key
        self._base_url = api_host
        self._queue = {}
        self._proxy_queue = ProxyQueue()
        self.sleep_time = {}
        self._last_request_time = {}

        self._proxies = load_proxies()

        if self._proxies:
            proxies_multiplier = len(self._proxies)
        else:
            proxies_multiplier = 1

        if max_requests is None:
            max_requests = float('inf')

        for method in enums.RequestMethod:
            self._last_request_time[method] = dt.datetime.fromtimestamp(0)
            self._queue[method] = PriorityQueue()
            self.sleep_time[method] = 1 / min((getattr(rate_limit, method.name) * proxies_multiplier), max_requests)

        for proxy in self._proxies:
            self._proxy_queue.put_nowait((0, proxy))

        self.timeout = timeout

        self._web3 = AsyncWeb3(
            AsyncWeb3.AsyncHTTPProvider(
                ethereum_node_url
            )
```

```
    )
    self._account: LocalAccount = self._web3.eth.account.from_key(private_key)

    self._cookies = {
        'authToken': None,
        'walletAddress': self.account_address.lower()
    }

    self.collection_fees = {}

    self.collection_addresses = {}

    self._chain_id = None

def check_auth_token(self, auth_token: str) -> bool:
    try:
        decoded = jwt.decode(
            jwt=auth_token,
            options={'verify_signature': False}
        )
    except ExpiredSignatureError as e:
        return False
    except InvalidTokenError as e:
        return False
    except Exception as e:
        logger.error(f'Failed to check auth token: {e}')
        return False
    else:
        wallet_address = decoded.get('walletAddress', None)
        expiration = decoded.get('exp', 0)

        refresh_token_dt = dt.datetime.now() + dt.timedelta(days=7)

        return wallet_address == self.account_address.lower() and expiration > refresh_token_dt.timestamp()

async def logined(self) -> bool:
    auth_token = self._cookies['authToken']

    if auth_token is None:
        return False
    else:
        return self.check_auth_token(auth_token)

async def login(self) -> bool:
    logined = await self.logined()

    if logined:
```

```
            return True

        challenge_json = await self._get_authchallenge()

        if challenge_json is False:
            return False

        signature = self._sign_challenge(challenge_json)

        challenge_json['signature'] = f'0x{signature}'

        accesstoken_response = await self._send_request(
            Request(
                method=enums.RequestMethod.POST,
                url=f'{self._base_url}/auth/login',
                json_data=challenge_json
            )
        )

        if not accesstoken_response.ok:
                logger.critical(f'Failed to get accesstoken: {accesstoken_response.status_code} {accesstoken_response.reason} {accesstoken_response.text}')
            return False

        accestoken_json = accesstoken_response.json()

        access_token = accestoken_json['accessToken']

        self._cookies['authToken'] = access_token

        return True

    async def _get_authchallenge(self) -> dict | bool:
        challenge_response = await self._send_request(
            Request(
                method=enums.RequestMethod.POST,
                url=f'{self._base_url}/auth/challenge',
                json_data={
                    'walletAddress': self.account_address
                }
            )
        )

        if not challenge_response.ok:
                    logger.critical(f'Failed to get challenge: {challenge_response.status_code} {challenge_response.reason} {challenge_response.text}')
            return False
```

```
            challenge_json = challenge_response.json()

            return challenge_json

        async def _sign_new_challenge(self) -> str | bool:
            challenge_json = await self._get_authchallenge()

            if challenge_json is False:
                return False

            return self._sign_challenge(challenge_json)

        def _sign_challenge(self, challenge: dict) -> str | bool:
            message = challenge['message']

            return self._sign_message(message=message)

        def _sign_message(self, message: str) -> str:
            signature = self._account.sign_message(encode_defunct(text=message))

            return signature.signature.hex()

        async def _get_cookies(self) -> dict:
            if not await self.login():
                raise ValueError('Failed to login')

            return self._cookies

        async def get_collection(self, collection_name: str) -> dict:
            collection_response = await self._send_request(
                Request(
                    method=enums.RequestMethod.GET,
                    url=f'{self._base_url}/v1/collections/{collection_name.lower()}'
                )
            )

            if not collection_response.ok:
                    raise ValueError(f'Failed to get collection info: {collection_response.status_code} {collection_response.reason}
{collection_response.text}')

            return collection_response.json()

        async def get_collection_address(self, collection_name: str) -> str:
            if collection_name not in self.collection_addresses:
                collection_json = await self.get_collection(collection_name)
```

```
            collection_address = collection_json['collection']['contractAddress']

            self.collection_addresses[collection_name] = collection_address

        return self.collection_addresses[collection_name]

    async def get_collection_fee(self, collection_name: str) -> Decimal:
        if collection_name not in self.collection_fees:
            fee_response = await self._send_request(
                Request(
                    method=enums.RequestMethod.GET,
                    url=f'{self._base_url}/v1/collections/{collection_name}/fees'
                )
            )

            if not fee_response.ok:
                raise ValueError(f'Failed to get collection fee: {fee_response.status_code} {fee_response.reason} {fee_response.text}')

            fee_json = fee_response.json()

            fee = Decimal(fee_json['fees']['byMarketplace']['BLUR']['minimumRoyaltyBips']) / 100

            self.collection_fees[collection_name] = fee

        return self.collection_fees[collection_name]

    async def get_collection_bids(self, collection_name: str, traits: dict = None):
        return await self._get_bids(
            collection_name=collection_name,
            traits=traits
        )

    async def _get_bids(
        self,
        wallet_address: str = None,
        collection_name: str = None,
        traits: dict = None
    ):
        if wallet_address is None and collection_name is not None:
            endpoint = f'collections/{collection_name.lower()}/executable-bids'
        elif wallet_address:
            endpoint = f'collection-bids/user/{wallet_address.lower()}'
        else:
            raise ValueError('wallet_address or collection_name must be specified')

        if collection_name is not None:
            collection_address = await self.get_collection_address(collection_name)
```

