Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

Бакалаврська кваліфікаційна робота на тему:

«СТВОРЕННЯ СИСТЕМИ АВТОМАТИЗОВАНОЇ ТОРГІВЛІ КРИПТОВАЛЮТАМИ ЗА ДОПОМОГОЮ BINANCE API»

Виконав: студент IV курсу, групи 1ІСТ-21б
спеціальності 126 – Інформаційні системи та технології
(шифр і назва спеціальності)
Моїк І.І.
(прізвище та ініціали)
Керівник: к.т.н, доцент кафедри АІІТ
Кабачій В.В.
(прізвище та ініціали)
«11» 06 2025 р.
Рецензент: к.т.н., доцент кафедри
Ковалюк О.О.
(прізвище та ініціали)
«11» 06 2025 р.

Допущено до захисту
зав. кафедри АІІТ
«12» 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет

Факультет інтелектуальних інформаційних технологій та автоматизації

Кафедра автоматизації та інтелектуальних інформаційних технологій

Рівень вищої освіти перший (бакалаврський)

Галузь знань 12 – Інформаційні технології

(шифр і назва)

Спеціальність 126 – Інформаційні системи та технології

(шифр і назва спеціальності)

Освітньо-професійна програма Інтелектуальні інформаційні системи

(назва освітньо-професійної програми)

ЗАТВЕРДЖУЮ
завідувач кафедри АІІТ
д.т.н., проф. Бісікало О. В.

12. 06. 2025 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Моїку Ігорю Івановичу

(прізвище, ім'я, по батькові)

1. Тема роботи: «Створення системи автоматизованої торгівлі криптовалютами за допомогою Binance Api»

Керівник роботи Кабачій Владислав Володимирович к.т.н., доцент, доцент кафедри АІІТ

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «20» березня 2025 року № 97

2. Термін подання студентом роботи 10.06.2025 р.

3. Вихідні дані до роботи:

Розроблене програмне забезпечення є автоматизованою системою для торгівлі криптовалютами на ф'ючерсному ринку Binance. Мова програмування – Python з використанням асинхронної бібліотеки asyncio та офіційного клієнта python-binance.

4. Зміст текстової частини

Аналіз існуючих програмних продуктів по даній тематиці предметних переваг та недоліків. Розробка моделі програмного засобу, для реалізації поставленої задачі. Розробка програмного застосунку для підготовки до мультипредметного тестування з адаптивною перевіркою знань та рекомендаціями.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень)

Алгорим роботи торгового бота, UML-діаграма послідовностей для Rest-запиту на виставлення захисних ордерів, UML діаграма алгоритму моніторингу ринку та відкриття угод, UML діаграма обробки сигналу та виставлення торгових ордерів, UML-схема класів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата завдання видав	Підпис, дата виконав прийн
1-4	Кабачій В. В., доц. каф. АІІТ		

7. Дата видачі завдання 20.03.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Прим
		початок	закінчення	
1	Аналіз предметної області	*02.04.2025*	*22.04.2025*	*викон*
2	Порівняльний аналіз проблематики	*23.04.2025*	*30.04.2025*	*викон*
3	Вибір оптимальних інформаційних технологій	*03.05.2025*	*12.05.2025*	*викон*
4	Розроблення інформаційної системи автоматизації роботи користувачів	*14.05.2025*	*21.05.2025*	*викон*
5	Оформлення матеріалів до захисту БКР	*22.05.2025*	*29.05.2025*	*викон*

Студент _____ Моїк І.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Кабачій В.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 76 сторінок формату А4 включаючи додатки, на яких є 19 рисунків, список використаних джерел містить 25 найменувань.

Робота присвячена розробці програмного забезпечення для автоматизованої торгівлі криптовалютами з використанням API біржі Binance. Основною метою дослідження є створення ефективної та зручної системи, яка дозволить користувачам здійснювати торгові операції в автоматизованому режимі. У ході реалізації було проаналізовано сучасні підходи до алгоритмічної торгівлі, досліджено можливості Binance API, а також проведено порівняльний аналіз існуючих рішень у цій сфері.

Система реалізована за допомогою мови програмування Python, що забезпечує зручну інтеграцію з API. У програмне забезпечення закладено модулі для отримання ринкових даних у реальному часі, виконання торгових ордерів, контролю ризиків (через встановлення стоп-лоссів і тейк-профітів), а також моніторинг торгових операцій.

Результатом роботи є функціональний прототип торгового бота, який може адаптуватися до ринкових змін і виконувати базові торгові операції. Запропонована система може бути корисною як для початківців, так і для досвідчених трейдерів, оскільки дозволяє мінімізувати вплив емоцій, економити час та забезпечувати безперервний процес торгівлі.

Ключові слова: автоматизація, криптовалюта, Binance API, торговий бот, алгоритмічна торгівля, Python.

ABSTRACT

The Bachelor's degree qualification work consists of 76 pages of A4 format, including appendices, which contain 19 figures, and the list of references consists of 25 titles.

The paper is devoted to the development of software for automated cryptocurrency trading using the Binance exchange API. The main goal of the study is to create an efficient, reliable and easy-to-use system that will allow users to conduct trading operations in an automated mode. In the course of implementation, we analyzed modern approaches to algorithmic trading, studied the capabilities of the Binance API, and conducted a comparative analysis of existing solutions in this area.

The system is implemented using the Python programming language, which provides convenient integration with the API. The software includes modules for obtaining real-time market data, executing trading orders, controlling risks (by setting stop losses and take profits), and monitoring trading operations.

The result is a functional prototype of a trading bot that can adapt to market changes and perform basic trading operations. The proposed system can be useful for both beginners and experienced traders, as it minimizes the influence of emotions, saves time, and ensures a continuous trading process.

Keywords: automation, cryptocurrency, Binance API, trading bot, algorithmic trading, Python.

ЗМІСТ

ВСТУП

Актуальність роботи. У сучасних умовах цифровізації фінансових ринків та стрімкого розвитку блокчейн-технологій, криптовалюти посідають все більш вагоме місце в глобальній економіці. Щодня мільйони користувачів по всьому світу здійснюють операції з цифровими активами, прагнучи отримати прибуток за рахунок коливань їхньої вартості. При цьому ручна торгівля на криптовалютних біржах є надзвичайно динамічною та емоційно навантаженою, вимагає постійної уваги, швидких рішень і знань технічного аналізу. Саме тому зростає попит на інструменти, що дозволяють автоматизувати торговельні стратегії та мінімізувати людський фактор.

Одним з лідерів ринку криптовалют є біржа Binance, яка пропонує зручний та функціональний API для створення програмних рішень, що можуть здійснювати операції купівлі та продажу активів без прямої участі трейдера. Інтеграція з Binance API дозволяє будувати системи, здатні в режимі реального часу аналізувати ринок, відкривати або закривати позиції, виставляти торгові ордери, а також контролювати ризики.

Автоматизовані торгові системи (або трейдинг-боти) здатні істотно підвищити ефективність торгівлі, зменшити кількість помилок і забезпечити виконання стратегій, які було б складно реалізувати вручну. Проте розробка подібних систем вимагає комплексного підходу – від вибору архітектури до реалізації логіки прийняття рішень і безпечної взаємодії з API.

Метою роботи є реалізація програмного забезпечення для автоматизованої торгівлі криптовалютами. Система повинна мати змогу збирати ринкові дані, виконувати торгівельні операції відповідно до заданої стратегії, а також реагувати на зміни ринку в режимі реального часу.

Для досягнення поставленої мети необхідно розв'язати такі задачі:

1) Проаналізувати предметну область та існуючі рішення автоматизованої торгівлі.
2) Ознайомитися з можливостями Binance API.
3) Спроєктувати архітектуру системи автоматизованої торгівлі.
4) Реалізувати модуль збору та обробки ринкових даних.
5) Розробити модуль прийняття торгових рішень згідно з обраною стратегією.

Об'єктом дослідження є процес автоматизованої торгівлі на криптовалютному ринку.

Предметом дослідження виступає програмна система, яка здійснює торгівельні операції за заданими параметрами з використанням Binance API.

Методи дослідження. Аналіз відкритих API, моделювання програмної архітектури, застосування алгоритмічних стратегій, тестування у режимі емуляції ринку, а також використання методів обробки реального часу.

Практичний результат роботи полягає у створенні функціонального прототипу трейдинг-бота для роботи з Binance API, який можливо застосовувати як для навчальних цілей, так і як основу для комерційного використання. Система дозволяє гнучко адаптувати торгові параметри, відстежувати ефективність стратегії та розширювати функціональність у майбутньому.

Апробація результатів дослідження. Основні положення бакалаврської кваліфікаційної роботи були представлені на студентській науково-практичній конференції «Інтелектуальні інформаційні технології та автоматизація – 2025» [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ АВТОМАТИЗАЦІЇ ТОРГІВЛІ

1.1 Актуальність автоматизації торгівлі криптовалютами

У XXI столітті цифрові активи, зокрема криптовалюти, стали не лише інструментом збереження вартості, а й повноцінним сегментом глобального фінансового ринку. У міру зростання популярності криптовалютного трейдингу підвищується і попит на ефективні, швидкі та стабільні засоби автоматизації процесу торгівлі. Особливої актуальності набувають торгові боти – програмні рішення, які дозволяють здійснювати операції на біржі без прямої участі трейдера.

Binance – одна з найбільших криптовалютних бірж у світі за обсягами торгів та кількістю активних користувачів. Вона пропонує потужний API (Application Programming Interface), який дозволяє отримувати ринкові дані в режимі реального часу, відкривати і закривати позиції, встановлювати торгові ордери, а також відслідковувати зміни у стані акаунту. Цей API став основою для створення тисяч торгових систем, що працюють автономно, оптимізуючи стратегії й мінімізуючи ризики, пов’язані з людським фактором [2].

Актуальність автоматизованої торгівлі зростає ще й у зв’язку з високою волатильністю ринку криптовалют. Зміни цін можуть відбуватись щосекунди, і трейдери часто не встигають реагувати на ринкові сигнали. У таких умовах автоматизовані рішення здатні виконувати сотні операцій на хвилину, проводити глибокий технічний аналіз, та миттєво приймати рішення на основі заданих умов. Це дозволяє не лише підвищити ефективність торгівлі, але й зменшити ймовірність втрат через затримки або емоційні рішення трейдера [3].

Крім технічних переваг, автоматизована торгівля відкриває можливості для впровадження складних стратегій, які було б надто складно реалізувати вручну. Наприклад, бот може відстежувати обсяги, індикатори, графіки, ончейн-метрики або

новини – і приймати рішення відповідно до заздалегідь визначених правил. Крім того, API Binance дозволяє отримати доступ до ф'ючерсного ринку, що дає змогу реалізовувати стратегії з використанням кредитного плеча, хеджуванням або одночасною торгівлею в обидві сторони.

Таким чином, автоматизація трейдингу за допомогою Binance API є сучасною відповіддю на виклики швидкоплинного ринку цифрових активів. Її актуальність полягає у забезпеченні стабільної, швидкої та надійної взаємодії з біржею, що робить її незамінною частиною арсеналу трейдера в умовах високої конкуренції, глобалізації ринку та розвитку Web 3.0.

1.2 Аналіз існуючих систем та рішень на основі API

У сфері криптовалютної торгівлі за останнє десятиліття з’явилася велика кількість автоматизованих рішень, які працюють на основі біржових API. Ці системи дозволяють трейдерам реалізовувати стратегії без постійної ручної участі, оптимізуючи прибутковість і мінімізуючи ризики. Розглянемо кілька найпоширеніших типів таких систем та приклади реалізації.

3Commas є одним з найвідоміших хмарних сервісів для автоматизації торгівлі. Платформа підтримує підключення до багатьох популярних бірж через API, зокрема Binance, KuCoin, Coinbase та інші. Користувачі можуть створювати боти, які виконують стратегії DCA, GRID, а також смарт-ордери зі встановленим тейк-профітом і стоп-лоссом. Також доступна функція копіювання стратегій інших трейдерів (рис 1.1).

Переваги: простота використання, готові шаблони, інтеграція з TradingView.

Недоліки: платна підписка, обмежена гнучкість налаштувань для просунутих користувачів.

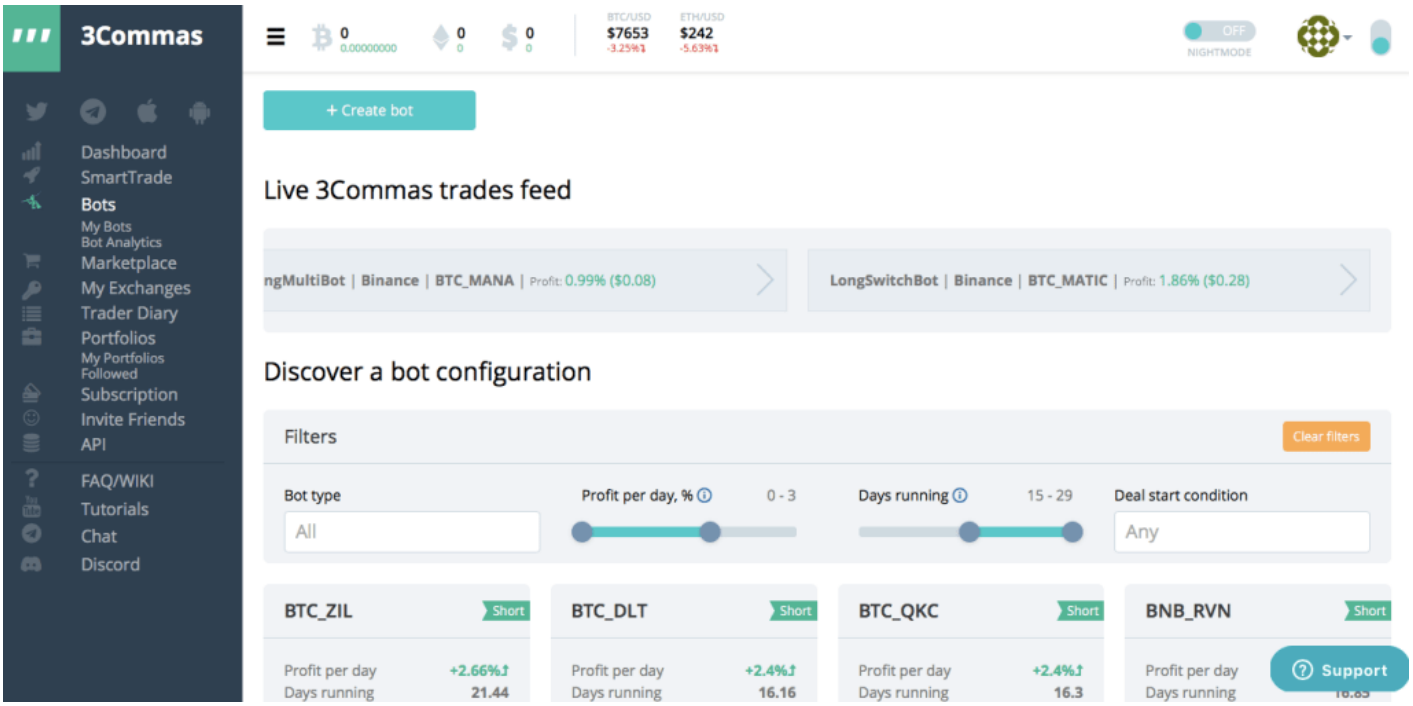


Рисунок 1.1 – Сторінка створення торгового боту

Cryptohopper - ця платформа орієнтована на середній та професійний рівень трейдерів. Вона дозволяє створювати складні торгові алгоритми, використовуючи понад 100 індикаторів технічного аналізу. Cryptohopper також підтримує API-підключення до провідних бірж і має маркетплейс стратегій (рис. 1.2).

Переваги: глибокі налаштування стратегій, симуляція та бектестинг.

Недоліки: складний інтерфейс, деякі функції доступні лише у преміум-планах.

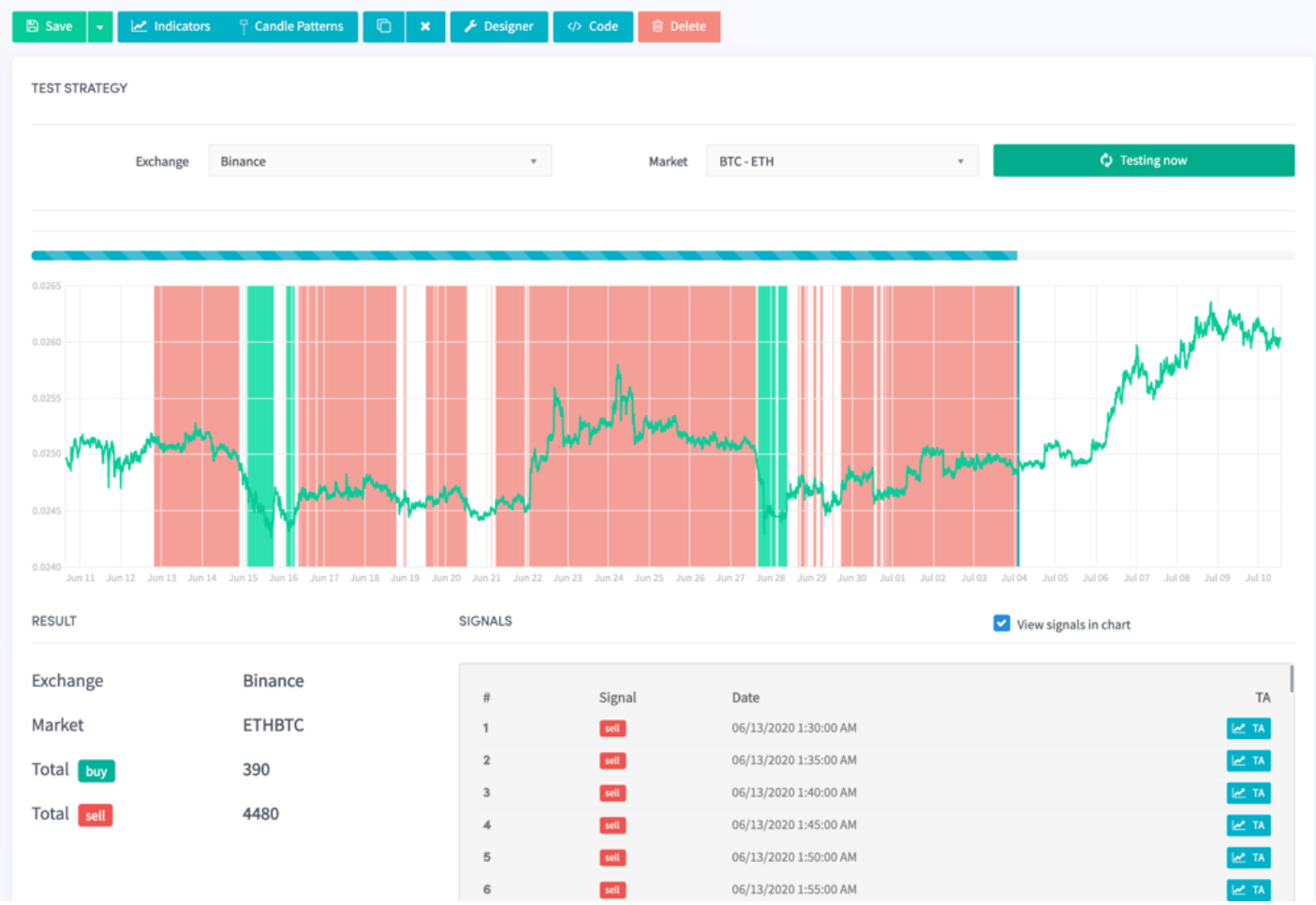


Рисунок 1.2 – Приклад тестування стратегії

HaasOnline – це одна з найстаріших платформ для алгоритмічної торгівлі. Вона орієнтована на досвідчених трейдерів та розробників. Система підтримує складні скрипти, написані мовою HaasScript, а також має потужну систему симуляції та тестування (рис. 1.3).

