Вищий навчальний заклад

«Університет економіки та права «КРОК»
Фаховий коледж

Циклова комісія з інформаційних технологій

Кваліфікаційна робота фахового молодшого бакалавра

на тему Створення месенджер-боту для пошуку учасників та алгоритмів на криптобіржах

Виконав _____
(Підпис)

__Опанасенко_Максим_Володимирович__
(прізвище, ім'я, по батькові)

_____

Науковий керівник
____Кириченко_Віктор_Вікторович__
(прізвище, ім'я, по батькові)

_____

_____
(Резолюція «До захисту»)

Попередній захист:

_____
(Висновок: "До захисту в екзаменаційній комісії ")

Голова циклової комісії _____ _____ _____
(Підпис) (Прізвище, ініціали) (Дата)

Київ – 2025 року

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

Фаховий коледж

Циклова комісія з інформаційних технологій

Спеціальність 121 інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Голова циклової комісії _____ Леонід УВАРОВ

(підпис)

«____» _____ 2025 року

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувач освіти: **Опанасенко Максим Володимирович**

1. Тема роботи: Створення месенджер-боту для пошуку учасників та алгоритмів на криптобіржах, затверджена наказом по університету
   від «___» _____ 202__ р. № _____
2. Термін здачі закінченої роботи: «30» травня 2025 р
3. Вихідні дані до роботи :
   1) Цільова аудиторія – користувачі, зацікавлені в оперативному отриманні фінансової інформації: приватні інвестори, трейдери, аналітики, а також ті, хто стежить за ринками задля особистих цілей.

   2) Функціональність – надсилання повідомлень про досягнення встановленого рівня ціни фінансовими активами (наприклад, криптовалютами), створення й керування умовами моніторингу, зберігання активних запитів користувача.

   3) Технічні вимоги – реалізація на базі платформи для обміну повідомленнями, застосування мови програмування з підтримкою API-запитів до фінансових джерел (наприклад, Python), використання бази даних для зберігання умов сповіщень та налаштувань.

   4) Джерела даних – зовнішні API-сервіси, які надають інформацію про ціни активів у реальному часі

   5) Можливості масштабування – підтримка розширення функціональності у майбутньому, включаючи додавання нових типів активів, нових умов сповіщень, а також інтеграцію з іншими сервісами.
4. Зміст пояснювальної записки
   Розділ 1. Теоретична частина – огляд існуючих цифрових інструментів для фінансового аналізу та ботів у месенджерах, переваги автоматизованого скринінгу ринкових даних,

обґрунтування вибору мови програмування, платформи для обміну повідомленнями та сторонніх сервісів для отримання фінансової інформації.

Розділ 2. Проєктування та розробка – створення інформаційної системи для пошуку учасників та алгоритмів на криптобіржах з логікою спілкування з користувачем через повідомлення, розробка механізму обробки запитів, генерації відповідей і динамічного формування списків акцій, інтерфейсна структура та зберігання налаштувань.

Розділ 3. Експериментальна частина. Проведення тестування роботи бота на різних пристроях. Перевірка основних сценаріїв: додавання умов для сповіщень, отримання повідомлень, керування активними запитами. Проаналізовано, наскільки бот відповідає поставленим цілям.

5. Перелік графічного матеріалу
   1) Скріншоти інтерфейсу бота
   2) Приклади повідомлень з результатами перевірки умов
   3) Таблиці з умовами, які користувачі можуть задати для моніторингу цін
   4) Проста схема з’єднання між компонентами системи (бот – база даних – джерело фінансових даних)
   5) Ілюстрації, як користувач задає умови для сповіщення та як виглядає повідомлення при їх виконанні

Дата видачі завдання 12 лютого 2025 року
Науковий керівник

_____ Кириченко Віктор Вікторович
(підпис) (прізвище, ім'я, по батькові)

Завдання прийняв до виконання _____ Опанасенко Максим Володимирович
(підпис) (прізвище, ім'я, по батькові)

РЕФЕРАТ

Пояснювальна записка: 54 сторінок, 4 рисунки, 3 таблиці, 2 додатків, 15 джерел.