```
    if not collection_address:
        raise ValueError(f'Invalid collection name: {collection_name}')
    filters_template = {
        'contractAddress': collection_address
    }
else:
    filters_template = {}

filters_list = []

if traits is None:
    filters_template['criteria'] = {
        'type': 'COLLECTION',
        'value': {}
    }

    filters_list.append(filters_template)
else:
    combinations = list(itertools.product(*traits.values()))

    traits_list = [dict(zip(traits.keys(), combination)) for combination in combinations]

    for traits_combination in traits_list:
        filters_template['criteria'] = {
            'type': 'TRAIT',
            'value': traits_combination
        }

        filters_list.append(filters_template.copy())

all_bids = []

for filters in filters_list:
    bids_response = await self._send_request(
        Request(
            method=enums.RequestMethod.GET,
            url=f'{self._base_url}/v1/{endpoint}',
            params={
                'filters': json.dumps(filters)
            }
        )
    )

    if not bids_response.ok:
        raise ValueError(f'Failed to get bids: {bids_response.status_code} {bids_response.reason} {bids_response.text}')

    bids_json = bids_response.json()
```

```
        if not bids_json['success']:
            raise ValueError(f'Failed to get bids: {bids_json}')

        bids = []

        for price_level in bids_json['priceLevels']:
            bids.append(
                Bid(
                    collection_name=collection_name,
                    price=Decimal(price_level['price']),
                    amount=price_level['executableSize'],
                )
            )

        all_bids.extend(bids)

    combined_bids = defaultdict(int)

    for bid in all_bids:
        combined_bids[bid.price] += bid.amount

    all_bids = [
        Bid(
            collection_name=collection_name,
            price=price,
            amount=amount
        ) for price, amount in combined_bids.items()
    ]

    return sorted(all_bids, key=lambda x: x.price, reverse=True)

async def get_listings(
    self,
    collection_name: str,
    traits: dict
) -> list[Listing]:
    collection_address = await self.get_collection_address(collection_name)

    listings = []

    filters = {
        'cursor': None,
        'traits': [
            {
                'type': key,
                'values': value if isinstance(value, list) else [value]
```

```
                } for key, value in traits.items()
            ] if traits else [],
            'hasAsks': True
        }

        while True:
            listings_response = await self._send_request(
                Request(
                    method=enums.RequestMethod.GET,
                    url=f'{self._base_url}/v1/collections/{collection_address}/tokens',
                    params={
                        'filters': json.dumps(filters)
                    }
                )
            )

            if not listings_response.ok:
                raise ValueError(f'Failed to get collection listings: {listings_response.status_code} {listings_response.reason} {listings_response.text}')

            listings_json = listings_response.json()
            tokens = listings_json['tokens']

            listings.extend(
                [
                    Listing(
                        token_id=token['tokenId'],
                        owner=token['owner']['address'],
                        price=token['price']['amount'],
                        suspicious=token['isSuspicious'],
                        marketplace=enums.Marketplace.from_string(token['price']['marketplace'])
                    )
                    for token in tokens
                ]
            )

            total_count = listings_json['totalCount']

            if total_count <= len(listings):
                break

            try:
                filters['cursor'] = {
                    'price': {
                        'unit': tokens[-1]['price']['unit'],
                        'amount': tokens[-1]['price']['amount'],
                        'listedAt': tokens[-1]['price']['listedAt']
```

```
            },
            'tokenId': tokens[-1]['tokenId']
          }
        except IndexError:
          break

    return listings

async def get_token(self, collection_name: str, token_id: int) -> dict:
    collection_address = await self.get_collection_address(collection_name)

    if collection_address is False:
        raise ValueError(f'Invalid collection name: {collection_name}')

    token_response = await self._send_request(
        Request(
            method=enums.RequestMethod.GET,
            url=f'{self._base_url}/v1/collections/{collection_address}/tokens/{token_id}',
        )
    )

    if not token_response.ok:
        raise ValueError(f'Failed to get token info: {token_response.status_code} {token_response.reason} {token_response.text}')

    return token_response.json()

async def get_owned_tokens(self, collection_name: str = None) -> list[dict]:
    tokens = []

    cursor = None

    filters = {
        'filters': {}
    }

    if collection_name:
        collection_address = await self.get_collection_address(collection_name)

        if collection_address is False:
            raise ValueError(f'Invalid collection name: {collection_name}')

        filters['contractAddress'] = collection_address

    while True:
        if cursor:
            filters['cursor'] = cursor
```

```
            tokens_response = await self._send_request(
                Request(
                    method=enums.RequestMethod.GET,
                    url=f'{self._base_url}/v1/portfolio/{self.account_address.lower()}/owned',
                    params={
                        'filters': json.dumps(filters, separators=(',', ':'))
                    },
                    cookies=await self._get_cookies()
                )
            )

            if not tokens_response.ok:
                raise ValueError(f'Failed to get owned tokens: {tokens_response.status_code} {tokens_response.reason} {tokens_response.text}')

            tokens_json = tokens_response.json()

            if not tokens_json['success']:
                raise ValueError(f'Failed to get owned tokens: {tokens_json}')

            new_tokens = tokens_json['tokens']

            if not new_tokens or len(tokens) == tokens_json['totalCount']:
                break
            else:
                tokens.extend(new_tokens)

                last_token = new_tokens[-1]

                cursor = {
                    'contractAddress': last_token['contractAddress'],
                    'lastCostBasisAt': last_token['lastCostBasis']['listedAt'],
                    'tokenId': last_token['tokenId']
                }

        return tokens

    async def get_loans(self, collection_name: str) -> list[dict]:
        collection_address = await self.get_collection_address(collection_name)

        if collection_address is False:
            raise ValueError(f'Invalid collection name: {collection_name}')

        loans_response = await self._send_request(
            Request(
                method=enums.RequestMethod.GET,
                url=f'{self._base_url}/v1/portfolio/{self.account_address.lower()}/loans',
```

```
            params={
                'contractAddress': collection_address
            },
            cookies=await self._get_cookies()
        )
    )

    if not loans_response.ok:
        raise ValueError(f'Failed to get loans: {loans_response.status_code} {loans_response.reason} {loans_response.text}')

    return loans_response.json()['loans']

async def submit_listing(
    self,
    collection_name: str,
    token_id: int,
    price: Decimal,
    list_time: float,
    lien_id: int = None
):
    collection_address = await self.get_collection_address(collection_name)

    if collection_address is False:
        raise ValueError(f'Invalid collection name: {collection_name}')

    price = price.quantize(Decimal('1.000000000000000000'), rounding=ROUND_DOWN)

    logger.info(f'Submitting listing for NFT {collection_name} #{token_id} for {price} ETH')

    now = dt.datetime.now(tz=dt.timezone.utc)
    expiration_time = now + dt.timedelta(minutes=list_time)

    collection_fee = await self.get_collection_fee(collection_name)

    format_json = {
        'marketplace': 'BLUR',
        'orders': [
            {
                'price': {
                    'amount': str(price),
                    'unit': 'ETH'
                },
                'tokenId': str(token_id),
                'contractAddress': collection_address,
                'expirationTime': expiration_time.strftime("%Y-%m-%dT%H:%M:%S.%f")[:-3] + 'Z',
                'feeRate': int(collection_fee * 100)
            }
```