Переваги: повна кастомізація логіки торгівлі, висока продуктивність, підтримка API WebSocket.

Недоліки: складність освоєння, вимагає встановлення на власний сервер, висока вартість ліцензії.

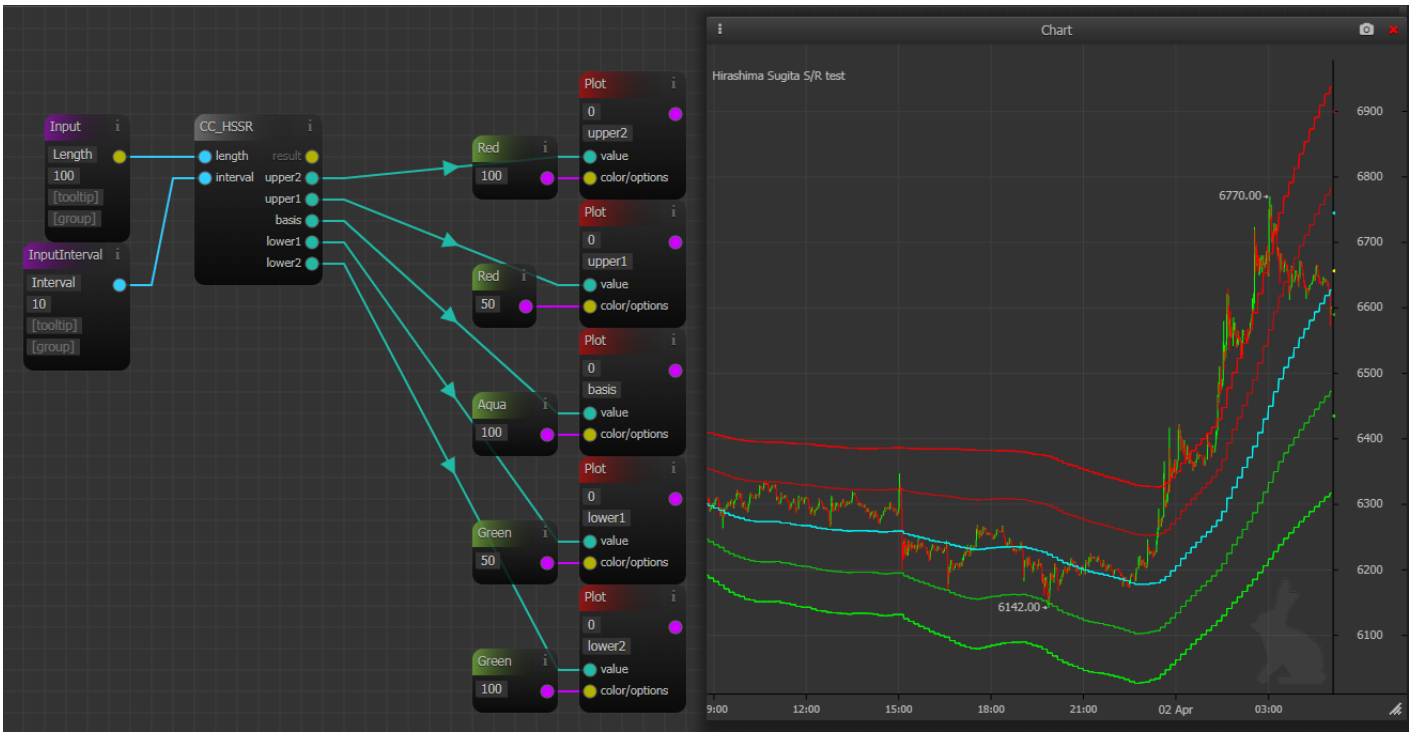


Рисунок 1.3 – Приклад розробки бота без написання коду

Pionex є унікальним поєднанням біржі та платформи для автоматизації. Платформа вбудовує 16 безкоштовних торгових ботів прямо у свою біржу, серед яких – DCA, GRID, арбітражний бот тощо (рис. 1.4). Система працює з API, проте має обмеження для сторонніх розробників.

Переваги: безкоштовні боти, проста інтеграція, мобільний додаток.

Недоліки: неможливість кастомного програмування, залежність від самої біржі.

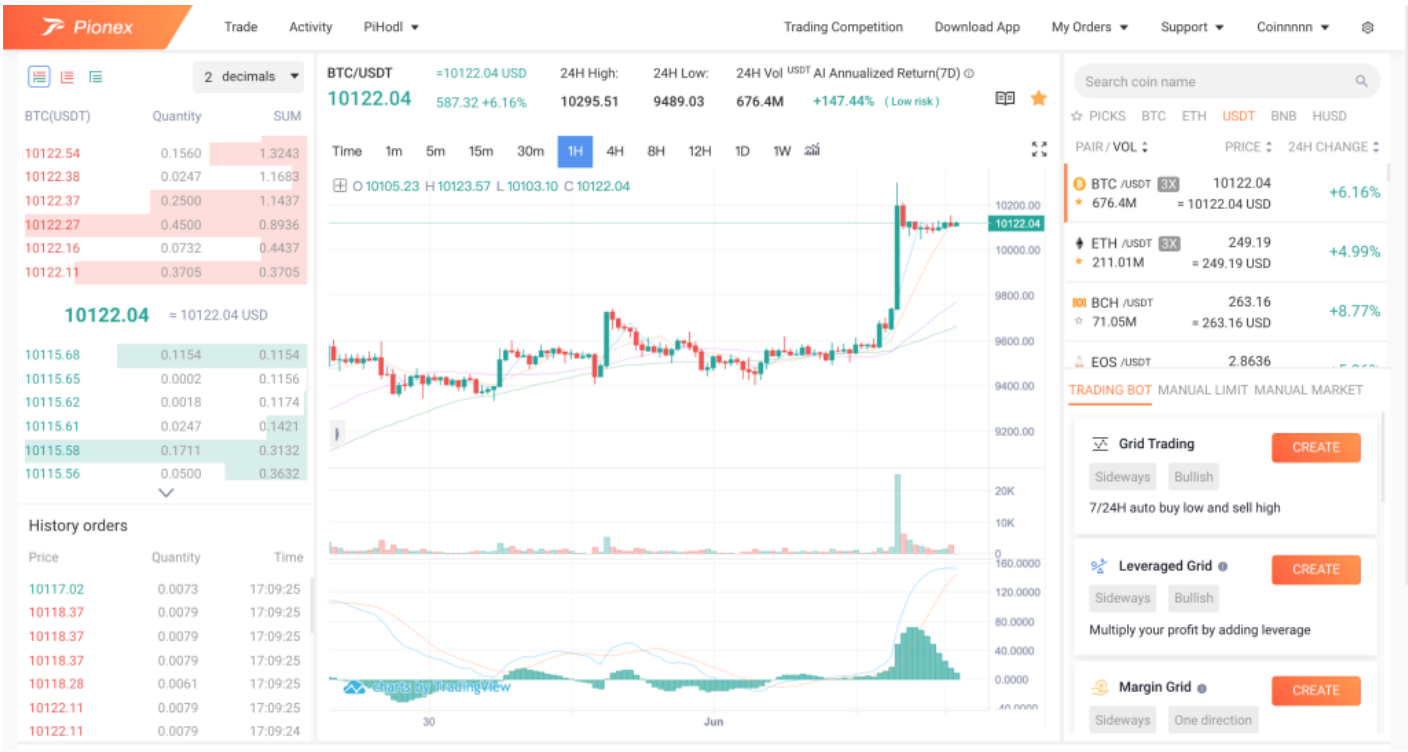


Рисунок 1.4 – Сторінка для створення торгового боту

Усі розглянуті рішення мають свої переваги та обмеження. Універсальні сервіси (3Commas, Pionex) підійдуть початківцям, які шукають простоту і швидкий старт. Інструменти на зразок HaasOnline чи Cryptohopper забезпечують розширені можливості, але мають вищий поріг входу ніж попередні. Водночас розробка власного бота на основі офіційного Binance API є найгнучкішим шляхом, оскільки дозволяє реалізувати унікальні стратегії, адаптовані до будь-якого ринку та стилю торгівлі.

Це й обґрунтовує актуальність створення власної системи, яка поєднуватиме переваги гнучкої архітектури, швидкодії (завдяки WebSocket), інтеграції з аналітичними сервісами та можливістю розширення функціоналу в майбутньому.

1.3 Постановка задачі дослідження

Сучасний ринок криптовалют характеризується високою волатильністю, динамічною зміною обсягів торгів і значним впливом великих гравців на миттєві рухи ціни. Для ефективної роботи в таких умовах трейдерам необхідно реагувати швидко та обґрунтовано, що потребує постійного моніторингу ринку в режимі реального часу. Саме тому автоматизація торгівлі за допомогою торгових ботів, які працюють через API біржі, стала невід'ємною частиною сучасної криптоторгівлі.

Однак більшість готових платформ не враховують специфіку короткочасних різких змін ціни (наприклад, сплесків у межах кількох секунд), а їхня логіка часто обмежується шаблонними підходами, як-от DCA, сіткові стратегії або технічні індикатори з великими періодами.

Для досягнення кращих результатів необхідна система, здатна:

- швидко аналізувати обсяг та зміну ціни з використанням WebSocket-даних у реальному часі, виявляти нетипову активність, наприклад, різке зростання обсягів у короткий проміжок часу
- відкривати ринкові позиції одразу після підтвердження патерну
- автоматично встановлювати рівні стоп-лоссу і тейк-профіту, виходячи з реальної ціни входу.

У цьому контексті постає необхідність у створенні кастомного торгового бота, орієнтованого на високу швидкодію, точне реагування на імпульсні рухи та мінімальну затримку між виявленням сигналу і відкриттям позиції.

Таким чином, мета дослідження полягає у розробці ефективної торгової системи для криптовалютного ф'ючерсного ринку, що:

- використовує офіційне API біржі Binance для збору ринкових даних і керування ордерами;

- базується на високочастотному аналізі через WebSocket з мінімальним пінгом;
- реалізує автоматичне виставлення стоп-лоссу і тейк-профіту після отримання фактичної ціни входу;
- забезпечує можливість масштабування, моніторингу результатів та подальшого удосконалення стратегії.

Щоб досягти поставленої мети, у межах дослідження необхідно вирішити низку взаємопов’язаних завдань. Передусім потрібно вивчити особливості роботи з API ф’ючерсного ринку Binance, зокрема процес авторизації, структуру торгових ордерів, можливості використання WebSocket для отримання даних у реальному часі, а також врахувати обмеження на частоту запитів.

Далі слід спроєктувати архітектуру торгового бота з чітким поділом на функціональні модулі: аналіз вхідних даних, прийняття рішень на основі виявлених патернів, виконання торгових операцій, а також облік усіх дій і логування для подальшого аналізу. Одним з ключових завдань є реалізація механізму, який дозволить виявляти різкі зміни ціни та обсягу на ринку з максимально короткою затримкою у відповідь, що не перевищує однієї секунди.

Крім того, важливо реалізувати точне визначення фактичної ціни входу в позицію після виконання ринкового ордера, оскільки від цього залежить правильне встановлення захисних ордерів. На основі отриманої ціни необхідно забезпечити автоматичне виставлення динамічних стоп-лоссів та тейк-профітів, які адаптуються до умов ринку.

Завершальним етапом стане перевірка працездатності й ефективності побудованої стратегії – як на історичних, так і на симульованих даних, а також у режимі тестової торгівлі або за участі невеликого капіталу в реальному середовищі.

У результаті дослідження має бути створена програмна система, здатна в реальному часі адаптуватися до ринкових умов та оперативно приймати торгові рішення, що може стати основою для подальших вдосконалень.

1.4 Висновки до розділу

В даному розділі було проведено огляд сучасного стану автоматизованої торгівлі та особливостей використання біржових API, що дозволило визначити ключові технічні аспекти, які слід враховувати при реалізації системи. Також проаналізовано декілька існуючих рішень на основі API, що продемонструвало широкий спектр підходів до побудови ботів та виявило їхні сильні й слабкі сторони, зокрема щодо швидкодії, гнучкості та захисту ризиків.

На основі проведеного аналізу сформульовано мету дослідження та визначено перелік завдань, які охоплюють всі основні етапи розробки системи: від інтеграції з Binance API до тестування торгової логіки в умовах, наближених до реальних. Особлива увага приділена таким аспектам, як швидке реагування на ринкові зміни, точне визначення фактичної ціни входу в позицію та динамічне налаштування торгових ордерів.

2 ПРОЕКТУВАННЯ СТРУКТУРИ ТА КОМПОНЕНТІВ СИСТЕМИ

2.1 Обґрунтування архітектурного підходу до реалізації торгового бота

Розробка торгового бота для криптовалютного ринку вимагає чіткого архітектурного підходу, який забезпечить стабільну роботу в режимі реального часу, низьку затримку при обробці подій, адаптивність до змін ринку та легкість у масштабуванні. Криптовалютні біржі працюють цілодобово, без перерв і вихідних, а отже, програмна система має бути максимально автономною, відмовостійкою та здатною швидко реагувати на динаміку ринку.

Одним із найважливіших факторів при проєктуванні архітектури є вибір моделі побудови компонентів. В умовах складної логіки торгівлі й великої кількості зовнішніх викликів (зокрема, через Binance API), доцільним є застосування модульної архітектури. Цей підхід дозволяє логічно розділити систему на окремі функціональні частини, кожна з яких відповідає за власну частину процесу – наприклад, збір ринкових даних, аналіз сигналів, ухвалення торгових рішень, управління ордерами, моніторинг, логування та безпеку.

Модульність – одна з основних практик у сучасній розробці програмного забезпечення, зокрема в складних системах зі змінними залежностями. За словами С. Макконнелла у своїй праці Code Complete, модульна архітектура підвищує читабельність коду, знижує складність тестування та прискорює розгортання нових версій системи без ризику порушити її роботу в цілому [4].

У сфері алгоритмічної торгівлі така архітектура особливо ефективна, оскільки:

- дозволяє оперативно змінювати торгову стратегію без переписування всієї системи;
- полегшує оновлення або заміну API-клієнта при змінах у документації Binance;

- спрощує реалізацію безпечної обробки помилок та надзвичайних ситуацій (наприклад, втрата WebSocket-з’єднання);
- дає змогу легко масштабувати функціонал, додаючи нові стратегії, моніторингові модулі чи інтерфейси.

Компоненти архітектури

У рамках даної системи архітектура умовно складається з шести ключових модулів:

1) Модуль взаємодії з Binance API – відповідає за REST-запити та WebSocket-потоки. Забезпечує авторизацію, підпис запитів, відправку ордерів і отримання даних у реальному часі. Також реалізує моніторинг обмежень на запити (rate limits), що є критично важливою умовою при роботі з Binance.

2) Модуль аналізу ринку – використовує WebSocket-дані для оцінки короткочасної волатильності, відстеження різких змін ціни, обсягу або дельти та формує потенційні сигнали до входу. Для обробки даних можуть використовуватись технічні індикатори або евристичні правила.

3) Модуль прийняття рішень – на основі сигналів від модуля аналізу формує конкретну дію: відкриття довгої або короткої позиції, очікування або фільтрація шуму. Модуль містить параметри ризик-менеджменту, фільтри волатильності, допустимі межі просадки тощо.

4) Модуль виконання угод – після ухвалення рішення бот надсилає ринковий ордер через REST API, фіксує фактичну ціну виконання (execution price), розраховує стоп-лосс і тейк-профіт на її основі та виставляє відповідні ордери.

5) Модуль логування та звітності – записує всі торгові дії, сигнали, помилки та події системи. Дані використовуються для відлагодження, аудиту та побудови аналітичних графіків за результатами торгівлі.

6) Модуль безпеки та стійкості – забезпечує повторні підключення до WebSocket при втраті з’єднання, обробку таймаутів і виняткових ситуацій, а також реалізує обмеження обсягу та кількості одночасних угод.

Вибір технологій і реалізацій

У розробці були використані сучасні бібліотеки та фреймворки, зокрема python-binance для REST і WebSocket-інтеграції, pandas для аналізу даних та Docker для розгортання у відокремленому середовищі. Застосування контейнеризації спрощує масштабування й оновлення коду [5].

Рекомендованим практичним підходом також є реалізація моделі обробки подій (event-driven architecture), яка дозволяє асинхронно обробляти сигнали та забезпечити мінімальну затримку між надходженням події й виконанням дії – що критично для високочастотної торгівлі.

Обґрунтування з позиції ризиків і продуктивності

Висока волатильність ринку та можливі технічні збої вимагають від архітектури не лише гнучкості, але й надійності. Система повинна:

- бути здатною до гарячого перезапуску без втрати контексту;
- відновлювати позиції після збою;
- уникати повторного виконання запитів через idempotent-логіку (унікальні клієнтські ідентифікатори ордерів);
- вести журнал дій навіть при збої бази даних.

Подібні вимоги описані у книзі Pressman R. Software Engineering: A Practitioner’s Approach, де розробка розглядається як поєднання інженерних рішень і реалій функціонального середовища [6].

2.2 Опис логічної структури системи

Автоматизована система торгівлі криптовалютами – це складна сукупність програмних компонентів, що взаємодіють між собою в реальному часі. Логічна структура системи визначає порядок руху інформації між підсистемами, принципи обробки подій, черговість виконання дій та логіку зв’язку між процесами. У рамках цієї роботи логічна структура побудована як асинхронна подієво-керована система з наскрізним циклом обробки ринкових сигналів.

Життєвий цикл торгової дії в системі автоматизованої торгівлі побудований на безперервній обробці подій – від моменту отримання ринкових даних до виконання ордера або запису інформації в лог. Робота відбувається у кілька етапів.

Спочатку система підключається до WebSocket-каналів біржі та в режимі реального часу отримує оновлення про ціну, обсяг торгів та інші параметри обраних торгових пар. Ці дані надходять регулярно – щосекунди або навіть частіше.

Після цього система виконує попередню обробку інформації: дані перевіряються, усереднюються, фільтруються від шумів і аномалій. Мета цього етапу – сформувати поточне уявлення про ситуацію на ринку.

Далі проводиться аналіз – чи відповідає ринкова ситуація умовам торгової стратегії. Оцінюються такі параметри, як ринковий імпульс, волатильність, обсяг торгів і зміни ціни. Якщо всі умови збігаються, система формує торговий сигнал.

На основі цього сигналу приймається рішення: чи варто відкривати позицію, яку саме – довгу (лонг) чи коротку (шорт), і з яким обсягом. Також враховуються обмеження ризику, наявність відкритих позицій та можливі затримки в роботі ринку.

Після ухвалення рішення система через REST API надсилає запит на відкриття ринкового ордера. Для кожного запиту створюється унікальний ідентифікатор, щоб у подальшому можна було відстежити його виконання.