Об'єкт дослідження – програмне забезпечення для обробки та аналізу даних з публічних API криптовалютних бірж.

Мета роботи – розробка чат-боту для моніторингу криптовалютних активів, пошуку торгових алгоритмів і учасників на криптобіржах шляхом аналізу API-даних.

У кваліфікаційній роботі представлено результати розробки програмного забезпечення чат-боту, що забезпечує збір, обробку та фільтрацію даних з криптовалютних бірж. Проведено аналіз предметної області криптовалютної торгівлі, визначено ключові вимоги до функціональності чат-боту. Побудовано моделі потоків даних (DFD-діаграми) із застосуванням CASE-засобу розробки інформаційних систем.

Розроблено структуру бази даних для збереження інформації про моніторинг активів, користувацькі запити та алгоритми. Побудовано логічну та фізичну модель даних за допомогою CASE-засобів. Програмна реалізація здійснена з використанням мови Python та бібліотеки для створення ботів у месенджерах. Для інтеграції з біржами використано публічні REST API Binance, Bybit та OKX.

Запропоновано механізм створення повідомлень на основі різких змін ринку та досягнення цінових порогів, заданих користувачами. Реалізовано можливість фільтрації активів за різними параметрами (зростання ціни, обсяги торгів, спільна динаміка на кількох біржах тощо).

Результати роботи можуть бути впроваджені у сферах автоматизованого моніторингу ринку криптовалют, трейдингу, арбітражу та фінансової аналітики.

Ключові слова: ЧАТ-БОТ, КРИПТОВАЛЮТНА БІРЖА, АРБІТРАЖ, API, ІНФОРМАЦІЙНА СИСТЕМА, ОБРОБКА ДАНИХ, ПРОГРАМНА ІНЖЕНЕРІЯ.

ABSTRACT

Explanatory note: 54 pages, 4 figures, 3 tables, 2 appendices, 15 sources.

The object of the study is software for processing and analyzing data from public APIs of cryptocurrency exchanges.

The purpose of the work is to develop a messenger bot for monitoring cryptocurrency assets, searching for trading algorithms and participants on exchanges using API data analysis.

The qualification paper presents the results of the development of a messenger bot software system that performs data collection, processing, and filtering from cryptocurrency exchanges. The subject area of cryptocurrency trading is analyzed, and key functional requirements for the bot are defined. Data flow models (DFD diagrams) were developed using CASE tools.

A database structure was designed to store information about asset monitoring, user queries, and algorithmic strategies. Logical and physical data models were developed using CASE tools. The software solution was implemented using Python and a messenger bot framework. Integration with exchanges was carried out using public REST APIs of Binance, Bybit, and OKX.

An alert mechanism was proposed based on rapid market movements and user-defined price thresholds. Functionality for filtering assets by parameters such as price growth, trading volumes, and cross-exchange momentum was implemented.

The results can be applied in the areas of automated crypto market monitoring, trading, arbitrage, and financial analytics.

Keywords: CHATBOT, CRYPTOCURRENCY EXCHANGE, ARBITRAGE, API, INFORMATION SYSTEM, DATA PROCESSING, SOFTWARE ENGINEERING.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

Chat-бот – Chat-бот, який використовується для взаємодії з користувачами.

API – Інтерфейс програмування додатків; засіб обміну даними між системами.

USDT – Tether; стабільна криптовалюта (стейблкоїн), прив’язана до долара США.

BTC, ETH тощо – Умовні позначення криптовалют (Bitcoin, Ethereum).

Binance, OKX, Bybit – Популярні централізовані криптовалютні біржі.

Пінг API – Регулярний запит до зовнішнього API для отримання оновлених даних.