```
    ]
}

if lien_id is not None:
    format_json['orders'][0]['lienId'] = str(lien_id)

format_response = await self._send_request(
    Request(
        method=enums.RequestMethod.POST,
        url=f'{self._base_url}/v1/orders/format',
        cookies=await self._get_cookies(),
        json_data=format_json,
        priority=dt.datetime.now().timestamp() / 100
    )
)

if not format_response.ok:
    logger.error(f'Failed to format order: {format_response.status_code} {format_response.reason} {format_response.text}')
    return False

format_json = format_response.json()

marketplace_data = format_json['signatures'][0]['marketplaceData']
marketplace = format_json['signatures'][0]['marketplace']
sign_data = format_json['signatures'][0]['signData']

sign_data['types']['EIP712Domain'] = [
    {
        'name': 'name',
        'type': 'string'
    },
    {
        'name': 'version',
        'type': 'string'
    },
    {
        'name': 'chainId',
        'type': 'uint256'
    },
    {
        'name': 'verifyingContract',
        'type': 'address'
    }
]
sign_data['primaryType'] = list(sign_data['types'].keys())[0]
message = sign_data.pop('value')
sign_data['message'] = message
```

```
for key, value in message.items():
    if isinstance(value, dict) and 'type' in value:
        if value['type'] == 'BigNumber':
            message[key] = int(value['hex'], 0)
        else:
            raise ValueError(f'Unknown type: {value["type"]}')
    else:
        message[key] = value

signature = self._account.sign_message(
    encode_typed_data(full_message=sign_data)
).signature.hex()

approvals = format_json['approvals']

for approval in approvals:
    gas_price = await self.get_gas_price()

    txn = {
        'chainId': await self._web3.eth.chain_id,
        'nonce': await self._web3.eth.get_transaction_count(self.account_address),
        'from': approval['transactionRequest']['from'],
        'to': approval['transactionRequest']['to'],
        'gas': 0,
        **gas_price,
        'value': 0,
        'data': approval['transactionRequest']['data']
    }

    try:
        txn['gas'] = await self._web3.eth.estimate_gas(txn)
    except Exception as e:
        if 'insufficient funds' in str(e):
            logger.critical(f'Insufficient balance to approve collection {collection_name}')
            return False
        logger.exception(f'Exception occured while estimating gas: {e}')
        return False

    signed_txn = self._account.sign_transaction(txn)

    txn_hash = await self._web3.eth.send_raw_transaction(signed_txn.rawTransaction)

    logger.info(f'Approve transaction: https://etherscan.io/tx/{txn_hash.hex()}')

    receipt = await self._web3.eth.wait_for_transaction_receipt(txn_hash)
```

```
            if receipt['status'] == 1:
                logger.success(f'Successfully approved collection {collection_name}')
            else:
                logger.error(f'Failed to approve collection {collection_name}')

        submit_dict = {
            'signature': f'0x{signature}',
            'marketplace': marketplace,
            'marketplaceData': marketplace_data
        }

        submit_response = await self._send_request(
            Request(
                method=enums.RequestMethod.POST,
                url=f'{self._base_url}/v1/orders/submit',
                cookies=await self._get_cookies(),
                json_data=submit_dict,
                priority=dt.datetime.now().timestamp() / 100
            )
        )

        if not submit_response.ok:
                    logger.error(f'Failed to submit order for {collection_name} #{token_id}: {submit_response.status_code}
{submit_response.reason} {submit_response.text}')
            return False

        submit_json = submit_response.json()

        if submit_json['success']:
            logger.success(f'Successfully submitted listing for {collection_name} #{token_id}')
            return True
        else:
            logger.error(f'Failed to submit listing for {collection_name} #{token_id}: {submit_json}')
            return False

    async def accept_highest_bid(
        self,
        collection_name: str,
        token_id: int,
        min_price: Decimal,
        max_priority_fee: typing.Optional[Decimal | dict[Decimal, Decimal]],
        auto_sell: bool
    ) -> bool | int:
        collection_address = await self.get_collection_address(collection_name)

        if collection_address is False:
            raise ValueError(f'Invalid collection name: {collection_name}')
```

```
            quote_dict = {
                'contractAddress': collection_address,
                'tokens': [
                    {
                        'tokenId': str(token_id)
                    }
                ]
            }

            quote_response = await self._send_request(
                Request(
                    method=enums.RequestMethod.POST,
                    url=f'{self._base_url}/v1/bids/quote',
                    json_data=quote_dict,
                    cookies=await self._get_cookies(),
                )
            )

            if not quote_response.ok:
                logger.error(f'Failed to get quote for {collection_name} #{token_id}: {quote_response.status_code} {quote_response.reason} {quote_response.text}')
                return False

            quote_json = quote_response.json()

            if not quote_json['success']:
                logger.error(f'Failed to get quote for {collection_name} #{token_id}: {quote_json}')
                return False

            prices = [Decimal(nft['price']['amount']) for nft in quote_json['nfts']]

            if not prices:
                return False

            sale_price = max(prices)

            if not auto_sell:
                return sale_price

            base_profit = sale_price - min_price

            if base_profit <= 0:
                return False

            required_gas = 250000
```

```
                gas_price = await utils.suggest_gas_fees(await self.chain_id)

                if max_priority_fee is None:
                    max_priority_fee_per_gas = 0
                elif isinstance(max_priority_fee, Decimal):
                    max_priority_fee_per_gas = max_priority_fee
                else:
                    max_priority_fee_per_gas = None

                    for profit_level, value in sorted(max_priority_fee.items(), reverse=True):
                     profit = base_profit - AsyncWeb3.from_wei(required_gas * (value + AsyncWeb3.from_wei(gas_price['maxFeePerGas'],
'gwei')), 'gwei')

                        if profit >= profit_level:
                            max_priority_fee_per_gas = value
                            break
                    else:
                        return False

                logger.info(f'Accepting highest bid for {collection_name} #{token_id} for {sale_price} ETH with max priority fee per gas
{max_priority_fee_per_gas} Gwei')

                collection_fee = await self.get_collection_fee(collection_name)

                quote_dict.update(
                    {
                        'feeRate': int(collection_fee * 100),
                        'quoteId': quote_json['quoteId']
                    }
                )

                accept_response = await self._send_request(
                    Request(
                        method=enums.RequestMethod.POST,
                        url=f'{self._base_url}/v1/bids/accept',
                        json_data=quote_dict,
                        cookies=await self._get_cookies(),
                        priority=dt.datetime.now().timestamp() / 100
                    )
                )

                if not accept_response.ok:
                        logger.error(f'Failed to accept highest bid for {collection_name} #{token_id}: {accept_response.status_code}
{accept_response.reason} {accept_response.text}')
                    return False

                accept_json = accept_response.json()
```

```
approvals = accept_json['approvals']

for approval in approvals:
    gas_price = await self.get_gas_price()

    txn = {
        'chainId': await self._web3.eth.chain_id,
        'nonce': await self._web3.eth.get_transaction_count(self.account_address),
        'from': approval['transactionRequest']['from'],
        'to': approval['transactionRequest']['to'],
        'gas': 0,
        **gas_price,
        'value': 0,
        'data': approval['transactionRequest']['data']
    }