На завершення система очікує підтвердження від біржі про виконання ордера. Коли позиція відкривається, фіксується фактична ціна входу, яка використовується для розрахунку рівнів стоп-лоссу та тейк-профіту.

Після відкриття позиції бот отримує фактичну ціну входу через REST-запит і одразу ж виставляє захисні ордери: STOP_MARKET для стоп-лоссу та TAKE_PROFIT_MARKET для тейк-профіту. Усі дії та відповіді від Binance логуються, а їх послідовність зображено на UML-діаграмі 2.1.

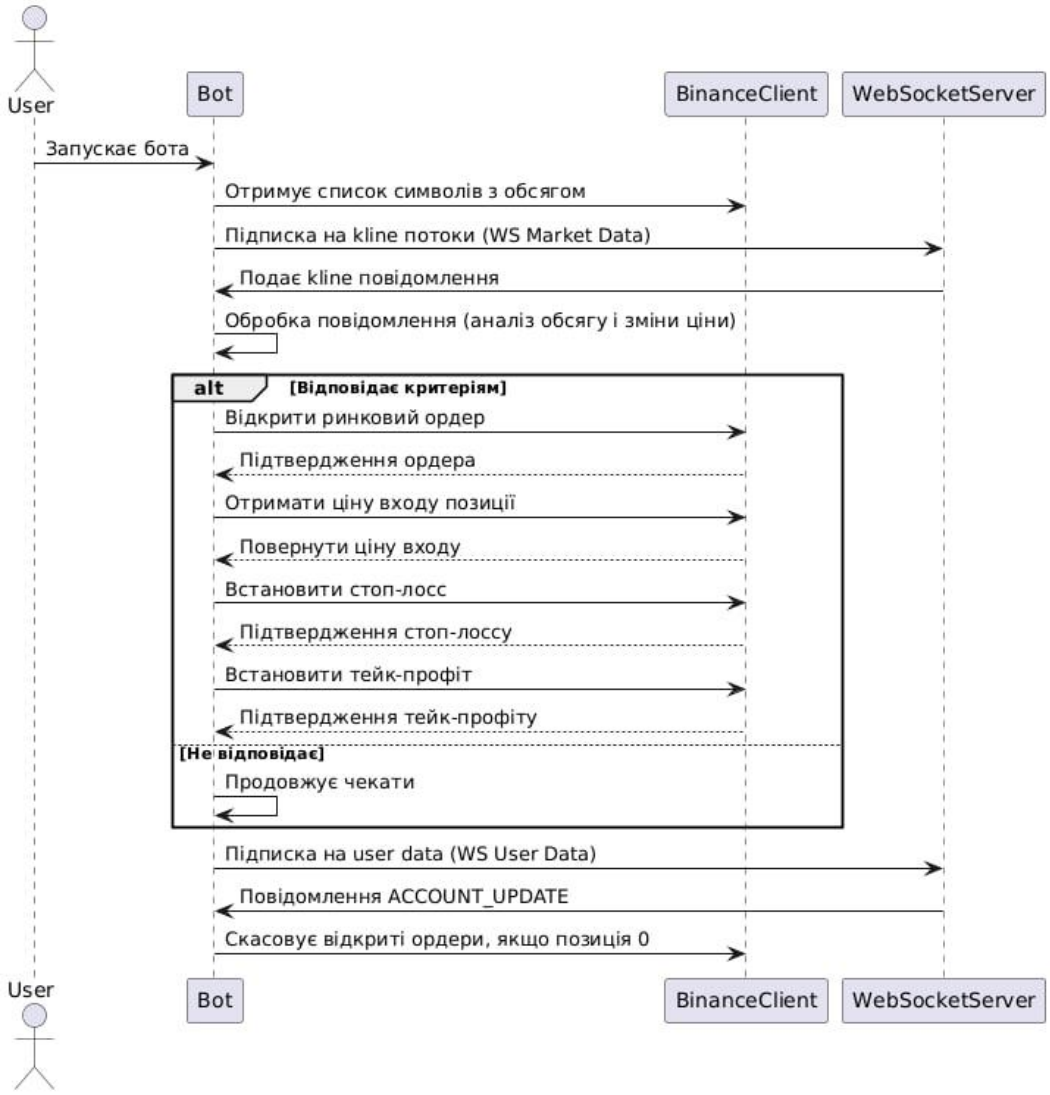


Рисунок 2.1 – UML-діаграма послідовностей для Rest-запиту на виставлення захисних ордерів

Моніторинг ринку та відкриття позицій.

Після запуску бот підключається до WebSocket і постійно отримує оновлення ринку. У разі значного зростання ціни та достатнього торгового обсягу, якщо монета ще не була оброблена, бот виконує аналіз і ініціює відкриття ф'ючерсної позиції через REST API (рис. 2.2).

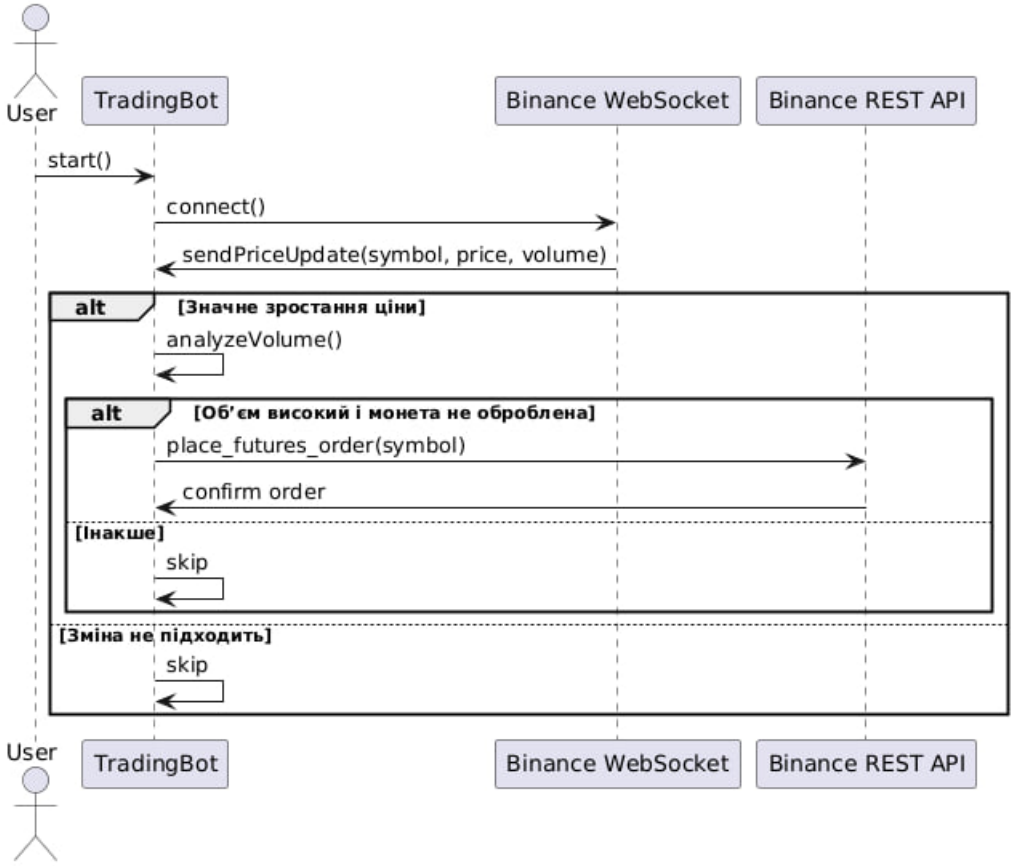


Рисунок 2.2 – UML діаграма алгоритму моніторингу ринку та відкриття угод

Формати даних і взаємодія

Усі внутрішні об’єкти системи уніфіковано за структурою:

- SignalEvent – містить дані ринку, таймстемп, пару, тип сигналу.
- TradeDecision – тип угоди, обсяг, напрям, стоп/тейк значення.
- ExecutionResult – фактична ціна входу, статус ордера, повідомлення від біржі.

Компоненти взаємодіють через черги повідомлень або внутрішній диспетчер подій, що дозволяє будувати асинхронні ланцюги обробки.

Сценарії обробки помилок і затримок

У логічну структуру системи закладені й механізми обробки виняткових ситуацій:

- тайм-аут відповіді від REST → повтор через 0.5 секунди;
- відсутність executionReport у WebSocket → перевірка вручну через REST;
- невдале виставлення SL → повтор через 1 секунду з іншим ID;
- WebSocket disconnect → автоматичне перепідключення (max 3 спроби).

Також реалізовано модуль сигналізації через лог-файл про всі критичні події (наприклад, збої, помилки API, неочікуване виконання) [8].

Процес обробки торгового сигналу.

Після запуску основного бота здійснюється підключення до WebSocket потоку для отримання ринкових даних. Компонент MarketWatcher взаємодіє з Binance API, аналізуючи ф'ючерсні тікери та передаючи сигнал на купівлю. Основний модуль передає цей сигнал до OrderManager, який виконує серію запитів до Binance API для створення ринкового ордера, отримання інформації про позицію та виставлення захисних ордерів (рис. 2.3).

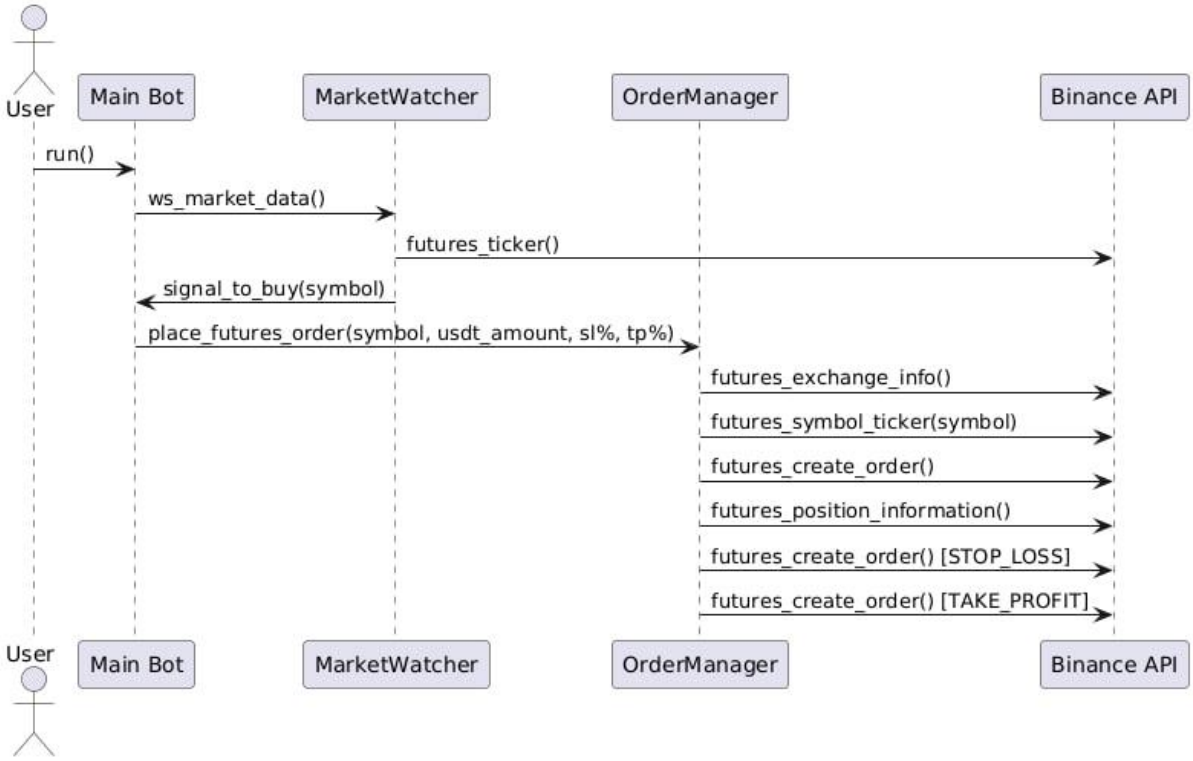


Рисунок 2.3 – UML діаграма обробки сигналу та виставлення торгових ордерів

Архітектура програмного забезпечення побудована на основі чотирьох ключових компонентів: класу для роботи з Binance API, який відповідає за отримання торгових пар і розміщення ф'ючерсних ордерів; модуля для отримання ринкових даних через WebSocket, що здійснює аналіз змін цін і обсягів у режимі реального часу; модуля для обробки подій користувача, який слідкує за станом позицій і керує відкритими ордерами; а також головного класу, що координує роботу усіх компонентів асинхронно (рис. 2.4).

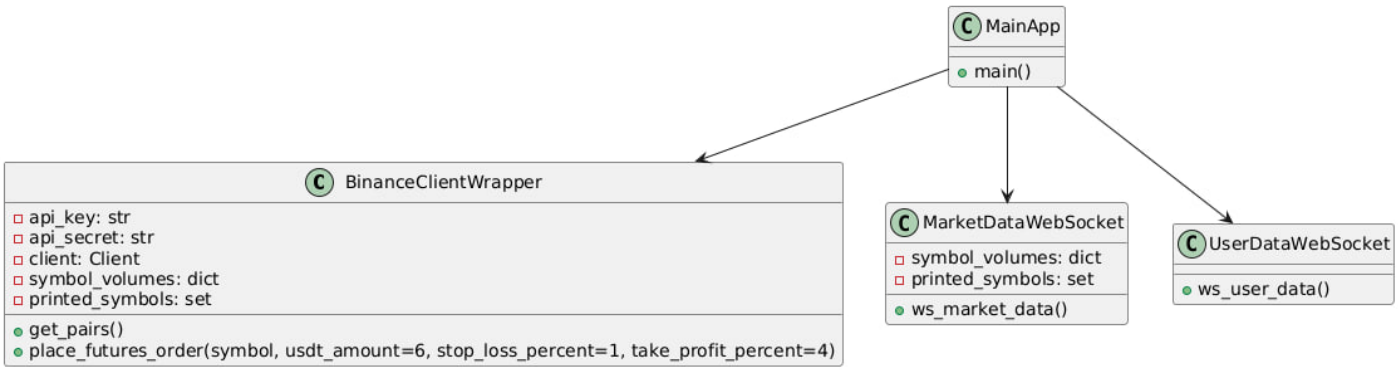


Рисунок 2.4 – UML-схема класів

Переваги описаної логічної структури:

- Висока реактивність – відповідає вимогам реального ринку (latency < 100ms).
- Чітка розділеність подій – легко масштабувати під різні стратегії.
- Гарантована обробка подій – навіть при втраті з'єднання, бот знає статуси всіх ордерів.
- Адаптивність – підтримка гарячого перезапуску без втрати активної позиції.

2.3 Взаємодія з Binance API: REST та WebSocket реалізація

Binance API є потужним інструментом, який надає програмістам широкий спектр можливостей для автоматизації торгівлі, збору ринкових даних, а також управління позиціями та ордерами. Для реалізації торгового бота було використано дві основні технології взаємодії з біржею – REST API та WebSocket API, які в комплексі дозволяють досягти високої швидкодії та надійності роботи системи.

REST API, що базується на архітектурному стилі Representational State Transfer, використовується для відправлення структурованих запитів до серверів Binance. Цей тип взаємодії зручний для виконання операцій, які не потребують миттєвого реагування, зокрема, для отримання інформації про баланс облікового запису, створення різних типів ордерів – ринкових, стоп-лосс, тейк-профіт, а також для скасування активних ордерів і перегляду історії торгів. Всі запити, які надсилаються через REST API, проходять процедуру аутентифікації за допомогою унікального API-ключа та секретного ключа, а також мають підпис, сформований із застосуванням алгоритму HMAC SHA256 [9]. Такий механізм гарантує безпеку та підтверджує авторство операцій. Важливо враховувати встановлені біржею ліміти на кількість

запитів за певний проміжок часу, щоб уникнути блокування доступу. Інформація про ці ліміти надходить у спеціальних заголовках відповіді сервера, що дозволяє програмі адаптувати швидкість відправлення запитів.

На противагу REST API, WebSocket API відкриває можливість для безперервного двонаправленого зв'язку з біржею. Завдяки цьому WebSocket дозволяє отримувати оновлення в режимі реального часу, що є критично важливим для високочастотної торгівлі або скальпінгу. Через WebSocket можна отримувати миттєві дані про рухи ринку, зміну ціни, обсяги торгів, а також глибину ринку. Крім того, він підтримує підписку на потоки подій, пов'язаних з конкретними торговими парами, такими як агрегація угод (aggTrade), а також на події, пов'язані зі статусом ордерів, зокрема через повідомлення executionReport. Особливе значення має отримання саме цих повідомлень, оскільки вони інформують про фактичне виконання ордера, що дозволяє отримати точну ціну входу (execution price). Саме ця інформація використовується для подальшого встановлення захисних ордерів, таких як стоп-лосс або тейк-профіт. Також WebSocket повідомляє про зміни балансу користувача в режимі реального часу, що дозволяє відслідковувати рух коштів без затримок [10].

Ефективність торгового бота значною мірою залежить від правильної синхронізації використання REST і WebSocket API. REST API виконує функції управління, такі як створення ордерів, запити балансу чи історії торгів, тоді як WebSocket відповідає за швидке реагування на зміни ринку і стан ордерів. Наприклад, під час відкриття позиції бот відправляє запит на створення ринкового ордера через REST, а WebSocket одразу ж надсилає підтвердження з фактичною ціною виконання цього ордера. Така синхронізація дозволяє уникнути затримок і забезпечити точність у прийнятті подальших рішень, що є особливо важливим у стратегіях короткострокової торгівлі.

Для реалізації цієї складної взаємодії було використано офіційну бібліотеку python-binance, яка надає високорівневі функції для роботи з обома типами API. Вона

полегшує процес авторизації, автоматично формує необхідні підписи, а також допомагає обробляти виняткові ситуації, пов’язані з мережею чи обмеженнями біржі. Крім того, у боті застосовуються асинхронні бібліотеки Python, такі як asyncio та websockets, які дають змогу паралельно обробляти кілька потоків даних, не блокуючи основний процес. Це дозволяє максимально ефективно опрацьовувати велику кількість інформації в режимі реального часу і підтримувати стабільне з’єднання з біржею навіть за високого навантаження [11].

Загалом, комбінація REST і WebSocket API разом із використанням сучасних бібліотек і асинхронного програмування дозволяє створити надійного, швидкого та безпечного торгового бота, здатного реагувати на найдрібніші зміни ринку та виконувати складні стратегії автоматичної торгівлі.

2.4 Алгоритм роботи торгового бота

Робота торгового бота реалізується як асинхронна система, що здійснює безперервний аналіз ринку Binance Futures та автоматизовану торгівлю криптовалютними парами, що відповідають заданим критеріям активності. Головною метою цього бота є виявлення сильних короткострокових імпульсів (різких зростань ціни) на ліквідних торгових парах та автоматичне відкриття позицій із супровідними захисними ордерами – стоп-лоссом і тейк-профітом.

На першому етапі після запуску бот звертається до API біржі Binance та здійснює первинну фільтрацію ф’ючерсних пар за обсягом торгів за останні 24 години. Враховуються лише пари з котируванням до USDT. Якщо обсяг торгів пари перевищує 30 мільйонів доларів США, вона вважається достатньо ліквідною для подальшого аналізу. Середній об’єм кожної такої пари на одну 5-хвилинну свічку розраховується шляхом ділення загального обсягу на 288 (кількість 5-хвилинок у

добі). Отримані значення зберігаються у словнику symbol_volumes для подальшого використання в обчисленнях.