Користувач – Особа, яка взаємодіє з Chat-ботом.

Скрінінг – Автоматичне відстеження фінансових активів на основі заданих умов.

Алерт (сповіщення) – Повідомлення користувача при виконанні певної умови (наприклад, досягнення ціни).

Умова моніторингу – Задана користувачем логіка перевірки стану активу (наприклад: BTC > 70,000).

БД – База даних для зберігання користувацьких налаштувань та активних умов.

Фінансовий актив – Криптовалюта або інший інструмент, що підлягає моніторингу.

JSON – Формат даних, який часто використовується для обміну інформацією через API.

Моніторинг – Перевірка змін параметрів активів у режимі реального часу.

Ринок (Market) – Пара валют (наприклад, BTC/USDT) на біржі, яка має поточну ціну.

Webhook – Механізм обробки подій через вхідні запити, який іноді використовується в ботах.

Запит (Request) – Звернення до сервісу API з метою отримання даних.

Реал-тайм – Робота з даними у режимі реального часу.

Трейдер – Користувач, який приймає торгові рішення на основі інформації, наданої ботом.

Система алертів – Компонент бота, що відповідає за перевірку умов і надсилання сповіщень.

ВСТУП

Актуальність завдання. У сучасному цифровому середовищі розвиток фінансових технологій та криптовалют відкриває нові можливості для інвесторів, трейдерів і розробників. Зростання обсягів торгівлі на криптобіржах, поява нових торгових стратегій та алгоритмічних моделей призводять до необхідності автоматизованих інструментів для збору, аналізу та представлення даних у реальному часі.

Оперативний доступ до інформації про активність трейдерів, стратегії, що використовуються на біржах, та динаміку цін є важливим фактором для прийняття ефективних торгових рішень. У цьому контексті месенджер-боти виступають як зручний та гнучкий інтерфейс взаємодії користувача з аналітичною системою, дозволяючи забезпечити швидке отримання релевантних даних із різних біржових платформ.

Розробка чат-бота, що виконує пошук активних учасників та торгових алгоритмів на криптобіржах, є актуальним завданням, яке дозволяє підвищити прозорість ринку, автоматизувати скринінг даних та забезпечити швидку реакцію на ринкові зміни.

Мета роботи. Розробити програмний засіб — месенджер-бот — для автоматизованого пошуку учасників, торгових стратегій і аномальної активності на криптовалютних біржах на основі відкритих біржових API.

Завдання роботи.

- Проаналізувати існуючі програмні рішення у сфері моніторингу криптобірж та визначити їхні переваги й обмеження;
- Розробити архітектуру системи, включаючи таблиці даних і логіку обробки запитів користувачів;
- Реалізувати механізм отримання та обробки даних з криптобірж (зокрема, OKX, Bybit, Binance) через їхні API;
- Реалізувати систему сповіщень для інформування про появу нових алгоритмів чи значних рухів цін;

- Провести тестування і налагодження системи для забезпечення її стабільної роботи в реальному режимі.

Об’єкт дослідження. Процес автоматизованого збору, фільтрації та представлення даних з криптовалютних бірж.

Предмет дослідження. Функціональні можливості чат-бота для моніторингу активності на криптобіржах та взаємодії з користувачем через інтерфейс месенджера.

Методи дослідження.У дипломній роботі використано такі методи:

- аналіз та узагальнення існуючих підходів до моніторингу біржової активності;
- проєктування клієнт-серверної архітектури для інтеграції з API криптобірж;
- розробка бот-інтерфейсу на мові Python із використанням відповідної бібліотеки для створення ботів у месенджерах;
- моделювання бази даних для зберігання біржових даних та інформації про користувачів;
- тестування та оптимізація роботи системи в умовах реального інформаційного навантаження.