    try:
        txn['gas'] = await self._web3.eth.estimate_gas(txn)
    except Exception as e:
        if 'insufficient funds' in str(e):
            logger.critical(f'Insufficient balance to approve collection {collection_name}')
            return False
        logger.exception(f'Exception occured while estimating gas: {e}')
        return False

    signed_txn = self._account.sign_transaction(txn)

    txn_hash = await self._web3.eth.send_raw_transaction(signed_txn.rawTransaction)

    logger.info(f'Approve transaction: https://etherscan.io/tx/{txn_hash.hex()}')

    receipt = await self._web3.eth.wait_for_transaction_receipt(txn_hash)

    if receipt['status'] == 1:
        logger.success(f'Successfully approved collection {collection_name}')
    else:
        logger.error(f'Failed to approve collection {collection_name}')

max_priority_fee_per_gas = AsyncWeb3.to_wei(max_priority_fee_per_gas, 'gwei')

gas_price['maxFeePerGas'] = gas_price['maxFeePerGas'] + max_priority_fee_per_gas
gas_price['maxPriorityFeePerGas'] = max_priority_fee_per_gas

accepts = accept_json['accepts']

if not accepts:
```

```
            logger.error(f'Failed to accept bid for collection {collection_name}: {accept_json}')
            return False

        for accept in accepts:
            txn = {
                'chainId': await self._web3.eth.chain_id,
                'nonce': await self._web3.eth.get_transaction_count(self.account_address),
                'from': self.account_address,
                'to': accept['txnData']['to'],
                **gas_price,
                'data': accept['txnData']['data']
            }

            try:
                txn['gas'] = await self._web3.eth.estimate_gas(txn)
            except Exception as e:
                if 'insufficient funds' in str(e):
                    logger.critical(f'Insufficient balance to accept bid for collection {collection_name}')
                    return False
                logger.exception(f'Exception occured while estimating gas: {e}')
                return False

            signed_txn = self._account.sign_transaction(txn)

            txn_hash = await self._web3.eth.send_raw_transaction(signed_txn.rawTransaction)

            logger.info(f'Accept transaction: https://etherscan.io/tx/{txn_hash.hex()}')

            receipt = await self._web3.eth.wait_for_transaction_receipt(txn_hash)

            if receipt['status'] == 1:
                logger.success(f'Successfully accepted bid for collection {collection_name}')
            else:
                logger.error(f'Failed to accept bid for collection {collection_name}')
                return False

    async def _get_proxy(self) -> str | None:
        if self._proxies:
            return await self._proxy_queue.get_least_used_proxy()

        return None

    async def _send_request(self, request: Request) -> requests.models.Response:
        queue = self._queue[request.method]

        try:
            await queue.put((request.priority, request))
```

```
    while True:
        diff = (dt.datetime.now() - self._last_request_time[request.method]).total_seconds()

        if queue.peek() == request and diff >= self.sleep_time[request.method]:
            break

        await asyncio.sleep(self.sleep_time[request.method] / 10)
except Exception as e:
    raise e
finally:
    self._last_request_time[request.method] = dt.datetime.now()

    while queue.peek() == request:
        await queue.get()

proxy = await self._get_proxy()

try:
    async with requests.AsyncSession() as session:
        response: requests.models.Response = await session.request(
            method=request.method.value,
            url=request.url,
            headers=request.headers,
            json=request.json_data,
            params=request.params,
            cookies=request.cookies,
            timeout=self.timeout,
            proxy=proxy,
            impersonate='safari17_2_ios'
        )
except Timeout:
    logger.warning(f'Request {request.identifier} timed out')
    return await self._send_request(request=request)
except ConnectionError as ce:
    logger.warning(f'Failed to execute request with proxy {proxy}: {ce}')
    return await self._send_request(request=request)
else:
    if response.status_code == 429:
        logger.warning(f'Rate limited for request with ID {request.identifier}: {response.status_code} {response.reason}')
        return await self._send_request(request)
    elif response.status_code == 403:
        logger.warning(f'Forbidden for request with ID {request.identifier}: {response.status_code} {response.reason}')
        return await self._send_request(request)
    elif response.status_code == 401:
        if await self.login():
            return await self._send_request(request)
```

```
        return response

async def get_gas_price(self) -> dict[str, float]:
    gas_price = await utils.suggest_gas_fees(chain_id=await self._web3.eth.chain_id)
    if gas_price is None:
        gas_price = {
            'gasPrice': await self._web3.eth.gas_price
        }
    return gas_price