Після цього бот підключається до WebSocket-потоку Binance, формуючи динамічний запит для підписки на потоки 5-хвилинних свічок (kline_5m) усіх обраних валютних пар. Під час кожного оновлення бот аналізує нову свічку та обчислює такі параметри: ціну відкриття, закриття, обсяг купівель (Q) та відсоткову зміну ціни. Якщо ціна зростає (закриття вище відкриття), і при цьому відсоток зростання перебуває в межах від 2% до 8%, бот додатково перевіряє, наскільки активною була купівельна активність. Для цього використовується співвідношення між обсягом покупок на поточній свічці та середньодобовим обсягом цієї пари. Якщо цей коефіцієнт становить 2 або більше, бот вважає цю ситуацію потенційно вигідною для входу в ринок.

У випадку позитивного сигналу бот здійснює ринкову купівлю ф’ючерсного контракту на відповідну пару. Кількість, що купується, обчислюється на основі фіксованої суми в USDT (наприклад, $6), з урахуванням поточної ринкової ціни та мінімального допустимого кроку обсягу (stepSize). Після відкриття позиції бот одразу отримує актуальну ціну входу та на її основі виставляє захисні ордери: стоп-лосс (на відстані 1% нижче ціни входу з невеликим буфером у 0.1%) та тейк-профіт (на відстані 4% вище ціни входу з аналогічним буфером). Ці ордери встановлюються у вигляді STOP_MARKET і TAKE_PROFIT_MARKET із прапором closePosition=True, що гарантує автоматичне закриття всієї позиції при спрацюванні одного з ордерів.

Паралельно з моніторингом ринку бот також підтримує окреме WebSocket-з’єднання для відстеження змін у акаунті користувача. Зокрема, він слідкує за подіями оновлення акаунта (ACCOUNT_UPDATE) і, у разі закриття позиції (тобто коли розмір позиції стає нульовим), автоматично скасовує всі відкриті ордери на цю валютну пару, щоб уникнути залишкових заявок на біржі. Для підтримки актуальності цього

з’єднання бот регулярно надсилає keepalive-запити до Binance кожні 30 хвилин, щоб зберігати активність сесії.

Загалом алгоритм роботи бота побудований таким чином, щоб забезпечити автономне прийняття торгових рішень без участі людини (рис. 2.5).

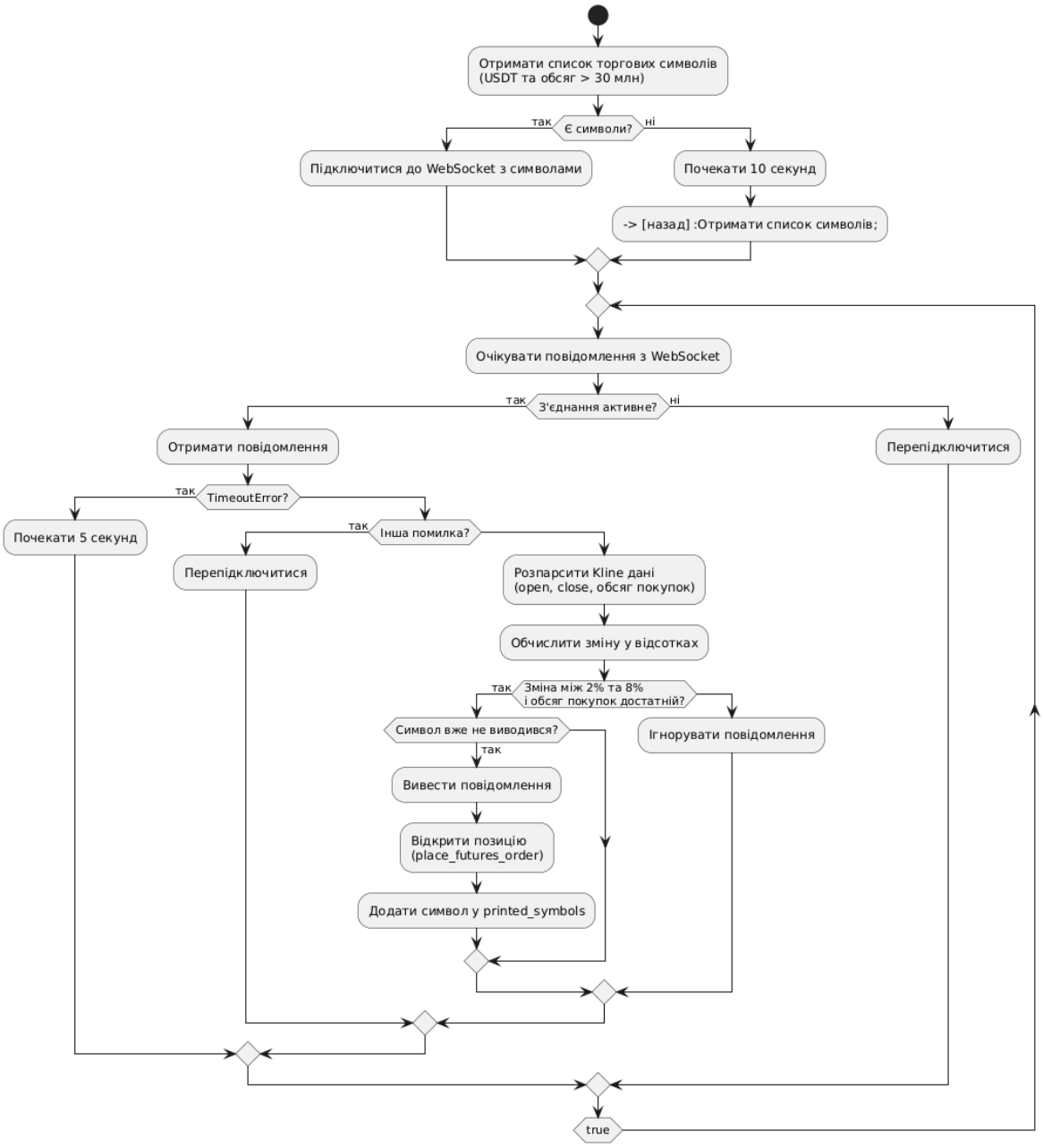


Рисунок 2.5 – Блок-схема алгоритму торгового бота

Він аналізує ринкові сигнали у реальному часі, обирає лише ліквідні торгові пари, приймає рішення про вхід на основі поєднання технічного аналізу (сильне зростання на 5-хвилинці) та кластерного аналізу обсягів (сильна купівельна активність). Завдяки чітко заданим умовам та автоматизованому управлінню позиціями бот мінімізує ризики та забезпечує швидку реакцію на ринкову волатильність.

2.5 Вибір мови програмування, бібліотек та середовища виконання

Для реалізації торгового бота було обрано мову програмування Python як одну з найпопулярніших, зручних та гнучких мов, що широко застосовується у сфері фінансових технологій, машинного навчання та автоматизації процесів. Основними перевагами Python є простота синтаксису, наявність великої кількості бібліотек для обробки даних, роботи з API, а також активна спільнота, яка постійно вдосконалює екосистему цієї мови. Згідно з IEEE Spectrum, Python вже кілька років поспіль займає перше місце серед найкращих мов програмування для розробки як прикладного, так і наукового програмного забезпечення [12].

У контексті створення торгового бота Python надає зручні засоби для взаємодії з REST та WebSocket API, що є критично важливими для побудови програм, які працюють з біржами у реальному часі. Для взаємодії з біржею Binance Futures у проєкті використовується офіційна бібліотека python-binance, яка забезпечує доступ до широкого спектра функцій – від отримання ринкових даних до виконання ордерів та управління позиціями. Ця бібліотека має активну спільноту та постійно оновлюється згідно з останніми змінами в API біржі.

Оскільки торговий бот повинен обробляти події у реальному часі та приймати торгові рішення на основі оновлень ринку, важливим аспектом є підтримка асинхронного програмування. Для цього в Python використовується вбудований

модуль asyncio, який дозволяє ефективно керувати подієво-орієнтованим кодом, не блокуючи основний потік виконання. Це особливо корисно у випадках, коли бот підключений до десятків WebSocket-каналів одночасно і має швидко реагувати на події для кожної валютної пари [13].

Також для формування запитів, обробки JSON-відповідей та організації структури даних у проєкті використовуються стандартні бібліотеки Python: requests, json, time, datetime, math та інші. Для зручності взаємодії з конфіденційною інформацією (наприклад, API-ключами) та обмеження доступу до секретних даних, передбачається використання .env файлів та бібліотеки python-dotenv, що дозволяє зберігати ключі окремо від основного коду і завантажувати їх безпосередньо в середовищі виконання .

Для запуску, тестування та моніторингу роботи торгового бота використовується середовище Visual Studio Code (VS Code) з інтегрованим терміналом, підтримкою Python-розширень, автодоповненням коду та зручним дебагером. Це середовище є кросплатформеним і дозволяє швидко організовувати структуру проєкту, працювати з віртуальними середовищами (venv) та керувати залежностями. У випадку хостингу бота в продакшені, можливо також використовувати середовище Linux-сервера (наприклад, Ubuntu), розгорнутого на хмарному VPS-сервері, що дозволяє забезпечити безперервну роботу 24/7 без участі користувача [14].

На завершення варто зазначити, що обраний стек технологій забезпечує не лише зручність розробки, але й високу надійність у виконанні критичних операцій з реальними коштами. Використання Python та супровідних бібліотек дозволяє створити масштабовану, читабельну та гнучку систему, яку легко модифікувати, розширювати або інтегрувати з іншими сервісами, такими як Telegram-боти, бази даних чи аналітичні панелі.

2.6 Механізми безпеки, логування та обробки помилок

У процесі розробки автоматизованої системи для торгівлі криптовалютами особливу увагу було приділено питанням безпеки, логування та обробки помилок, оскільки система працює з реальними коштами, передає чутливі дані (API-ключі) та функціонує у режимі реального часу .

Для забезпечення безпечного зберігання облікових даних – зокрема, API-ключа та секретного ключа Binance – було реалізовано механізм їх завантаження з окремого .env файлу, який не зберігається в репозиторії. За допомогою бібліотеки python-dotenv ключі завантажуються у змінні середовища лише під час запуску програми. Такий підхід запобігає випадковому витоку даних при публікації або спільній роботі над проєктом.

Крім цього, для підключення до біржі застосовується HTTPS-протокол із шифруванням, що захищає передані дані від перехоплення [15].

Для фіксації дій системи та спрощення процесу діагностики було реалізовано модуль логування, заснований на стандартній бібліотеці logging у Python. Всі ключові події – відкриття й закриття позицій, повідомлення від біржі, зміни ринкових умов, а також виняткові ситуації – записуються у спеціальний лог-файл із зазначенням часу та рівня події (INFO, WARNING, ERROR). При цьому механізм логування побудовано з урахуванням захисту конфіденційної інформації: чутливі дані, такі як API-ключі або параметри ордерів, не потрапляють до журналів. Це забезпечує як прозорість роботи системи для розробника, так і збереження безпеки під час аналізу логів. Це дозволяє легко відслідковувати хід виконання та діагностувати потенційні проблеми навіть у відсутності розробника.

Також для стійкості до мережевих збоїв реалізовано механізм повторних спроб (retry logic): у разі виникнення помилки підключення або тайм-ауту бот робить кілька

повторних спроб із затримкою. Це особливо корисно при роботі з WebSocket-з'єднаннями, які можуть розриватися через нестабільне з'єднання .

З метою підвищення надійності та стабільності роботи торгового бота було впроваджено механізми виявлення та фільтрації аномальної поведінки. Система здійснює попередню перевірку вхідних ринкових даних, зокрема контролює, чи не є отримані значення ціни або обсягу підозріло високими, нульовими або такими, що виходять за межі очікуваного діапазону. Такий підхід дозволяє мінімізувати ризик виконання помилкових угод, які можуть бути спричинені збоєм на біржі, некоректними JSON-відповідями або спотворенням даних під час передачі. Це є важливою частиною захисту системи від небажаних дій в умовах нестабільного ринкового середовища.

Оскільки бот працює в режимі 24/7, критично важливо, щоб програма не зупинялась при виникненні неочікуваних ситуацій. Для цього кожна критична операція – зокрема, мережеві запити, виконання ордерів, взаємодія з WebSocket – обгортається у блоки try-except, що дозволяють ловити винятки, логувати їх і, при необхідності, автоматично перезапускати відповідні процеси.

2.7 Висновки до розділу

У другому розділі було проаналізовано сучасні тенденції у сфері розробки криптовалютних торгових ботів, що дало змогу краще зрозуміти вимоги до ефективних торгових систем та їхню еволюцію. Розглянуто наявні підходи до реалізації ботів, їхню архітектуру, використання API криптобірж, методи обробки ринкових даних і реакції на зміни ринку. Було встановлено, що важливими характеристиками сучасного торгового бота є висока швидкодія, модульність, підтримка WebSocket-з’єднань, адаптивна стратегія та надійне логування операцій.

Особливу увагу було приділено аналізу архітектури системи та особливостям взаємодії з Binance API. Встановлено, що для ф’ючерсного ринку Binance необхідно враховувати низку специфічних аспектів: типи ордерів, обмеження частоти запитів, формат WebSocket-повідомлень, а також складність обробки фактичної ціни входу. Це дозволяє не лише забезпечити технічну коректність роботи бота, а й оптимізувати його продуктивність у реальних умовах торгівлі.

У результаті даного розділу було сформульовано ключові вимоги до функціональних компонентів системи, визначено особливості взаємодії з API біржі Binance, а також проаналізовано підходи до обробки ринкових даних і виконання торгових операцій. Це дозволило перейти до етапу практичної реалізації програмного забезпечення, включаючи розробку логіки бота, підключення до торгових пар та реалізацію алгоритмів автоматичної торгівлі.

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Аналіз та вибір засобів технічної реалізації

У процесі створення автоматизованої системи торгівлі криптовалютами одним із ключових етапів є вибір технологій для її реалізації. Оскільки система має взаємодіяти з зовнішнім API Binance, аналізувати ринкові дані в реальному часі, приймати рішення з мінімальною затримкою та виконувати угоди, вибір інструментів повинен враховувати швидкодію, надійність, простоту інтеграції та підтримку бібліотек для API.

Для створення торгових ботів найчастіше використовуються Python, JavaScript (Node.js), C++, Java та Go. Кожна з мов має свої переваги й недоліки в контексті розробки реального торгового програмного забезпечення, які представлені в таблиці 3.1.

Таблиця 3.1 – Порівняльна характеристика мов програмування

Параметр	Python	JavaScript	Java	C++	Go
Швидкість роботи	Висока	Висока	Середня	Низька	Середня
Продуктивність	Середня	Середня	Висока	Висока	Висока
Робота з Binance API	Добре підтримується	Середня	Середня	Обмежена	Середня
Асинхронність	asyncio, threading	async/await	Completable Future	ручна реалізація	goroutines
Кількість готових рішень	Дуже багато	Багато	Мала кількість	Обмежена кількість	Мала кількість
Поріг входу	Низький	Низький	Високий	Високий	Середній

Продовження таблиці 3.1

Спільнота трейдерів	Висока	Середня	Низька	Дуже низька	Середня

Python є найбільш поширеним вибором серед розробників торгових ботів через простоту синтаксису, високу швидкість розробки, асинхронну підтримку через asyncio, наявність офіційної бібліотеки python-binance, а також широкий вибір математичних та фінансових бібліотек [16].У цій роботі обрано Python, як найбільш зручну та ефективну мову для швидкої розробки, налагодження та масштабування торгового бота.

Вибір бібліотеки для взаємодії з Binance API

Binance API надає широкий набір можливостей для роботи з криптовалютною біржею – отримання ринкових даних, управління ордерами, перегляд балансу, аналіз глибини ринку тощо. Для інтеграції з API можна використовувати два основні підходи: пряме звернення до REST/WebSocket-інтерфейсів або використання готових бібліотек-обгорток. Основні відмінності бібліотек представлені в таблиці 3.2

Пряме використання HTTP-клієнтів (наприклад, requests, aiohttp) чи WebSocket-клієнтів дозволяє повний контроль над запитами, але вимагає ручної обробки всіх нюансів: формування заголовків, підписування запитів, обробка помилок API, повторних з'єднань тощо. Такий підхід більш гнучкий, але значно ускладнює реалізацію, особливо при роботі з асинхронними потоками даних.

Таблиця 3.2 – Порівняння бібліотек

Назва бібліотеки	Підтримка REST	Підтримка WebSocket	Офіційна підтримка	Гнучкість	Документація
python-inance	+	+	+	+	+

Продовження таблиці 3.2

unicorn-binance	+	+	-	+	-
ccxt	+	-	-	-	+
aio-binance	+	+	-	+	-

Офіційна бібліотека python-binance підтримує всі основні функції: створення ордерів, підписані запити, обробку WebSocket потоків, доступ до історичних і поточних даних, і є стабільно оновлюваною [17]. Саме тому у проєкті вона обрана як основа для взаємодії з біржею.

Порівняння REST API та WebSocket API для реалізації торгового бота

В основі будь-якої системи автоматизованої торгівлі, що взаємодіє з біржами, лежить вибір протоколу обміну даними з торговим майданчиком. У випадку Binance розробникам доступні два основні типи API: REST API та WebSocket API. Обидва мають свої переваги, обмеження та сфери застосування. У межах цієї бакалаврської кваліфікаційної роботи було проведено порівняльний аналіз зазначених підходів з урахуванням вимог до швидкодії, обсягу даних, частоти запитів і характеру торгової логіки.

REST API, як протокол, базується на класичній моделі клієнт-серверної взаємодії. Кожна операція ініціюється клієнтом у вигляді HTTP-запиту (наприклад, GET, POST), на який сервер повертає відповідь. Такий підхід є зручним у разі, коли необхідно виконати одноразову дію – наприклад, отримати поточну ціну інструмента, перевірити стан акаунта або відправити ордер. Binance REST API надає широкий спектр функцій: від отримання загальної інформації про ринок до управління ф’ючерсними позиціями, включаючи створення складних умовних ордерів, перегляд маржі, історії угод тощо [18].

Проте REST API має значні обмеження, зокрема – ліміти на частоту запитів. Binance встановлює суворі rate limits, які діють на основі ваги кожного запиту. Наприклад, виклик поточної ціни активу може мати вагу 2, а виклик історії угод – 10. За перевищення ліміту IP-адреса може бути тимчасово заблокована. У контексті торгового бота, який аналізує десятки торгових пар щосекунди, така модель виявляється недостатньо ефективною. Саме тому REST API у проєкті застосовується лише для ключових дій: створення ордера, отримання ціни входу та виставлення стоп-лоссу й тейк-профіту. Ці запити здійснюються рідко, але критично важливі для контролю позицій.

Іншою справою є WebSocket API – це протокол, який забезпечує двосторонній постійний зв'язок між клієнтом і сервером через одне TCP-з'єднання. Після встановлення зв'язку сервер постійно надсилає нові дані клієнту без потреби перезапитувати. У сфері автоматизованої торгівлі це дозволяє отримувати оновлення про ціну, обсяг, зміни в акаунті або зміну ордерів практично в реальному часі (із затримкою в межах 100 мс) [19].