Практичне значення одержаних результатів. Розроблений месенджер-бот може бути використаний трейдерами, аналітиками та розробниками для ефективного моніторингу активності на криптобіржах, оперативного реагування на ринкові події та вдосконалення власних алгоритмічних стратегій. Це сприяє автоматизації аналізу ринку та підвищенню точності прийняття рішень у сфері криптовалютної торгівлі.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖЕННЯ

1.1. Аналіз сучасних цифрових інструментів для фінансового моніторингу

В умовах сьогодення, коли цифрова економіка стрімко розвивається, особливої ваги набувають інструменти, які дають змогу автоматично проводити фінансовий моніторинг, аналізувати ринок та виявляти перспективи для торгівлі в криптосфері. Серед найбільш перспективних напрямків у цій області виділяється застосування чат-ботів та автоматизованих систем, що інтегруються з криптовалютними біржами за допомогою публічних API.

Наявні рішення для фінансового моніторингу можливо умовно розділити на категорії відповідно до їх основних функціональних задач.

Боти для моніторингу цін на криптовалюти, такі як Cscalpbot, Linerscreaner чи Digashscalp, дають змогу користувачам відстежувати поточні котирування, коливання цін за добу, тиждень або місяць, а також налаштовувати сповіщення при досягненні певних порогових значень. Такі боти є зручними для швидкого отримання інформації з бірж без потреби відвідувати спеціалізовані веб-сайти.

Аналітичні боти та сервіси, зокрема CryptoHopper, 3Commas та Token Metrics, надають більш глибокий фінансовий аналіз. Вони взаємодіють з біржами, проводять скринінг ринку, створюють графіки, аналізують торгові обсяги, індикатори та генерують сигнали для купівлі чи продажу активів. Нерідко такі сервіси містять AI-модулі або машинне навчання для передбачення ринкових тенденцій.

Боти для спостереження за великими транзакціями та діями "китів" (великих інвесторів), наприклад, Whale Alert, моніторять блокчейн-мережі на предмет підозріло великих переказів між гаманцями або біржами. Це дозволяє трейдерам оперативно реагувати на ймовірні зміни ринку, спричинені великими гравцями.

Системи моніторингу торгових стратегій або так звані "стратегічні сканери" (наприклад, LuxAlgo, TradingView bots) дозволяють користувачам відслідковувати виконання різноманітних торгових алгоритмів на основі

технічного аналізу, графічних патернів або заданих умов. Часто вони інтегровані з месенджерами або браузерними розширеннями для зручної доставки сповіщень.

Разом із тим, аналіз існуючих рішень дозволяє виокремити низку обмежень, притаманних сучасним інструментам фінансового моніторингу:

- **обмежена кастомізація**: багато ботів не дозволяють користувачеві створювати власні умови фільтрації або адаптувати сповіщення під специфічні потреби;
- **висока вартість підписки**: професійні функції або доступ до алгоритмів часто потребують платної підписки, що обмежує доступ до сервісів для початківців;
- **низький рівень автоматизації пошуку учасників і стратегій**: більшість ботів орієнтовані на моніторинг цін і обсягів, проте не надають можливостей для аналізу активності конкретних трейдерів або виявлення унікальних алгоритмів;
- **ризики конфіденційності**: при інтеграції з біржовими акаунтами через API користувач повинен надати ключі доступу, що потребує особливої уваги до безпеки даних.

Таким чином, наявні рішення дозволяють ефективно вирішувати типові задачі моніторингу крипторинку, однак потребують доповнення інструментами для аналізу алгоритмічної торгівлі, виявлення активних гравців та побудови гнучких пошукових систем у реальному часі. Розробка власного месенджер-бота, адаптованого під конкретні завдання трейдингового скринінгу, є актуальним напрямом удосконалення цифрових засобів моніторингу.