@property
def private_key(self) -> str:
    return self._private_key

@property
def account_address(self) -> str:
    return self._account.address

@property
async def chain_id(self):
    if self._chain_id is None:
        self._chain_id = await self._web3.eth.chain_id
    return self._chain_id
```

ДОДАТОК Г. Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МЕТОДУ ТА ПРОГРАМНИХ ЗАСОБІВ АВТОМАТИЗОВАНОЇ СИСТЕМИ МОНІТОРИНГУ ТА ПРОДАЖУ NFT НА МАРКЕТПЛЕЙСАХ OPENSEA ТА BLUR

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота на тему:
«Розробка методу та програмних засобів автоматизованої системи моніторингу та продажу NFT на маркетплейсах OpenSea та Blur»

Автор: ст.групи 1ПІ-21б Коберник М.В.
Науковий керівник: к.т.н., доцент каф. ПЗ Войтко В.В.

Вінниця – 2025

Рисунок Г.1 – Титульний слайд

Актуальність теми

Торгівля NFT предметами перебуває у стадії активного розвитку з моменту свого заснування у 2021 році, та, попри певну волатильність, продовжує демонструвати значний ріст, досягши рівня обсягу торгівлі у мільярди доларів щорічно. Щоденно проводяться тисячі транзакцій, що є значним обсягом торгівлі, тому, для прийняття успішних рішень щодо покупки чи продажу предметів NFT необхідний не лише якісний, а і швидкий аналіз, який стає уже неможливо провести вручну, без використання автоматизованих систем, які проводитимуть аналіз в автоматичному режимі за короткий проміжок часу. Такі автоматизовані системи моніторингу та торгівлі надають своїм користувачам значну конкурентну перевагу, дозволяючи оперативно виявляти потенційно прибуткові та цінні активи, реагувати на зміни цін та використовувати можливості арбітражу між різними платформами для торгівлі. Тому актуальною є розробка такої системи.

Особливо актуальною вона є з огляду на те, що ринок NFT переходить стадію розвитку з спекулятивного до екосистеми з різноманітним реально корисним застосуванням. NFT можна використовувати не лише як цифрове мистецтво, а й як функціональні активи в метавсесвітах та ігрових системах, де все більше використовуються NFT токени для залучення гравців та підвищення мотивації грати ігри, адже під час гри можна отримувати NFT предмети, які мають реальну вартість, та на продажу яких можна заробити реальні гроші, що трансформує всю сферу ігор із джерела витрат грошей на джерело їх заробітку для найуспішніших гравців.

Рисунок Г.2 – Актуальність теми

Мета, об’єкт та предмет дослідження