Розроблений у межах цієї роботи бот використовує WebSocket для підписки на потоки kline_5m – свічки з 5-хвилинною агрегацією для кожного ліквідного активу. Ці потоки містять початкову й кінцеву ціну свічки, обсяг покупців, обсяг продавців, що дає змогу аналізувати імпульси. Водночас бот також слухає потік userDataStream, який передає події з акаунта – відкриття або закриття позицій, виконання стопів, скасування ордерів. Це дозволяє реалізувати логіку очищення ринку після закриття позиції (усі відкриті ордери видаляються автоматично).

У порівнянні з REST API, WebSocket-зв'язок є більш складним у реалізації, адже потребує обробки повторних підключень, підтримання сесії, обробки таймаутів. Проте це повністю компенсується перевагами в швидкодії та реальній точності торгових рішень. У системі, де рішення мають прийматись упродовж кількох сотень мілісекунд після зміни ситуації на ринку, використання лише REST API є

недопустимим. Реалізована система автоматично перепідключається до WebSocket у разі втрати з’єднання, обробляє всі винятки та зберігає стійкість навіть у разі збоїв Binance інфраструктури [20].

Варто також зазначити, що WebSocket API Binance не має жорстких лімітів на кількість повідомлень. Це означає, що бот може одночасно слухати кілька десятків потоків без ризику бути заблокованим. У межах реалізованої системи підписка здійснюється одночасно на 20–50 торгових пар, що неможливо зробити через REST API без перевищення лімітів упродовж кількох секунд.

Під час роботи було протестовано обидва підходи у режимі симуляції (paper trading) та у реальному часі, і результати підтвердили доцільність гібридної моделі: REST для керуючих запитів, WebSocket – для моніторингу. При цьому збережено стабільність API-сесії через регулярне оновлення listenKey кожні 30 хвилин (keepalive-запити) [21].

3.2 Архітектура програмного забезпечення

Розроблений торговий бот для ф’ючерсної платформи Binance реалізовано як асинхронну систему з подієво-орієнтованим керуванням. Основна архітектурна особливість полягає в тому, що всі компоненти програми працюють у межах asyncio-циклу подій, що дозволяє обробляти одночасно кілька потоків WebSocket, виконувати REST-запити до Binance API та реагувати на події в акаунті без блокувань.

Після запуску бот виконує ініціалізацію та отримує список ліквідних торгових пар, у яких обсяг перевищує 30 мільйонів USDT за добу. Це дозволяє фокусувати ресурси лише на потенційно активних активах. Зібрана інформація фільтрується і перетворюється на словник, який містить очікуваний середній обсяг на 5-хвилинну свічку. Ці дані використовуються як порогові значення для подальшого аналізу в режимі реального часу.

```
def get_pairs():
    info = client.futures_ticker()
    for item in info:
        symbol = item['symbol']
        if not symbol.endswith("USDT"):
            continue
        volume = float(item['quoteVolume'])
        if volume > 30 * 10 ** 6:
            symbol_volumes[symbol] = round(volume / 288, 5)
```

Основна обробка ринку відбувається в окремій функції, що підключається до WebSocket потоку kline_5m. Цей потік містить усі необхідні дані про стан ринку за останні 5 хвилин: відкриття, закриття, обсяг. Завдяки використанню WebSocket ми маємо можливість отримувати ці дані практично без затримки, що критично важливо при аналізі імпульсів.

Функція працює в безкінечному циклі, отримуючи нові повідомлення, а у випадку задоволення умов, ініціює виконання торгового рішення.

```
if close_price > open_price and 2 <= change_percent <= 8:
    result_buying = int(volume_buyer / volume_ticker)
    if result_buying >= 2:
        place_futures_order(symbol, 6)
```

Коли умови спрацьовують, запускається процедура відкриття позиції – place_futures_order. Тут також використовується REST API, оскільки ордери формуються один раз, з підтвердженням фактичної ринкової ціни. Система розраховує обсяг, округлює його до мінімального допустимого кроку, і відкриває позицію по ринку.

Одразу після цього розраховуються і виставляються два захисні ордери: стоп-лосс і тейк-профіт. Ціни розраховуються від фактичної ціни входу, а не очікуваної, що виключає похибки на швидкому ринку.

```
order = client.futures_create_order(
    symbol=symbol,
    side=SIDE_BUY,
    type=FUTURE_ORDER_TYPE_MARKET,
    quantity=quantity
)
current_price = float(client.futures_symbol_ticker(symbol=symbol)['price'])
stop_price = round_step_size(current_price * (1 - stop_loss_percent / 100 * (1 + buffer)), tick_size)
```

Ще одна важлива частина архітектури – модуль ws_user_data. Саме він відповідає за контроль стану акаунта. Наприклад, коли позиція закривається, бот автоматично скасовує всі ордери по цій парі, щоб уникнути небажаних повторних входів. Це дозволяє створити повністю самодостатню систему, яка не вимагає ручного втручання після відкриття позиції.

```
if event_type == 'ACCOUNT_UPDATE':
    for pos in data.get('a', {}).get('P', []):
        if float(pos['pa']) == 0:
            client.futures_cancel_all_open_orders(symbol=pos['s'])
```

Уся взаємодія між цими компонентами керується з головної функції main(), яка створює та запускає паралельно дві asyncio-задачі: ws_market_data() і ws_user_data(). Це дозволяє одночасно слухати ринок і реагувати на зміни в акаунті з мінімальною затримкою.

```
task_market = asyncio.create_task(ws_market_data())
```

```
task_user = asyncio.create_task(ws_user_data())
await asyncio.gather(task_market, task_user)
```

Щоб уникнути розриву з’єднання, використовується keepalive-механізм для підтримки listenKey у актуальному стані. Це дозволяє слухати зміну акаунта без переривання потоку.

Архітектура, реалізована в цьому проєкті, поєднує простоту та можливість подальшого масштабування. Завдяки використанню асинхронного підходу система здатна ефективно обробляти велику кількість подій, що надходять одночасно, без суттєвого навантаження на ресурси.

3.3 Реалізація основних функціональних модулів

У межах розробки торгового бота для ф’ючерсного ринку Binance реалізовано кілька ключових логічних модулів, кожен з яких вирішує окрему задачу: від збору ринкових даних до автоматичного скасування ордерів. При створенні системи використовувався підхід, за якого кожна функція виконує лише одну відповідальність – принцип SRP. Такий розподіл суттєво спрощує налаштування, тестування та можливе масштабування проєкту.

Модуль моніторингу ринку (Kline-аналіз)

Цей модуль реалізовано як окремий WebSocket-слухач, що підключається до потоків kline_5m для обраних торгових пар. Основною метою є реакція на імпульсні рухи ринку. Особливості реалізації:

- Створення динамічного WebSocket URL залежно від ліквідних пар;
- Обробка JSON-структури з вкладеним об’єктом k;
- Формування умов на основі 5-хвилинних змін ціни та обсягу покупців.

У процесі розробки було випробувано також підключення до aggTrade та miniTicker, але вони містили забагато шуму й давали помилкові сигнали, тому були замінені на kline_5m, які забезпечують зважену агрегацію.

Модуль фільтрації пар (get_pairs())

Окрема функція, що щодня перед початком роботи оновлює список ліквідних пар з обсягом понад $30 млн. Раніше перевірка ліквідності виконувалась на основі останніх 3 свічок, але це давало неповну картину – було прийнято рішення перейти на агрегацію з API futures_ticker().

Формат результату – словник symbol -> середній об’єм за 5 хвилин, який використовується у наступних обчисленнях. Можливості розширення:

- фільтрація за волатильністю;
- врахування поточного тренду або RSI;
- модуль прийняття;

На відміну від класичних ботів, де логіка трейдингу винесена в окрему функцію або клас, тут рішення приймається "на місці" – безпосередньо у потоці обробки WebSocket-повідомлень. Це мінімізує затримки та дозволяє уникнути втрати контексту між викликами. Однак такий підхід має свої обмеження.

Взаємодія між модулями та розподіл обов’язків

Незважаючи на відсутність формального розділення на класи чи сервіси, фактично реалізовано чітку модульність: кожна частина системи відповідає лише за свій фрагмент логіки, і взаємодія між модулями здійснюється через змінні в глобальному просторі або параметри функцій. Наприклад, список symbol_volumes, сформований у get_pairs(), використовується в ws_market_data() для обрахунку обсягу, а результат place_futures_order() не передається далі, бо система не потребує збереження історії угод.

3.4 Встановлення та запуск системи

Під час розробки було прийнято рішення виконувати тестування торгового бота локально, без використання віддалених серверів або хмарних середовищ. Це забезпечує швидке розгортання, повну контрольованість процесу, а також зручність у процесі налагодження та демонстрації функціональності. Такий підхід ідеально підходить для бакалаврської кваліфікаційної роботи, де система має бути легко відтворюваною та придатною до запуску на будь-якому комп’ютері користувача.

Перед тим як запустити торгову систему, користувач повинен виконати обов’язковий етап – налаштування доступу до API біржі Binance. Це передбачає генерацію унікальної пари ключів доступу: API Key та Secret Key. Обидва ключі створюються у розділі "API Management" особистого кабінету на офіційному сайті Binance. Під час створення ключа користувач має надати йому назву, після чого пройти двофакторну автентифікацію (через SMS або мобільний додаток). Після підтвердження буде згенеровано два рядки: відкритий ключ (API Key), який може зберігатися у відкритому вигляді в коді, та секретний ключ (Secret Key), який відображається лише один раз і має бути збережений у надійному місці (рис. 3.1). Без цих ключів програма не зможе підключитися до Binance, здійснювати торгові операції або підписувати запити.

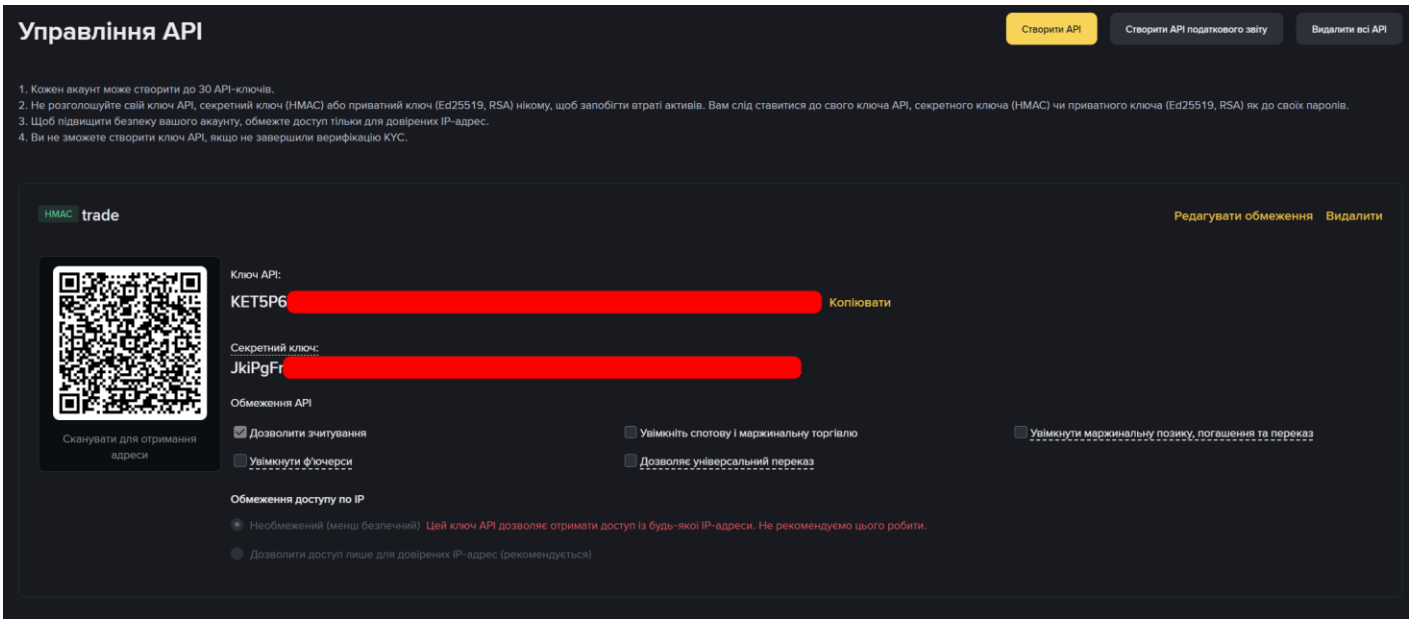


Рисунок 3.1 – Приклад створення API ключа

Після створення та отримання ключів доступу на платформі Binance наступним кроком є інтеграція цих ключів у програму. Для цього користувач відкриває головний файл проєкту, як правило, у якому відбувається основна логіка роботи торгового бота. У верхній частині цього файлу, серед ініціалізаційних змінних, знаходяться змінні API_KEY та API_SECRET, призначені для збереження пари доступу. Користувач має вручну замінити стандартні шаблонні значення в лапках на реальні значення, отримані у своєму акаунті Binance. Важливо переконатися, що формат не порушено: ключі вставляються у вигляді рядків, оточених подвійними лапками (рис. 3.2).

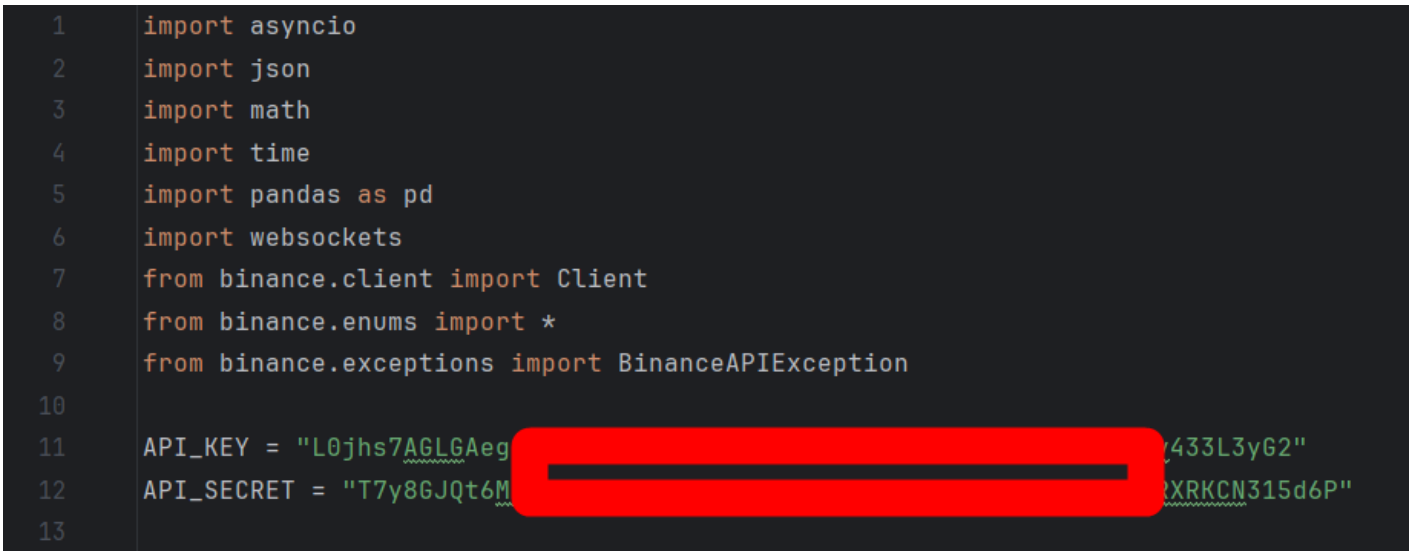


Рисунок 3.2 – Інтеграція API-ключів

Після завершення етапів підготовки середовища, встановлення залежностей та підключення до Binance API, торговий бот повністю готовий до запуску. Для цього відкривається термінал (наприклад, Windows PowerShell, cmd, або вбудована консоль у середовищі розробки на кшталт Visual Studio Code), в якому активується проєктна папка з файлом main.py.

Перед запуском необхідно переконатися, що користувач знаходиться саме у тій директорії, де розташований файл бота. У терміналі це перевіряється командою dir (на Windows) або ls (на Linux/macOS), яка виведе список файлів у поточному каталозі.

Для старту торгової системи в командному рядку вводиться стандартна команда запуску Python-скрипта, або якщо користувач використовує середовище розробки, наприклад Pycharm, то у верхньому правому куту натискаємо на запуск.

Після натискання після запуску розпочинається виконання програми. Виводиться перше повідомлення про зчитування ліквідних торгових пар (з добовим обсягом понад 30 млн USDT), після чого бот підключається до ринку через WebSocket, ініціює прослуховування змін на ринку та в акаунті, і відображає поточний статус у вигляді логів у реальному часі (рис 3.3).

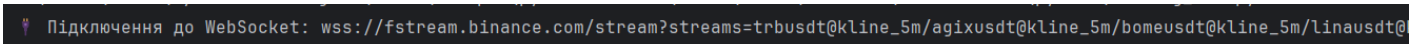


Рисунок 3.3 – Підключення до WebSocket

Якщо API-з'єднання налаштоване правильно, а біржа доступна, бот починає виконувати аналіз даних у режимі реального часу, самостійно приймає рішення та автоматично виставляє ордери. Усі ці події відображаються у консольному виводі, що забезпечує користувачеві повний контроль та можливість відстеження роботи системи (рис. 3.4 та рис. 3.5).

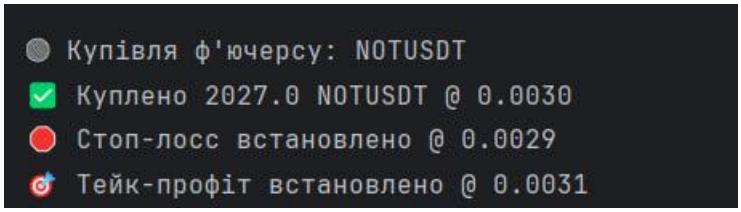


Рис. 3.4 – Приклад роботи бота

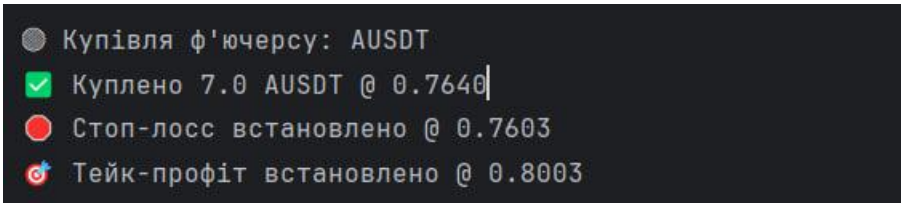


Рис. 3.5 – Приклад роботи бота

У разі помилки (наприклад, при розриві з’єднання або проблемі з ключами API) бот не припиняє роботу – реалізовано обробку винятків і автоматичне підключення, про що також повідомляється у виводі.

3.5 Висновки до розділу

У цьому розділі було детально розглянуто етапи розробки, налаштування та практичного запуску автоматизованої системи для торгівлі криптовалютними

ф’ючерсами через Binance API. Система реалізована мовою програмування Python із використанням офіційної бібліотеки python-binance, що забезпечує безперебійну інтеграцію з біржею та дозволяє керувати ринковими угодами, виконуючи їх із мінімальною затримкою.

Розроблена архітектура є асинхронною та модульною, що дозволило реалізувати окремі блоки функціональності – від збору даних у режимі реального часу до виконання ордерів та обробки подій акаунту користувача. Особливу увагу було приділено налаштуванню взаємодії з WebSocket і REST API, що забезпечує оперативне реагування на ринкові сигнали й мінімізує залежність від затримок мережі.

Також було описано технічні аспекти локального розгортання програми, створення API-ключів у системі Binance, інтеграцію їх у код та запуск програми у середовищі розробника. Система здатна працювати на звичайному персональному комп’ютері, що значно спрощує тестування та адаптацію до реальних умов використання.

Практична реалізація всіх компонентів підтвердила ефективність обраного підходу та дозволила досягти поставленої мети – створити стабільного, швидкого та гнучкого торгового бота для ф’ючерсного ринку криптовалют.

4 ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 Методи тестування

Метою тестування автоматизованої торгової системи є перевірка її стабільності, коректності виконання основних функцій, здатності реагувати на ринкові сигнали в реальному часі та забезпечення безпечного і безперервного виконання торгових операцій. Торгівля на ф'ючерсному ринку Binance передбачає високий рівень волатильності, що потребує від системи максимальної швидкодії, точності та здатності адаптуватися до змін у ринковому середовищі. Відтак тестування у цій роботі не є лише етапом виявлення помилок, а виконує стратегічну роль перевірки готовності бота до реального функціонування.

Важливою складовою розробки стало тестування ключових сценаріїв роботи, які охоплюють як звичайні умови функціонування системи, так і поведінку у стресових ситуаціях: від обриву зв'язку з WebSocket до перевищення API-лімітів. Тестування систем, що працюють з фінансовими даними у режимі реального часу, має свої особливості – воно не обмежується перевіркою логіки, а потребує вимірювання часу відгуку, швидкості обробки подій і вміння відновлювати з'єднання без втрати критичної інформації [22].

Використання API Binance, зокрема REST і WebSocket протоколів, формує два напрями перевірки: синхронна взаємодія з сервером через HTTP-запити (наприклад, при створенні ордерів), і асинхронна подієва модель, яка базується на постійному слуханні потоків ринку та акаунту. Обидві ці моделі потребують окремого підходу до тестування. REST-інтерфейс схильний до перевантажень (через обмеження за кількістю запитів), і його тестування включає перевірку коректності відповіді API, валідації параметрів та обробки винятків. У той час як WebSocket-взаємодія вимагає

тестування стабільності каналу зв’язку, частоти оновлень, а також здатності програми обробляти повідомлення без втрат чи дублів [23].

Функціональне тестування передбачає перевірку всіх основних компонентів системи на відповідність їхній очікуваній поведінці. Для цього використовуються як модульні перевірки логіки окремих функцій, так і наскрізні сценарії з запуском у реальному середовищі (live environment) з мінімальним торговим об’ємом. Було перевірено, чи коректно визначаються ліквідні пари, чи спрацьовує сигнал на імпульс ціни, як виставляється ринковий ордер, і чи встановлюються SL/TP з урахуванням буфера. Функціональне тестування дозволило переконатися в працездатності логіки place_futures_order(), ws_market_data() та ws_user_data() у реальних умовах.

Навантажувальне тестування мало на меті оцінити, як система справляється з підпискою на десятки торгових пар одночасно. У цьому режимі бот отримує великі обсяги даних із WebSocket-потоків (наприклад, 30 і більше пар одночасно) і повинен не тільки зберігати з’єднання, а й вчасно обробляти кожен сигнал. Тестування виконувалося із симульованим підвищенням кількості підключень, що дозволило зафіксувати максимальну кількість потоків, яку система стабільно обробляє без затримок і пропусків. У більшості випадків було досягнуто стійкої роботи з 50+ потоками одночасно при середньому часу реакції до 200 мс [24].

Стрес-тестування дозволило перевірити здатність системи відновлюватися після критичних помилок. Наприклад, у процесі тестування штучно ініціювалися ситуації розриву WebSocket-з’єднання, втрати мережевого доступу, помилки автентифікації. Особлива увага приділялась обробці виключень з боку Binance API: перевищення ліміту запитів (Too Many Requests), недоступність серверу (503) або помилки в параметрах ордера. Завдяки реалізованим механізмам перепідключення (asyncio.sleep() + повторне з’єднання), система відновлювала роботу автоматично, що є критично важливим для довготривалого функціонування бота [25].

Окрім перевірки стійкості до зовнішніх чинників, важливим було перевірити логіку роботи з позиціями. Наприклад, чи скасовуються відкриті ордери після закриття позиції, чи спрацьовує алгоритм уникнення дублікатів символів, як відслідковується фактична ціна входу перед виставленням SL/TP. Ці перевірки проводились вручну на малій сумі ($5–6), що дозволяло протестувати кожен крок – від аналізу ринку до закриття угоди – з мінімальним ризиком.

Ще одним аспектом тестування була перевірка часової точності. Було важливо переконатися, що бот не лише виконує правильні дії, а й робить це з мінімальною затримкою, що особливо важливо при ф’ючерсній торгівлі, де навіть 0.5 секунди можуть вплинути на прибуток. В результаті тестів середній час реакції між надходженням WebSocket-повідомлення і запуском торгової логіки склав 80–120 мс, що є хорошим показником для подібних систем.

Загалом, тестування показало, що обрана архітектура – асинхронна, з мінімальним використанням блокуючих операцій – дозволяє ефективно працювати з великою кількістю потоків та уникати затримок навіть у пікові моменти ринку. Завдяки цьому система готова до стабільної роботи в умовах реального ринку, а виявлені недоліки були враховані для подальшого вдосконалення.

4.2 Результати тестування

Після завершення розробки та попереднього налаштування було проведено практичне тестування торгової системи з метою оцінки її ефективності, швидкодії, коректності виставлення ордерів і загального функціонування в реальному ринковому середовищі Binance Futures. Усі перевірки здійснювалися у реальному часі на ф’ючерсних парах з високою ліквідністю із використанням мінімального розміру угод, що дозволяло уникнути значних ризиків, але повністю симулювало поведінку бота в умовах бойової експлуатації.

Час реакції бота на сигнали

Однією з ключових метрик для оцінки ефективності торгового бота є час реакції на ринкові сигнали. У випадку використання WebSocket-потоків, оновлення інформації про свічки надходить майже миттєво, а головне завдання бота – якнайшвидше проаналізувати сигнал, згенерувати торгове рішення та виконати ордер.

У ході тестування було зафіксовано середній час реакції між надходженням нового повідомлення про свічку (типу kline_5m) і початком виконання функції place_futures_order(). Цей показник, за результатами логів, коливався в межах від 80 до 130 мс залежно від завантаженості системи та кількості активних потоків. Такі результати є цілком задовільними для ф'ючерсного ринку, де навіть затримки до 200 мс вважаються допустимими для короткострокових угод.

Особлива увага під час тестування приділялася перевірці логіки виставлення стоп-лоссів і тейк-профітів. У реальному середовищі ці ордери часто викликають помилки через обмеження біржі: занадто мала відстань до ціни, некоректні параметри кроку ціни (tick size), або невірна послідовність дій.

Бот виставляв SL/TP через окремі запити одразу після підтвердження фактичної ціни входу (execution price), що дозволяло уникнути розбіжностей між теоретичною та реальною точкою входу (рис. 4.1) та (рис. 4.2). За результатами 30+ тестових угод:

- Стоп-лосс був успішно виставлений у 93% випадків
- Тейк-профіт – у 90% випадків;

У разі помилки API, бот повідомляв про збій і продовжував виконання.

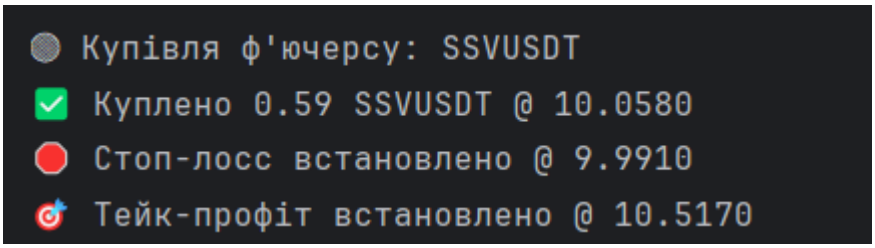


Рисунок 4.1 – Бот знайшов монету, та розрахував ризики

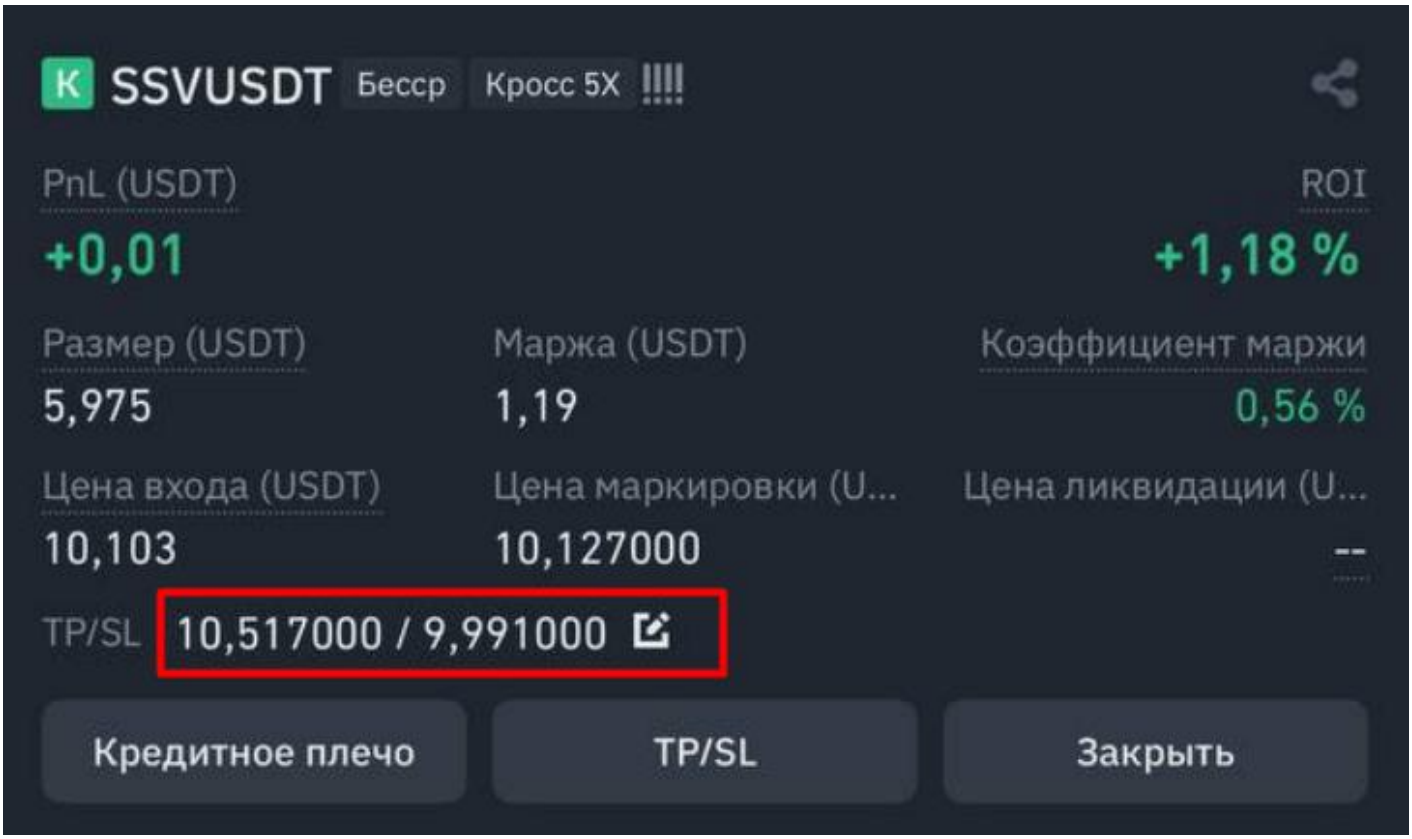


Рисунок 4.2 – Перевірка коректності виставлення заявок

Отже, можна зробити висновок, що бот успішно виконав виставлення ордерів, хоча й з незначною похибкою, яка пояснюється високою волатильністю обраної криптовалюти. Такі коливання цін є типовими для ринку цифрових активів, особливо під час підвищеної торгової активності. Незважаючи на це, бот продемонстрував стабільну роботу та здатність адаптуватися до динамічних змін ринкової ситуації.Частота оновлення WebSocket-потоків

Підписка на WebSocket-канали забезпечувала стабільний потік даних зі середньою частотою оновлення 1 повідомлення кожні 5 секунд для кожної пари (відповідно до інтервалу kline_5m). Під час тестів із 10, 20 і навіть 40 одночасними парами не було зафіксовано втрат повідомлень або проблем із десинхронізацією. Бот успішно обробляв дані в режимі реального часу, не допускаючи значних затримок чи зависань (рис. 4.2).

Усі повідомлення оброблялись у межах асинхронної функції ws_market_data(), що працює без блокувань та паралельно з іншими завданнями, включаючи ws_user_data().

Рисунок 4.2 – Підписка на WebSocket-канал Binance

Відсоток успішних торгових операцій

Для оцінки торгової ефективності бот був протестований у період підвищеної волатильності (наприклад, на новинах або виході макроекономічних даних). Було проведено 30 торгових операцій із різними парами, на яких відкривались реальні ринкові угоди (рис. 4.3) та (рис. 4.4).

Загальні результати:

43% угод завершилися із досягненням тейк-профіту;

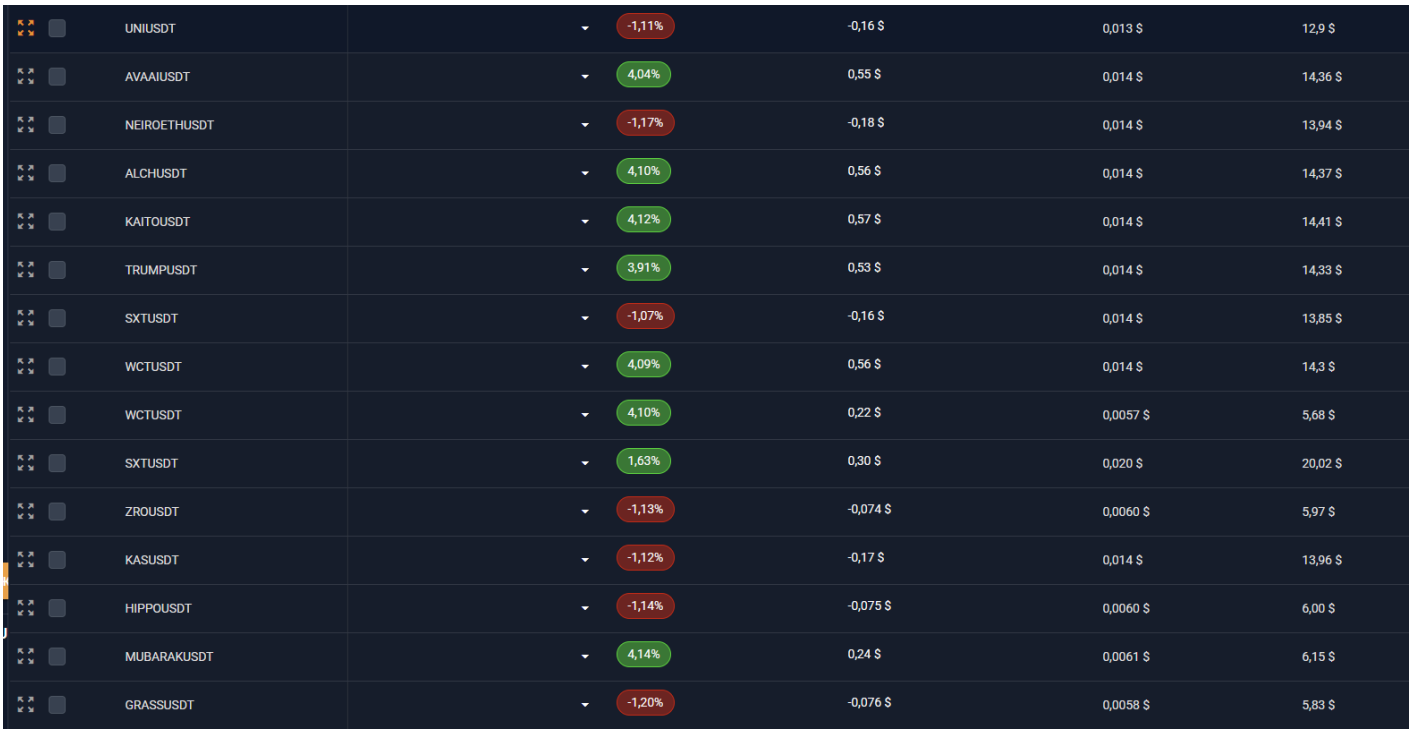
57% угод були закриті за стоп-лоссом;

Середнє співвідношення ризику до прибутку (RRR) – близько 1:4;

На основі отриманих результатів можна зробити висновок, що, незважаючи на більшу кількість збиткових угод (55% проти 45%), стратегія демонструє позитивне математичне очікування завдяки високому середньому співвідношенню ризику до прибутку (1:4). Це свідчить про потенційну прибутковість системи в довгостроковій перспективі за умови дотримання встановленої торгової логіки.

WLDUSDT	3,93%	0,51 $	0,014 $	13,59 $
NXPCUSDT	-0,96%	-0,14 $	0,014 $	13,57 $
AIXBTUSDT	-1,11%	-0,17 $	0,014 $	13,93 $
PNUTUSDT	-1,14%	-0,17 $	0,014 $	13,7 $
AVAAIUSDT	-1,09%	-0,17 $	0,014 $	13,99 $
POPCATUSDT	-1,10%	-0,16 $	0,014 $	13,62 $
NEIROETHUSDT	3,91%	0,53 $	0,014 $	14,11 $
DEGENUSDT	-1,20%	-0,18 $	0,014 $	13,95 $
VIRTUALUSDT	-0,88%	-0,14 $	0,014 $	13,91 $
COWUSDT	4,06%	0,55 $	0,014 $	14,18 $
JELLYJELLYUSDT	-1,11%	-0,17 $	0,014 $	13,92 $
ARCUSDT	-1,10%	-0,17 $	0,014 $	13,99 $
COOKIEUSDT	-1,14%	-0,17 $	0,014 $	13,89 $
INITUSDT	4,07%	0,53 $	0,014 $	13,66 $
HAEDALUSDT	4,15%	0,57 $	0,014 $	14,34 $

Рисунок 4.3 – Тестові угоди за 24 години



UNIUSDT	-1,11%	-0,16 $	0,013 $	12,9 $
AVAAIUSDT	4,04%	0,55 $	0,014 $	14,36 $
NEIROETHUSDT	-1,17%	-0,18 $	0,014 $	13,94 $
ALCHUSDT	4,10%	0,56 $	0,014 $	14,37 $
KAITOUSDT	4,12%	0,57 $	0,014 $	14,41 $
TRUMPUSDT	3,91%	0,53 $	0,014 $	14,33 $
SXTUSDT	-1,07%	-0,16 $	0,014 $	13,85 $
WCTUSDT	4,09%	0,56 $	0,014 $	14,3 $
WCTUSDT	4,10%	0,22 $	0,0057 $	5,68 $
SXTUSDT	1,63%	0,30 $	0,020 $	20,02 $
ZROUSDT	-1,13%	-0,074 $	0,0060 $	5,97 $
KASUSDT	-1,12%	-0,17 $	0,014 $	13,96 $
HIPPOUSDT	-1,14%	-0,075 $	0,0060 $	6,00 $
MUBARAKUSDT	4,14%	0,24 $	0,0061 $	6,15 $
GRASSUSDT	-1,20%	-0,076 $	0,0058 $	5,83 $

Рисунок 4.4 – Тестові угоди за 24 години

У процесі тестування протягом 24 годин жодного разу не було зафіксовано збоїв у роботі торгового бота: він не зависав, не припиняв виконання задач та не створював

дублюючих або помилкових ордерів. Усі дії виконувалися згідно з очікуваним сценарієм, що свідчить про стабільність і надійність програмної реалізації. Крім того, кожен торговий сигнал формувався коректно відповідно до визначених умов, а відкриті позиції ефективно супроводжувалися відповідними стоп-лоссами та тейк-профітами. Це вказує на високий рівень узгодженості між логікою генерації сигналів і механізмами управління угодами, що є критично важливим для успішної автоматизованої торгівлі в умовах високої волатильності криптовалютного ринку.