Мета та завдання дослідження. Метою роботи є удосконалення процесу торгівлі NFT на маркетплейсах OpeanSea та Blur шляхом розробки спеціалізованої автоматизованої системи, яка дозволяє оптимізувати процеси моніторингу та торгівлі NFT.
Об’єктом дослідження є процес розробки програмних засобів автоматизованої системи моніторингу, купівлі та продажу NFT на маркетплейсах OpenSea та Blur.
Предметом дослідження є методи і засоби реалізації автоматизованої системи моніторингу, купівлі та продажу NFT на маркетплейсах OpenSea та Blur.

Рисунок Г.3 – Мета, об'єкт та предмет дослідження

Наукова новизна

1. Подальшого розвитку отримав метод асинхронної черги з rate-limiting на рівні HTTP-методів, який, на відміну від відомих, гарантує, що запити одного і того ж методу не перевищують заданого ліміту на одиницю часу, що забезпечує стабільну роботу в умовах мінливих лімітів і ненадійних проксі без втручання розробника.
2. Подальшого розвитку отримав метод розумного добору пріоритетної плати за газ при прийнятті біду, який, на відміну від відомих, дозволяє обрати таку величину, при якій очікуваний чистий прибуток від продажу не опуститься нижче заданого мінімального рівня.

Рисунок Г.4 – Наукова новизна

Завдання дослідження

Основними задачами дослідження є:
- розробити архітектуру автоматизованої системи NFT продажів на маркетплейсах;
- розробити алгоритми роботи автоматизованої системи NFT продажів на маркетплейсах;
- розробити функціонал системи;
- розробити інтерфейс користувача застосунку;
- провести тестування розробленої системи.

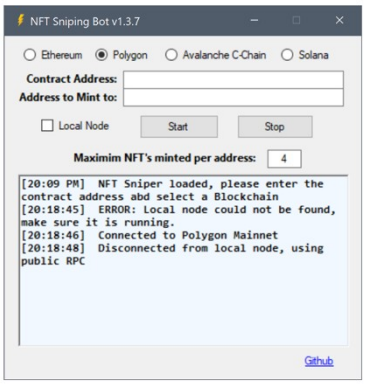
Рисунок Г.5 – Завдання дослідження

Практична цінність отриманих результатів

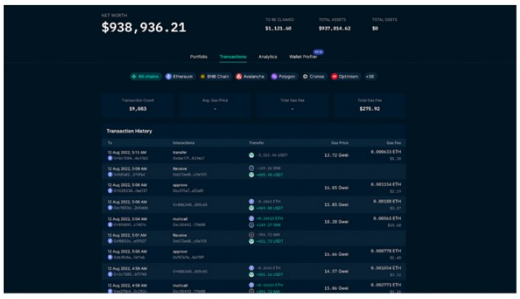
Практична цінність одержаних результатів полягає в можливості використання розроблених програмних засобів для автоматизованого моніторингу, купівлі та продажу NFT.

Рисунок Г.6 – Практична цінність отриманих результатів

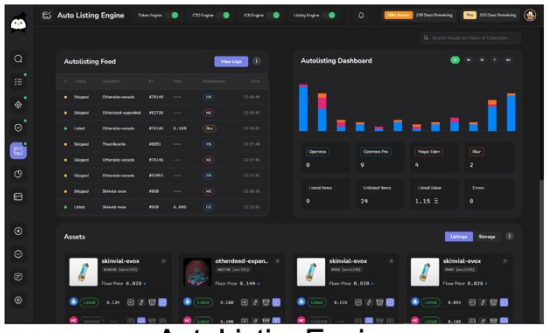
Аналоги



NFT Sniper Bot



Nansen Portfolio



AutoListingEngine

Рисунок Г.7 – Аналоги

Порівняльний аналіз аналогів

Характеристика	NFT Sniper Bot	Nansen Portfolio	Auto Listing Engine	Власна розробка
Інтеграція з OpenSea	+	+	-	+
Інтеграція з Blur	+	+	+	+
Автоматичне виконання транзакцій	+	-	+	+
Підтримка найпопулярніших блокчейнів	+	+	-	+
Налаштування ризиків	-	-	-	+
Звітність	-	+	+	+
Аналіз рідкісних NFT предметів	+	+	+	+
Сумарний бал	5	5	4	7

Рисунок Г.8 – Порівняльний аналіз аналогів

Використані технології