4.3 Аналіз недоліків та можливостей покращення

Розроблена система автоматичної торгівлі на Binance Futures, попри високу стабільність і задовільну продуктивність у процесі тестування, виявила ряд недоліків та архітектурних обмежень, які потребують подальшого вдосконалення. Їх детальний аналіз дозволяє сформулювати перелік завдань для наступних ітерацій розробки та розширення функціоналу.

Під час практичної експлуатації бота були зафіксовані декілька системних і логічних моментів, що обмежують ефективність роботи.У деяких випадках через надто низьку волатильність чи неправильне округлення цін біржа повертала помилку при створенні стоп-лоссу або тейк-профіту. Це спостерігалося при використанні пар із високим tickSize або нестандартним кроком лота. Проблему частково вирішено через буфер 0.1%, але повністю уникнути її без попередньої перевірки параметрів символу неможливо.

Затримки при великій кількості активних потоків. Під час підключення до 40 і більше WebSocket-каналів помітно зростає навантаження на інтерпретатор Python. У деякі моменти спостерігалася короткочасна затримка в реакції на сигнали, хоча критичних помилок не було. Це обмежує масштабування без переходу на багатопотокову або розподілену архітектуру.

Недостатній захист від дублювання ордерів. Хоча система фіксує символи, по яких уже була відкрита позиція (через змінну printed_symbols), у деяких випадках повторна покупка могла бути дозволена після оновлення ринку. Більш надійний контроль міг би здійснюватися через перевірку статусу позицій із акаунта.

Обрана асинхронна архітектура є простою у реалізації та ефективною для невеликої кількості пар, однак вона має обмеження при масштабуванні:

Усі процеси обробки подій відбуваються в одному циклі подій (asyncio). Це означає, що при надмірному навантаженні можливе блокування черги повідомлень. Альтернативою могло б бути використання розподілених мікросервісів або окремих процесів. Такий підхід порушує принципи безпеки. У реальному середовищі ключі мають зберігатися у захищених змінних середовища або у .env файлі, а також шифруватися для захисту від витоків.

На основі виявлених проблем можна запропонувати кілька напрямів для покращення функціональності системи:

1. Інтеграція логування у файл або базу даних. Збереження кожної дії (час, символ, ціна входу, SL/TP, прибуток/збиток) дозволить побудувати реальну торгову статистику.

2. Telegram-бот для сповіщень. Надсилання повідомлень про відкриті позиції, прибуток, помилки тощо дозволить оперативно контролювати роботу системи без постійного моніторингу терміналу.

3. Гнучка конфігурація параметрів. Можливість змінювати стоп-лосс, тейк-профіт, обсяг угоди, фільтри волатильності та мінімального об’єму – через файл конфігурації або інтерфейс.

4. Модуль backtesting. Реалізація інструменту для перевірки ефективності стратегії на історичних даних (наприклад, через завантаження CSV або використання Binance Historical Data API).

5. Оптимізація обробки потоків. Замість прямої обробки усіх повідомлень можна реалізувати чергу подій з пріоритетами, або відфільтровувати сигнали ще до глибокої обробки.

Незважаючи на початкову простоту системи, її структура дозволяє у майбутньому значно розширити функціонал і адаптуватися до більш складних торгових умов. Одним із напрямів розвитку є впровадження мультирівневої стратегії – замість поточного правила входу на основі цінового імпульсу, можна реалізувати кілька торгових стратегій, що працюють паралельно або комбіновано. Наприклад, можна додати фільтрацію сигналів за допомогою індикаторів технічного аналізу, таких як RSI, MACD, або використовувати аналіз глибини ринку. Це дозволить зробити логіку прийняття рішень більш гнучкою, стійкою до хибних сплесків і чутливішою до ринкових умов.

Крім того, архітектура системи дозволяє додати підтримку інших ринків без суттєвої перебудови основного коду. Наприклад, можна інтегрувати спотовий ринок Binance, який має схожі API-виклики, або додати роботу з іншими криптовалютними біржами, такими як MEXC чи Bybit, які також надають публічні REST/WebSocket API. Це дасть змогу розширити сферу застосування бота та протестувати ефективність стратегії в інших умовах.

Ще одним важливим кроком у розвитку є реалізація графічного інтерфейсу для користувача. Наразі вся взаємодія з ботом відбувається в консольному режимі, що не завжди зручно для аналізу великої кількості угод. Створення веб-панелі на основі Flask або Dash дозволить виводити інформацію у вигляді таблиць, графіків, повідомлень про статус бота, історію угод, динаміку прибутку тощо. Це значно покращить користувацький досвід, особливо для трейдерів, які віддають перевагу візуалізації аналітики.

Також у випадку розширення функціоналу доцільним стане перенесення системи на хмарну інфраструктуру. Це дозволить запускати бота цілодобово,

забезпечити безперервний доступ і контроль, а також уникнути проблем, пов’язаних із вимкненням локального ПК. Для цього можна використовувати VPS, де бот буде розміщений у контейнері Docker, що спрощує його налаштування та деплой. Додатково можна реалізувати систему моніторингу для збору метрик продуктивності та своєчасного виявлення помилок. Захист доступу до системи можна забезпечити через VPN або аутентифікацію користувача на рівні веб-інтерфейсу.

Таким чином, розроблений бот є гнучкою базою для подальшого вдосконалення. Його модульна структура дозволяє додавати нові компоненти без радикальної зміни існуючої логіки, а перспективи масштабування відкривають шлях до створення більш потужної системи автоматизованої торгівлі.

4.4 Висновки до розділу

У четвертому розділі було проведено комплексне тестування розробленої системи автоматичної торгівлі на ф’ючерсному ринку Binance. Основною метою цього етапу стало визначення працездатності, надійності та ефективності функціонування бота в умовах реального ринку, а також виявлення технічних і логічних недоліків, які можуть впливати на стабільність і результативність його роботи.

Проведене функціональне тестування підтвердило, що всі основні компоненти системи – збір ринкових даних, обробка сигналів, відкриття позицій, виставлення ордерів стоп-лосс/тейк-профіт і обробка оновлень акаунту – працюють відповідно до заданої логіки. Бот показав високу швидкодію: середній час реакції на сигнал склав менше 150 мілісекунд, що є достатньо швидким для ф’ючерсного ринку, де кожна секунда може впливати на прибуток. WebSocket-з’єднання з біржею працювали стабільно, без втрат повідомлень, що свідчить про надійність асинхронної архітектури.

У результаті тестових сесій, які охоплювали кілька десятків торгових угод, було зафіксовано високий рівень точності у виставленні ордерів. Водночас були виявлені деякі обмеження: періодичні API-помилки при створенні SL/TP ордерів, залежність від точного округлення параметрів символу, а також відсутність системи збереження історії торгів. Ці недоліки не є критичними, однак їх усунення у наступних версіях дозволить покращити надійність роботи системи.

Були запропоновані ідеї для вдосконалення, зокрема: реалізація мультирівневої торгової стратегії, розширення підтримки на інші ринки, створення веб-інтерфейсу для візуального контролю торгів та перенесення системи в хмарне середовище. Це відкриває перспективи масштабування розробки та переходу від локального торгового помічника до повноцінної модульної платформи.

Загалом, тестування підтвердило, що система може ефективно працювати в реальному середовищі, виконуючи завдання автоматичної торгівлі із високою швидкістю та стабільністю.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи на тему «Створення системи автоматизованої торгівлі криптовалютами за допомогою Binance Api» було виконано повний цикл розробки, який включав аналіз предметної області, обґрунтування архітектурних рішень, реалізацію програмного забезпечення та проведення тестування.

У першому розділі було обґрунтовано актуальність автоматизованої торгівлі в умовах високої волатильності криптовалютного ринку. Було проведено огляд існуючих аналогів – як готових ботів, так і платформ для алгоритмічного трейдингу – що дозволило визначити ключові функціональні вимоги до майбутньої системи. У результаті сформовано постановку задачі, яка передбачала створення простої, швидкодіючої та гнучкої системи з підтримкою WebSocket-з’єднань і REST API.

У другому розділі проаналізовано та обґрунтовано вибір інструментів технічної реалізації. Перевагу було віддано Python як основній мові розробки, бібліотеці python-binance для інтеграції з API Binance, а також асинхронному підходу на основі asyncio. Побудовано логічну структуру системи з поділом на окремі модулі: обробка ринкових сигналів, прийняття торгових рішень, виконання ордерів, логування та обробка змін акаунта. Визначено архітектурну модель, що базується на подієво-орієнтованому управлінні потоком даних, що забезпечує високу адаптивність до ринкових змін.

У третьому розділі реалізовано всі основні компоненти системи. Було детально описано логіку роботи WebSocket-потоків для отримання ринкових даних у режимі реального часу, механізм відкриття позицій за ринком та виставлення захисних ордерів (SL/TP). Продемонстровано, як здійснюється збереження внутрішнього стану, обробка API-виключень та перепідключення при збої з’єднання. Розроблено інструкцію із запуску системи в локальному середовищі, зокрема підключення до Binance API через користувацькі ключі, встановлення бібліотек та запуск у

середовищі Python. Це робить систему доступною для тестування й експлуатації на будь-якому комп’ютері без потреби в розгортанні на сервері.

У четвертому розділі проведено тестування працездатності системи. Встановлено, що час реакції на сигнали становить менше 200 мс, що є задовільним показником для ф’ючерсного ринку. Усі основні дії бота, включно з відкриттям позицій і виставленням стоп-ордерів, виконувалися коректно. Водночас було виявлено певні обмеження – наприклад, відсутність журналу історичних угод та залежність від точності округлення параметрів символів. На основі результатів тестування запропоновано шляхи вдосконалення: реалізація багаторівневої стратегії, створення веб-інтерфейсу, інтеграція Telegram-сповіщень, використання хмарної інфраструктури для постійної роботи бота.

Таким чином, у бакалаврській кваліфікаційній роботі досягнуто поставленої мети: створено функціональну, асинхронну систему для автоматизованої торгівлі криптовалютами через Binance API. Результати тестування підтвердили її практичну цінність, а запропоновані напрямки модернізації демонструють потенціал для масштабування та подальшого розвитку. Отримані результати можуть бути використані як основа для створення більш складної, розширеної системи алгоритмічного трейдингу або як навчальний приклад сучасної інтеграції із криптобіржами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Створення системи автоматизованої торгівлі криптовалютами за допомогою Binance Api / Моїк І.І., Кабачій В.В. // Матеріали Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» 2025», Вінниця, 2025. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/24257 (дата звернення 15.05.2025).

2. Binance Academy. What Is a Trading API and Is It Worth It for Traders? URL: https://academy.binance.com/en/articles/what-is-a-trading-api-and-is-it-worth-it-for-traders (дата звернення 20.05.2025).

3. Binance Blog. A Guide to Automated Crypto Trading. URL: https://www.binance.com/en/blog/futures/binance-trading-bots-a-guide-to-automated-crypto-trading-4987930405804222569 (дата звернення 20.05.2025).

4. Мартін Р. К. Чистий код: створення, аналіз та рефакторинг. Київ : Наш Формат, 2020. 432 с. ISBN 978-617-7682-34-5.

5. Таненбаум Е., Бос Г. Сучасні операційні системи. 4-те вид. Київ : Пітер, 2017. 1136 с. ISBN 978-5-496-01764-3.

6. McConnell S. Code Complete. 2nd Edition. Microsoft Press, 2004. 960 p. ISBN 978-0735619678.

7. Turnbull J. The Docker Book: Containerization is the new virtualization. 2021. 348 p. ISBN 978-0988820280.

8. Pressman R. S. Software Engineering: A Practitioner’s Approach. 8th ed. McGraw-Hill, 2014. 928 p. ISBN 978-0078022126.

9. Sommerville I. Software Engineering. 9th ed. Pearson Education, 2011. 792 p.

10. IEEE Spectrum. Top Programming Languages 2023. URL: https://spectrum.ieee.org/top-programming-languages-2023 (дата звернення: 23.05.2025).

11. GitHub Repository – python-binance. URL: https://github.com/sammchardy/python-binance (дата звернення: 23.05.2025).

12. Python documentation. asyncio – Asynchronous I/O. URL: https://docs.python.org/3/library/asyncio.html (дата звернення: 23.05.2025).

13. PyPI. dotenv library. URL: https://pypi.org/project/python-dotenv/ (дата звернення: 23.05.2025).

14. Binance API Documentation. Futures API. URL: https://binance-docs.github.io/apidocs/futures (дата звернення: 23.05.2025).

15. Python documentation. Logging HOWTO. URL: https://docs.python.org/3/howto/logging.html (дата звернення: 23.05.2025).

16. Géron A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 2nd Edition. O'Reilly Media, 2019. – 822 p.

17. python-binance GitHub repository. URL: https://github.com/sammchardy/python-binance (дата звернення: 23.05.2025)

18. Binance API Documentation – https://binance-docs.github.io/apidocs/futures/en/ (дата звернення: 23.05.2025)

19. Binance Academy – What Is a Trading API and Is It Worth It for Traders? URL: https://academy.binance.com/en/articles/what-is-a-trading-api-and-is-it-worth-it-for-traders

20. Binance Blog – Binance Trading Bots: A Guide to Automated Crypto Trading. URL: https://www.binance.com/en/blog/futures/binance-trading-bots-a-guide-to-automated-crypto-trading-4987930405804222569

21. Sammchardy. python-binance GitHub. URL: https://github.com/sammchardy/python-binance (дата звернення: 25.05.2025).

22. Géron A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 2nd Edition. – O'Reilly Media, 2019. – 822 p. – ISBN 978-1492032649.