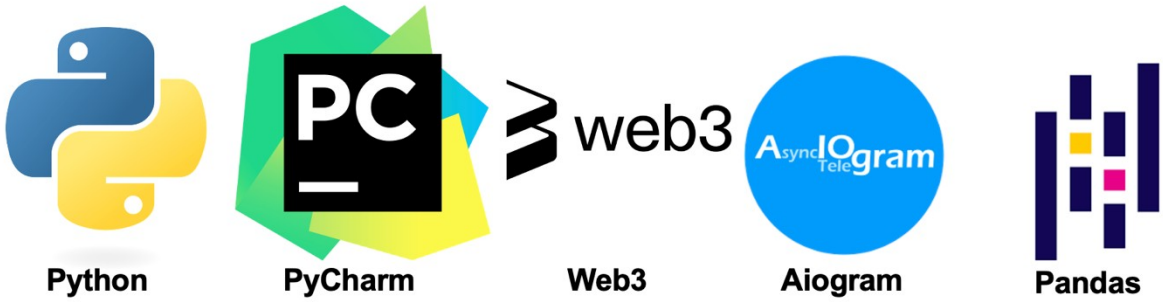


Рисунок Г.9 – Використані технології

Розробка методу асинхронної черги з rate-limiting на рівні HTTP-методів

1. Ініціалізація. На початку для кожного методу створюється власна порожня PriorityQueue і встановлюється sleep_time = 1 / фактичний_ліміт (секунд між запитами). Фактичний ліміт визначається за формулою:
фактичний_ліміт = min(rate_limit_for_method × кількість_проксі, max_requests).
2. Додавання запиту в чергу. Перед виконанням запит вкладається в чергу відповідного методу у вигляді пари (priority, Request).
3. Очікування часу. Обчислення diff за формулою:

$$\text{diff} = \text{час\_зараз} - \text{час\_останнього\_відправленого\_запиту}.$$

Якщо diff < sleep_time, чекаємо ще sleep_time − diff секунд (або дробово спимо кілька разів по sleep_time/10, поки умова не виконається).
4. Вибір проксі. Якщо проксі завантажені, з черги ProxyQueue дістається найменш використовуваний проксі.
5. Виконання HTTP-запиту. Через асинхронну сесію надсилається запит із заданим методом, URL, параметрами та обраним проксі.
6. Оновлення стану черги. Після відправки та перед обробкою відповіді видаляємо запит з черги та оновлюємо її.
7. Обробка помилок. Якщо сталося Timeout чи ConnectionError → рекурсивно повторюємо кроки 2–7 (той самий об'єкт Request). Якщо відповідь має статус 429 або 403 → логимо, робимо паузу й повторюємо те саме. Якщо 401 → виконуємо логін (login()), оновлюємо cookies і знову запускаємо повтор.
8. Повернення. Після успішного отримання відповіді (код ≠ 429/403/401, немає таймаутів) повертаємо об'єкт Response користувачу.

Рисунок Г.10 – Розробка методу асинхронної черги з rate-limiting на рівні HTTP-методів

Розробка методу асинхронної черги з rate-limiting на рівні HTTP-методів

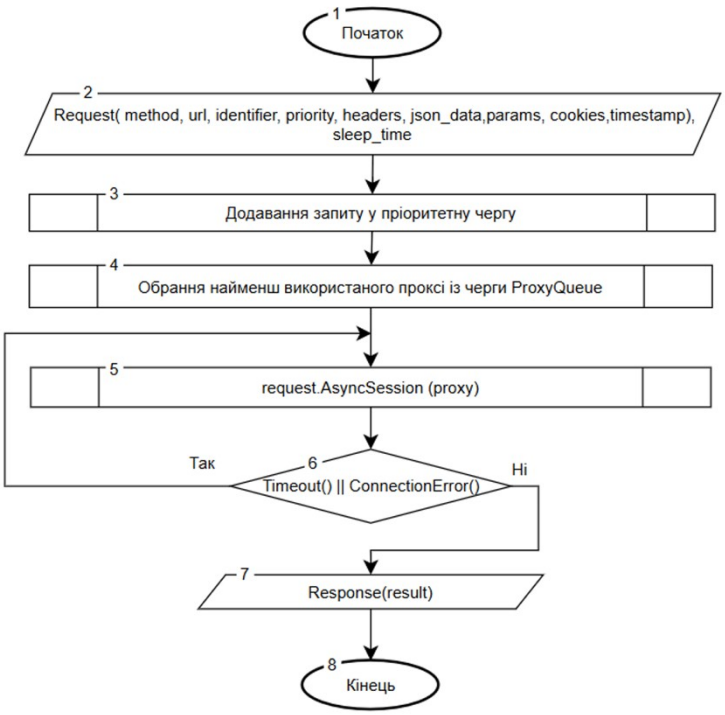


Рисунок Г.11 – Розробка методу асинхронної черги з rate-limiting на рівні HTTP-методів

Розробка розумного добору пріоритетної плати за газ при прийнятті біду

1. Обчислення базового прибутку

$$base\_profit = sale\_price - min\_price$$

Якщо base_profit ≤ 0, значить операція невигідна і виконання методу зупиняється.

2. Динамічний випадок (словник). Перебираємо словник max_priority_fee у порядку спадання порогів прибутку. Якщо жоден fee_value не забезпечує необхідний поріг чистого прибутку — відмовитися від угоди.

3. Накопичення повної пропозиції. Перетворюємо обраний maxPriorityFeePerGas в одиниці Wei та додаємо до maxFeePerGas.