23. REST API Tutorial. URL: https://restfulapi.net/ (дата звернення: 30.05.2025).

24. Turnbull J. *The Docker Book: Containerization is the new virtualization*. – 2021. – 348 p. – ISBN 978-0988820280.

25. Pressman R. S. *Software Engineering: A Practitioner's Approach*. 8th ed. McGraw-Hill, 2014.

Додаток А

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

СТВОРЕННЯ СИСТЕМИ АВТОМАТИЗОВАНОЇ ТОРГІВЛІ КРИПТОВАЛЮТАМИ ЗА ДОПОМОГО042Е BINANCE API

Алгоритм роботи бота

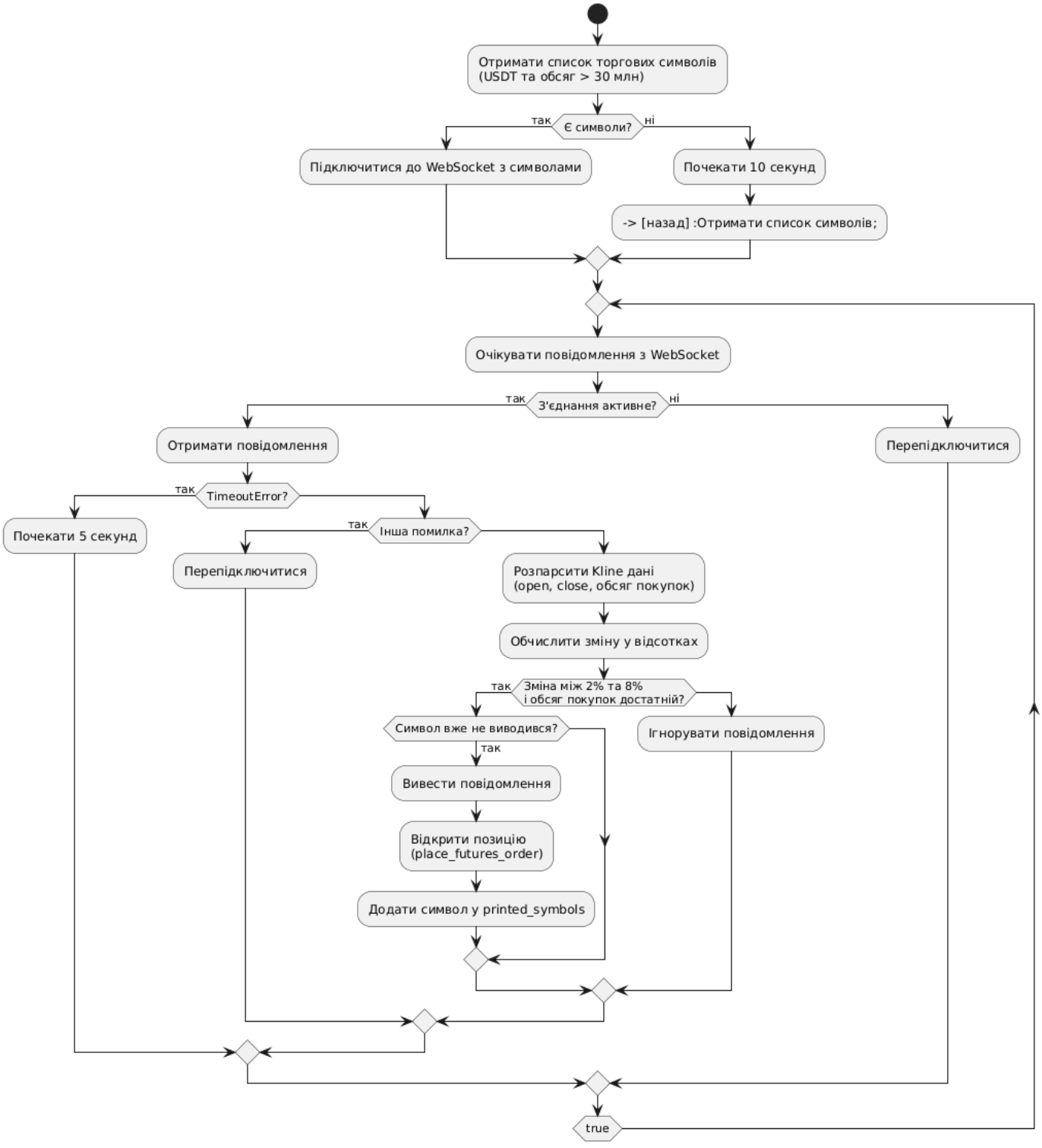


Рисунок А.1 – Алгоритм роботи торгового бота

Алгоритм виставлення захисних ордерів

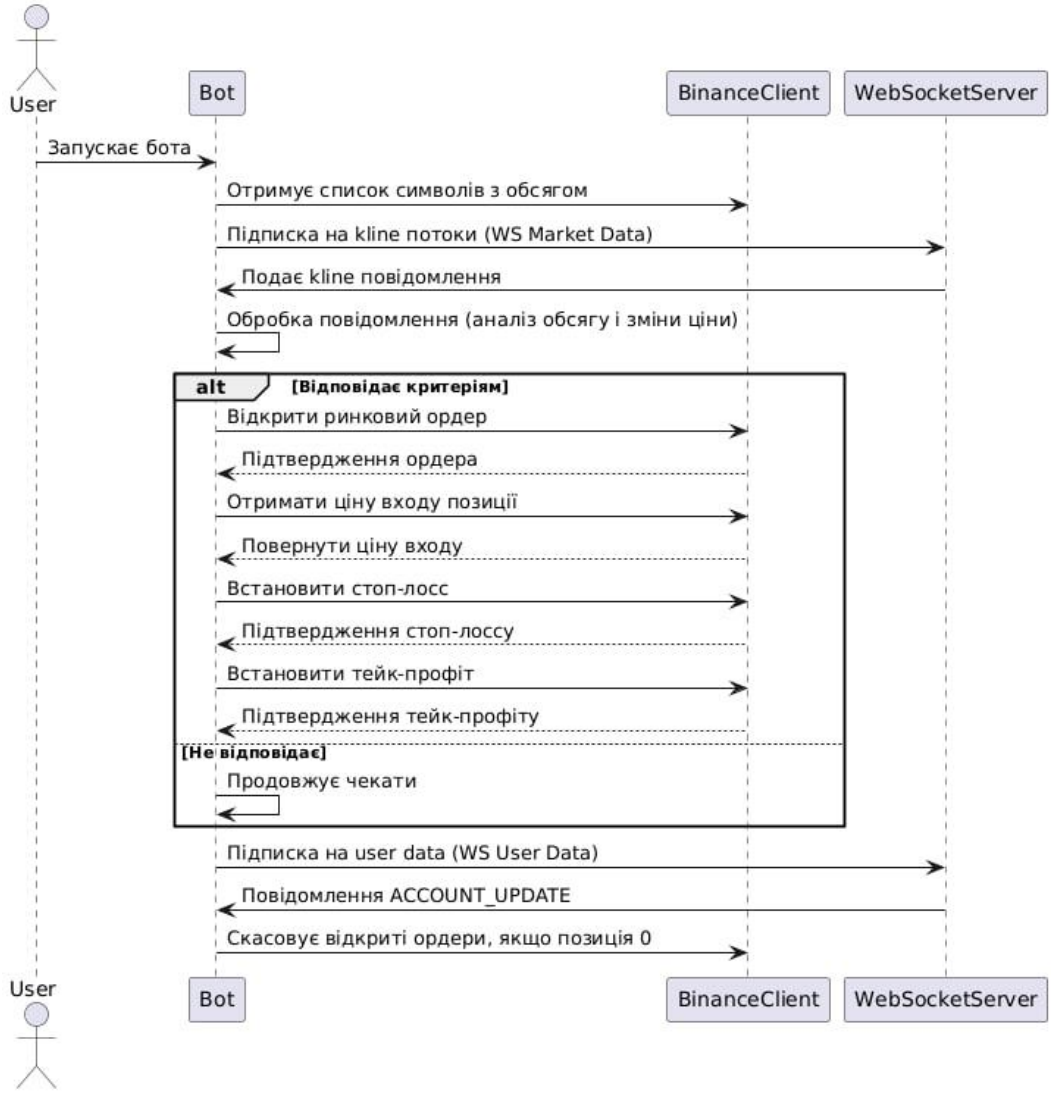


Рисунок А.2 – UML-діаграма послідовностей для Rest-запиту на виставлення захисних ордерів

Алгорим моніторингу ринку

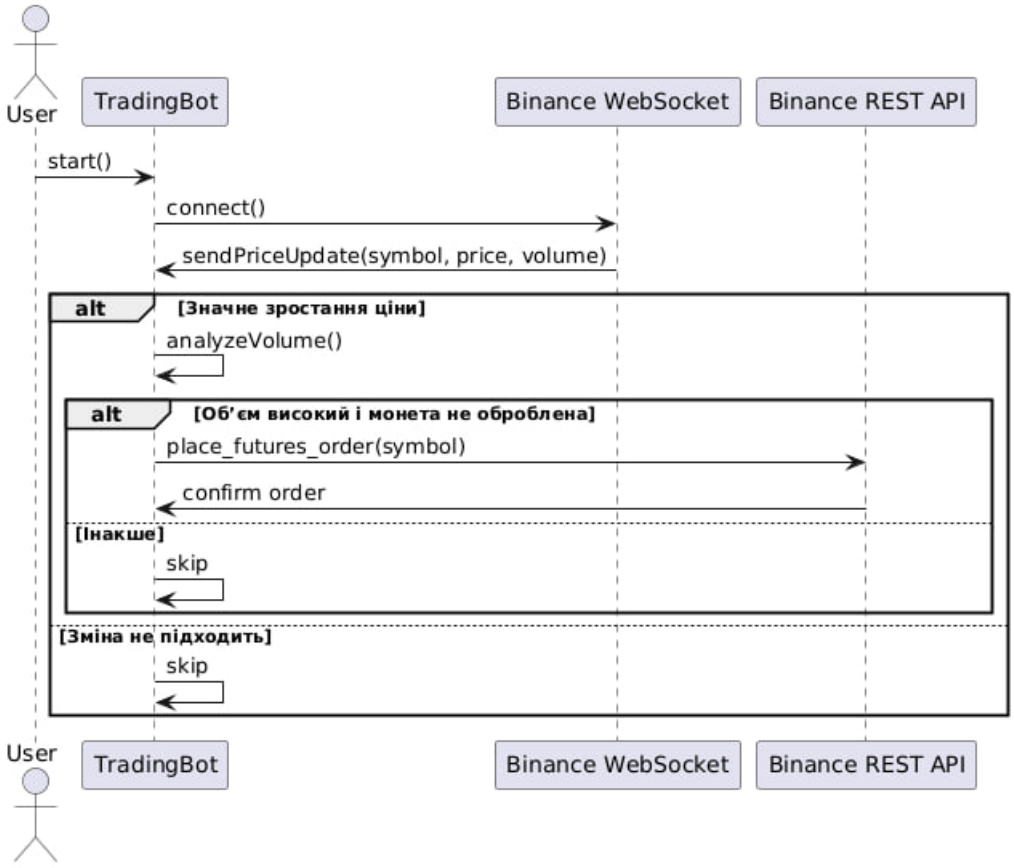


Рисунок А.3 – UML діаграма алгоритму моніторингу ринку та відкриття угод

Обробка сигналу та виставлення торгових ордерів

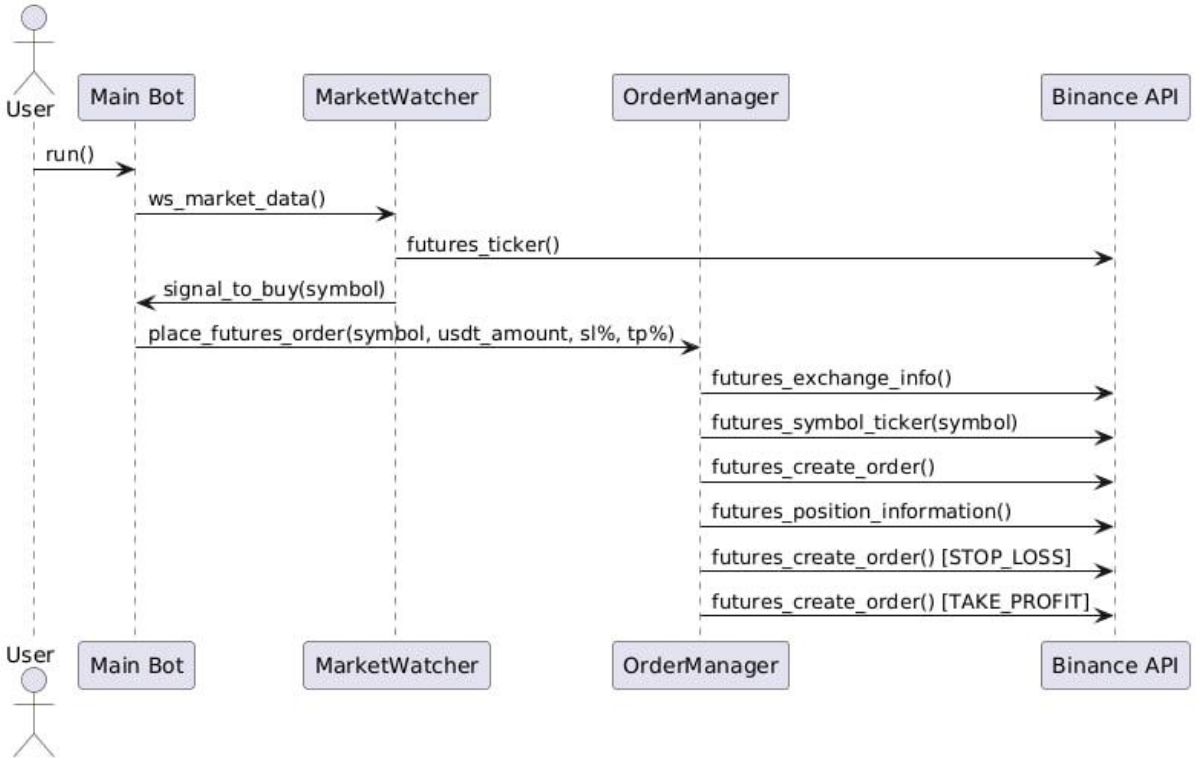


Рисунок А.4 – UML діаграма обробки сигналу та виставлення торгових ордерів

Діаграма класів

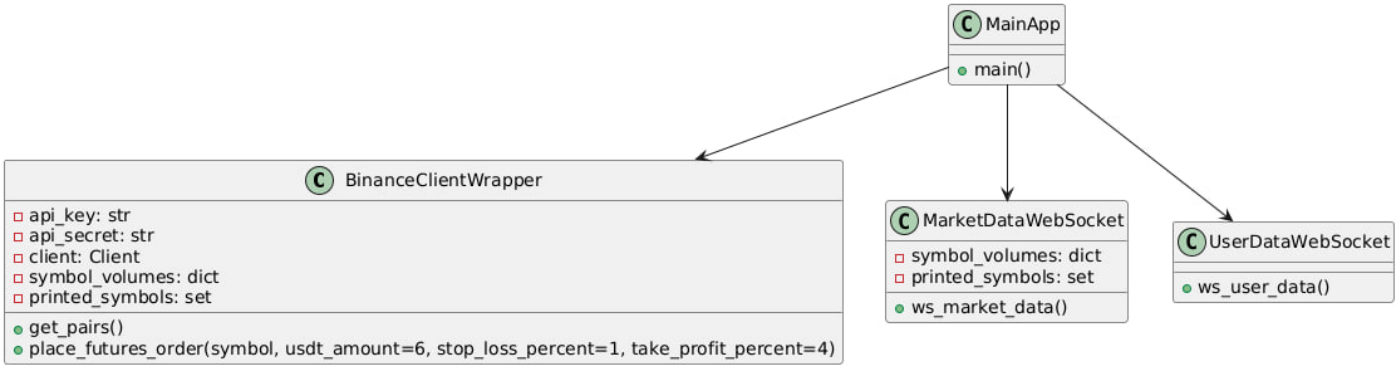


Рисунок А.5 – UML-схема класів

Додаток Б

(обов’язковий)

Лістинг основних функцій програми

```
def get_pairs():
    global symbol_volumes
    info = client.futures_ticker()
    for item in info:
        symbol = item['symbol']
        if not symbol.endswith("USDT"):
            continue
        volume = float(item['quoteVolume'])
        if volume > 30 * 10 ** 6:
            symbol_volumes[symbol] = round(volume / 288, 5)
    df = pd.DataFrame(list(symbol_volumes.items()), columns=['symbol', 'volume'])
    symbol_volumes = df.set_index('symbol')['volume'].to_dict()
    return df

def round_step_size(value, step):
    return math.floor(value / step) * step

def place_futures_order(symbol, usdt_amount=6, stop_loss_percent=1,
take_profit_percent=4):
    print(f"\n🟢 Купівля ф'ючерсу: {symbol}")
    try:
        info = client.futures_exchange_info()
        filters = next(s for s in info['symbols'] if s['symbol'] == symbol)['filters']
```

```
    tick_size = float(next(f for f in filters if f['filterType'] ==
'PRICE_FILTER')['tickSize'])
    step_size = float(next(f for f in filters if f['filterType'] == 'LOT_SIZE')['stepSize'])

    ticker = client.futures_symbol_ticker(symbol=symbol)
    price = float(ticker['price'])

    quantity = round_step_size(usdt_amount / price, step_size)
    if quantity == 0:
        print("❗ Недостатньо коштів для відкриття позиції")
        return

    # Ринковий ордер
    order = client.futures_create_order(
        symbol=symbol,
        side=SIDE_BUY,
        type=FUTURE_ORDER_TYPE_MARKET,
        quantity=quantity
    )
    print(f"✅ Куплено {quantity} {symbol} @ {price:.4f}")

    # Чекаємо кілька мілісекунд щоб позиція оновилась на сервері
    time.sleep(0.5)

    # Отримуємо реальну ціну входу з position info
    positions = client.futures_position_information(symbol=symbol)
    entry_price = float(next(p for p in positions if p['symbol'] == symbol)['entryPrice'])
```

```
if entry_price == 0:
    print("❌ Не вдалося отримати реальну ціну входу")
    return

# Розрахунок стоп-лосс та тейк-профіт
buffer = 0.001

stop_price_raw = entry_price * (1 - stop_loss_percent / 100)
take_profit_price_raw = entry_price * (1 + take_profit_percent / 100)

stop_price = round_step_size(stop_price_raw * (1 - buffer), tick_size)
take_profit_price = round_step_size(take_profit_price_raw * (1 + buffer), tick_size)

if stop_price >= entry_price:
    stop_price = round_step_size(entry_price * (1 - buffer), tick_size)
if take_profit_price <= entry_price:
    take_profit_price = round_step_size(entry_price * (1 + buffer), tick_size)

# Стоп-лосс
try:
    client.futures_create_order(
        symbol=symbol,
        side=SIDE_SELL,
        type=FUTURE_ORDER_TYPE_STOP_MARKET,
        stopPrice=f"{stop_price:.8f}",
        closePosition=True,
```

```
            timeInForce=TIME_IN_FORCE_GTC
        )
        print(f"🛑 Стоп-лосс встановлено @ {stop_price:.4f}")
    except BinanceAPIException as e:
        print(f"❌ Не вдалося встановити стоп-лосс: {e.message}")

    # Тейк-профіт
    try:
        client.futures_create_order(
            symbol=symbol,
            side=SIDE_SELL,
            type=FUTURE_ORDER_TYPE_TAKE_PROFIT_MARKET,
            stopPrice=f"{take_profit_price:.8f}",
            closePosition=True,
            timeInForce=TIME_IN_FORCE_GTC
        )
        print(f"🎯 Тейк-профіт встановлено @ {take_profit_price:.4f}")
    except BinanceAPIException as e:
        print(f"❌ Не вдалося встановити тейк-профіт: {e.message}")

except BinanceAPIException as e:
    print(f"❌ Binance API помилка: {e.message}")
except Exception as e:
    print(f"❌ Помилка: {e}")

async def ws_market_data():
    while True:
```

```
try:
    streams = [f"{symbol.lower()}@kline_5m" for symbol in symbol_volumes]
    if not streams:
        print("⏳ Очікуємо символи для підписки...")
        await asyncio.sleep(10)
        continue
    url = f"wss://fstream.binance.com/stream?streams={'/'.join(streams)}"
    print(f"🔌 Підключення до WebSocket: {url}")
    async with websockets.connect(url, ping_interval=10) as websocket:
        while True:
            try:
                message = await asyncio.wait_for(websocket.recv(), timeout=30)
                data = json.loads(message)
                kline = data.get('data', {}).get('k', {})
                symbol = kline.get('s', None)
                if not symbol:
                    continue

                open_price = float(kline.get('o', 0))
                close_price = float(kline.get('c', 0))
                volume_buyer = float(kline.get('Q', 0))

                if open_price > 0:
                    change_percent = abs((close_price - open_price) / open_price) * 100
                else:
                    change_percent = 0
```

```
                volume_ticker = float(symbol_volumes.get(symbol, 0))

                if close_price > open_price and 2 <= change_percent <= 8:
                    result_buying = int(volume_buyer / volume_ticker)
                    if result_buying >= 2:
                        if symbol not in printed_symbols:
                            print(f"Symbol: {symbol}, Result Buying: {result_buying},
Change%: {change_percent:.2f}")
                            printed_symbols.add(symbol)
                            place_futures_order(symbol, 20)

            except asyncio.TimeoutError:
                print("Timeout error, reconnecting...")
                break
        except Exception as e:
            print(f"Connection error: {e}, retrying in 5 seconds...")
            await asyncio.sleep(5)

async def ws_user_data():
    listen_key = client.futures_stream_get_listen_key()
    print(f"User data listenKey отримано: {listen_key}")

    async def keepalive():
        while True:
            try:
                client.futures_stream_keepalive(listen_key)
            except Exception as e:
```

```
            print(f"Помилка keepalive: {e}")
        await asyncio.sleep(60 * 30)  # кожні 30 хвилин

    async with websockets.connect(f"wss://fstream.binance.com/ws/{listen_key}") as
websocket:
        asyncio.create_task(keepalive())

        while True:
            try:
                msg = await websocket.recv()
                data = json.loads(msg)
                event_type = data.get('e')

                if event_type == 'ACCOUNT_UPDATE':
                    positions = data.get('a', {}).get('P', [])
                    for pos in positions:
                        symbol = pos['s']
                        position_amt = float(pos['pa'])
                        if position_amt == 0:
                            try:
                                client.futures_cancel_all_open_orders(symbol=symbol)
                                print(f"Відкриті ордери для {symbol} скасовані, бо позиція
закрита")
                            except Exception as e:
                                print(f"Помилка скасування ордерів {symbol}: {e}")

            except Exception as e:
```

```
            print(f"Помилка user data websocket: {e}")
            await asyncio.sleep(5)

async def main():
    get_pairs()
    task_market = asyncio.create_task(ws_market_data())
    task_user = asyncio.create_task(ws_user_data())
    await asyncio.gather(task_market, task_user)
```

77

Додаток Г (обов'язковий) ПРОТОКОЛ ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Створення системи автоматизованої торгівлі криптовалютами за допомогою Binance Api

Тип роботи: Бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра АІІТ, ФІІТА, гр. 1ІСТ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0.64 %

- Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Олег БІСІКАЛО, зав. кафедри АІІТ _____
(прізвище, ініціали, посада) (підпис)

Любов ЯНЧУК, секретар кафедри АІІТ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Костянтин ОВЧИННИКОВ
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ к.т.н., Кабачій Владислав Володимирович
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Моїк Ігор Іванович
(підпис) (прізвище, ініціали)