4. Подальший ланцюжок. Використовуємо цей gas_price із новими значеннями в транзакції /v1/bids/accept. Далі підписуємо й надсилаємо approval- та accept-транзакції (за аналогією з алгоритмом лістингу).

Рисунок Г.12 – Розробка розумного добору пріоритетної плати за газ при прийнятті біду

Розробка розумного добору пріоритетної плати за газ при прийнятті біду

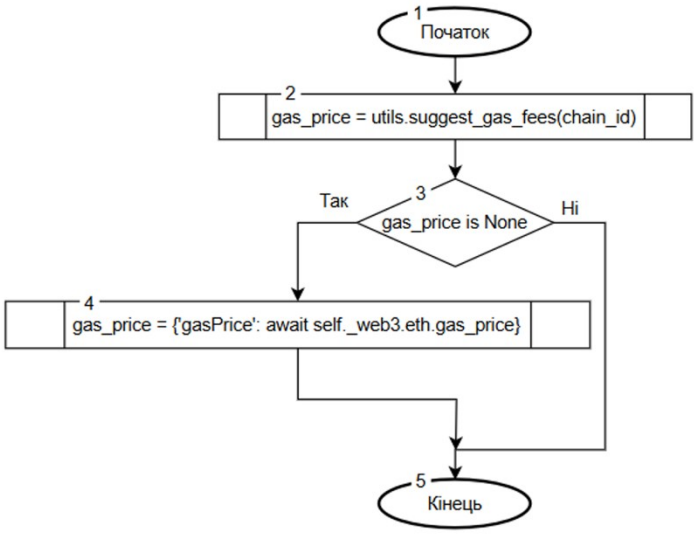


Рисунок Г.13 – Розробка розумного добору пріоритетної плати за газ при прийнятті біду

Розробка моделі системи

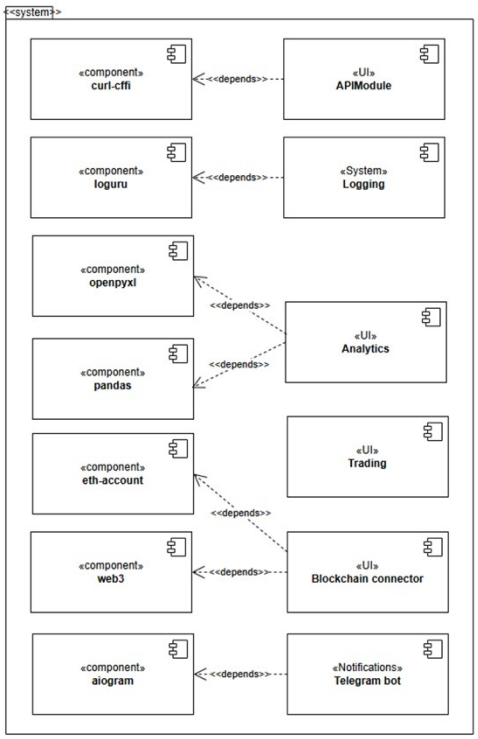


Рисунок Г.14 – Розробка моделі системи

Тестування системи

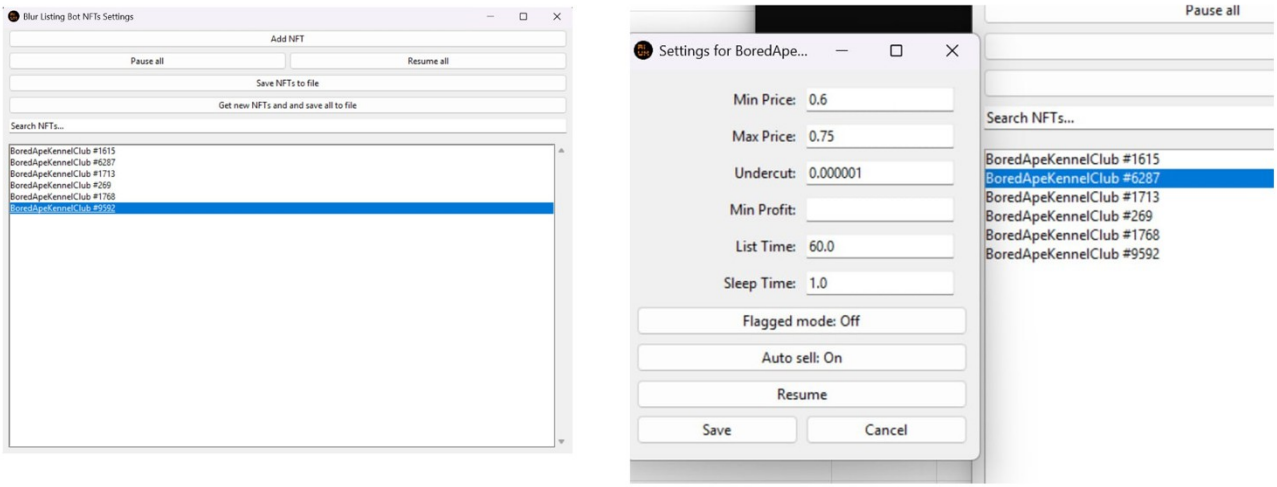


Рисунок Г.15 – Тестування системи

Тестування системи

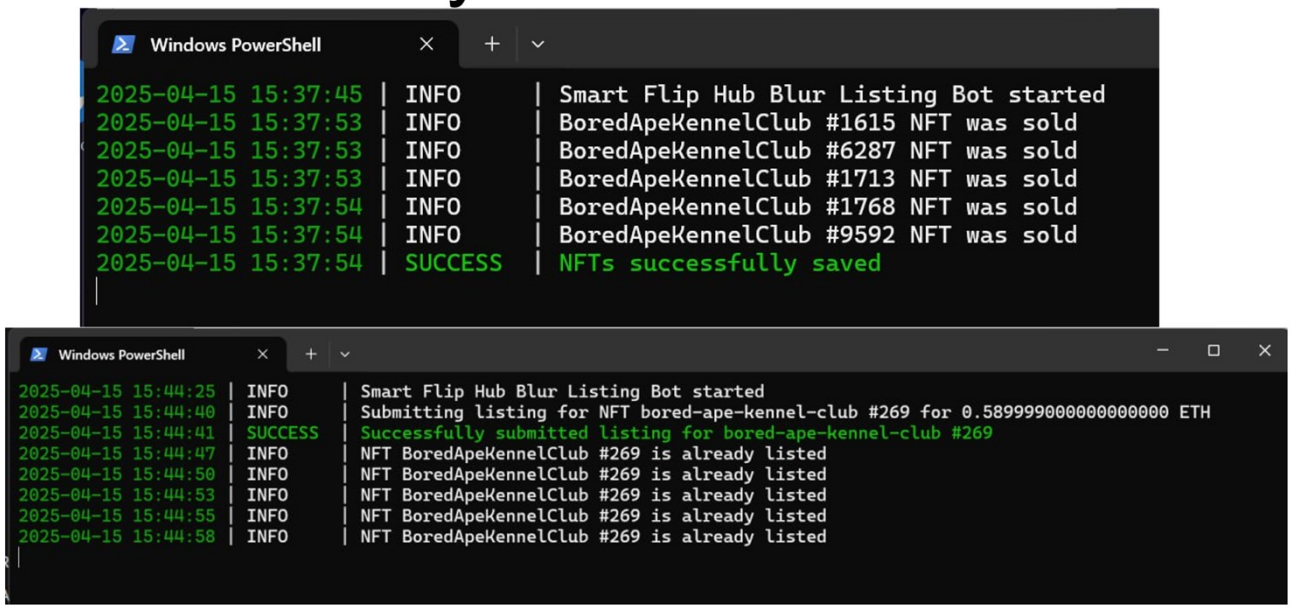


Рисунок Г.16 – Тестування системи

Тестування системи

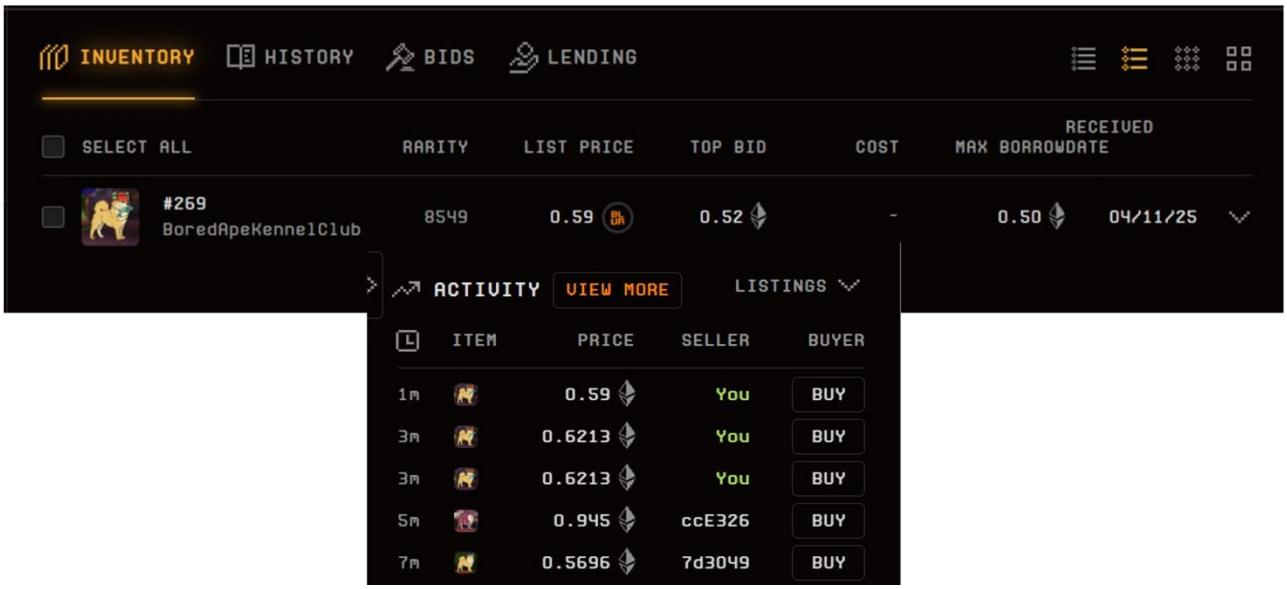


Рисунок Г.17 – Тестування системи

Тестування системи

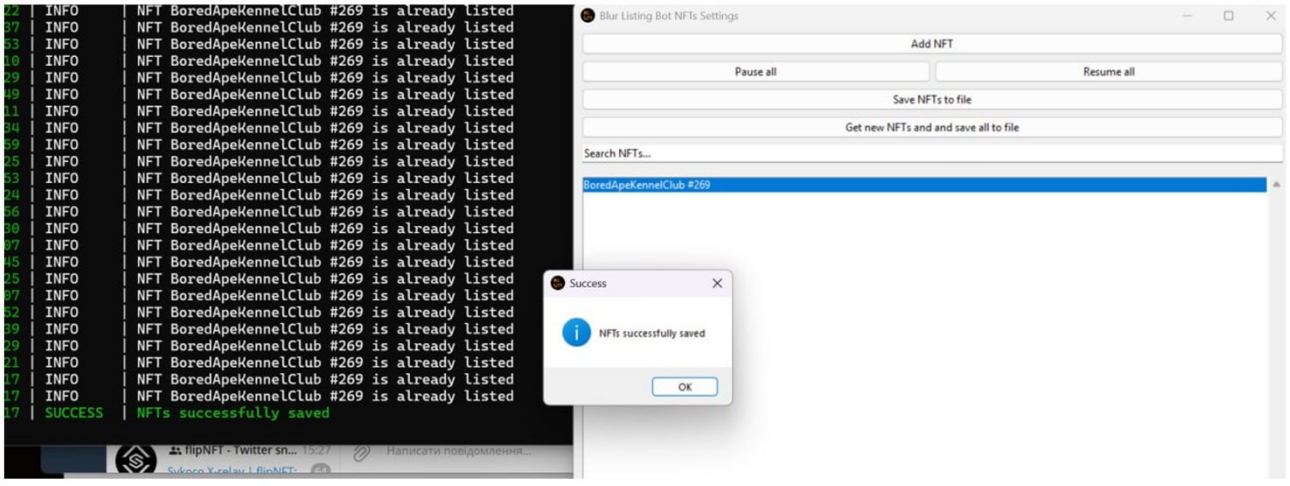


Рисунок Г.18 – Тестування системи

Висновки

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено методи та програмні засоби автоматизованої системи моніторингу та продажу NFT на маркетплейсах OpenSea і Blur. Реалізовано отримання даних про лістинги, прийняття найвигідніших пропозицій, використання актуальної ціни газу та збереження конфігурації NFT. Проведено аналіз стану розробки застосунків для NFT-торгівлі, порівняння аналогів і вибір методу на основі готових API. Розроблено модель системи, алгоритми її роботи, метод асинхронної черги з rate-limiting та метод розумного добору плати за газ. Обґрунтовано вибір Python і PyCharm як основних технологій розробки. Проведено тестування системи, що підтвердило її працездатність і відповідність поставленим завданням.

Рисунок Г.19 – Висновки

Апробація результатів роботи і публікації

Апробація результатів роботи. Описані у бакалаврській кваліфікаційній роботі положення доповідались на конференції «LIV Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2025)».
Публікації. Войтко В.В., Черноволик Г.О., Гаврилюк О.В., Барчук Н.Є., Коберник М.В. Розробка засобів автоматизованої системи моніторингу, купівлі та продажу NFT на Маркетплейсах OPENSEA та BLUR / Матеріали LIV Всеукраїнської науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2025) : збірник доповідей [Електронний ресурс]. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24402/20094

Рисунок Г.20 – Апробація результатів роботи і публікації

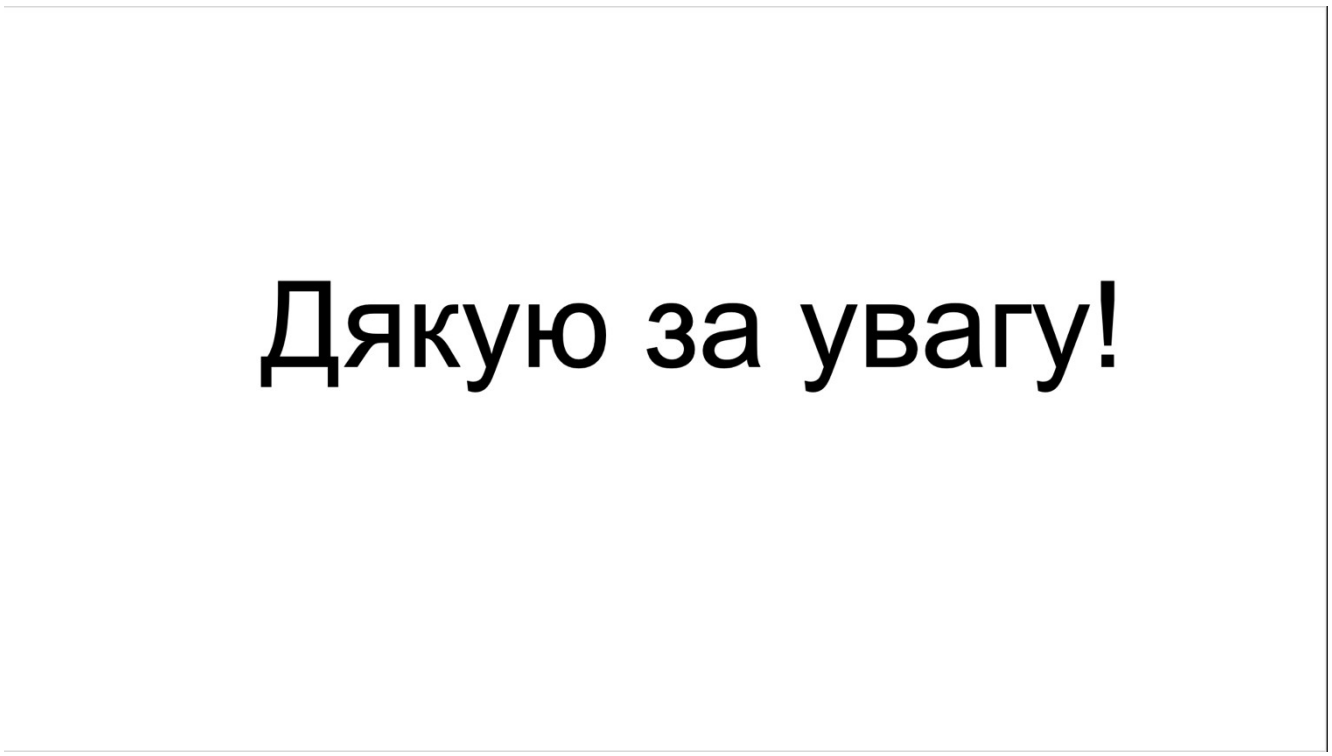


Рисунок Г.21 – Фінальний слайд