1.2. Боти у месенджерах: огляд можливостей та прикладів використання

Розробка ботів для месенджерів розкриває великий потенціал для автоматизації комунікаційних, інформаційних та аналітичних процедур у фінансовій галузі. Через простоту використання, доступність і інтеграцію з відомими платформами, ці боти являють собою ефективний інструмент для втілення систем моніторингу, сповіщень, управління даними та виконання запитів користувачів в реальному часі.

У розробці чат-ботів для фінансового моніторингу, першочергову увагу слід приділити наступному:

Захист чутливої інформації. Фінансові відомості, серед яких API-ключі до рахунків на біржах, дані про активи, сповіщення про великі операції чи стратегії користувача, є конфіденційними та потребують шифрування при зберіганні та передачі.

Інтуїтивний та адаптивний інтерфейс користувача. Боти мають мати просту навігацію з використанням кнопок, меню або структурованих команд. Це полегшує початок роботи для нових користувачів та підвищує ефективність взаємодії. Скажімо, користувач повинен мати змогу без зусиль додати монету до списку спостереження або встановити ціновий алерт у декілька кліків.

Інтеграція з біржами та зовнішніми API. Ефективний моніторинговий бот повинен мати надійне підключення до бірж, таких як Binance, OKX, Bybit, а також до агрегаторів даних (наприклад, CoinGecko або CoinMarketCap). Це дозволяє отримувати актуальну інформацію в режимі реального часу, виконувати запити щодо обсягу торгів, змін ціни та волатильності.

Функціонал аналітики та сповіщень. Боти повинні не просто відображати дані, а й надавати аналітичну інформацію: виявлення різких цінових змін, одночасне зростання вартості активів на різних біржах, моніторинг активності великих гравців. Система сповіщень має бути гнучкою — з можливістю налаштування порогів, таймерів, шаблонів повідомлень.

Масштабованість та персоналізація. Інфраструктура бота повинна підтримувати масштабування — як за кількістю користувачів, так і за кількістю активів, бірж, стратегій моніторингу. Крім того, важливо надати користувачам можливість створювати власні фільтри, теги та пошукові запити — наприклад, відстеження лише волатильних монет або тільки нових лістингів.

Приклади реалізованих рішень:

- **CryptoWhaleBot** — надсилає сповіщення про великі транзакції між біржами та гаманцями;
- **Aalsttrade** — дозволяє налаштувати торгові сигнали, які автоматично виконують дії на біржах;

- **OrcalistingBot** — інтегрований із децентралізованими платформами та моніторить нові лістинги токенів;
- **SireyanTeamBot** — аналізує згадки монет у соціальних мережах для формування трендових списків.

Таким чином, чат-боти у месенджерах виступають не лише як засіб інформування, а як повноцінний інструмент аналітики, пошуку та супроводу фінансових рішень. Їхня ефективність залежить від гнучкості налаштувань, надійності джерел даних і безпечної інтеграції з користувацькими акаунтами.

1.3 Обґрунтування вибору технологій: мова Python, API, криптобіржи Binance

Для втілення чат-бота, що шукає фінансові алгоритми й учасників на криптовалютних платформах, було прийнято рішення застосувати певні технологічні підходи, які гарантують гнучкість, ефективність та легкість інтеграції. Головний акцент робився на зручності взаємодії з API бірж, швидкості розробки та можливості масштабування системи.

Вибір мови програмування: Python

Python є однією з найпопулярніших мов програмування у сфері фінансових технологій завдяки таким перевагам:

Широкий набір бібліотек для роботи з API (requests, aiohttp, websockets), що дозволяє ефективно здійснювати обробку запитів до криптобірж у реальному часі.

Простота синтаксису, що пришвидшує розробку та знижує кількість помилок.

Наявність інструментів для аналізу та обробки даних, таких як pandas, NumPy, що є корисними для реалізації функціоналу скринінгу, фільтрації монет, виявлення трендів тощо.

Сумісність з Telegram-ботами через бібліотеки pyTelegramBotAPI, aiogram, telebot тощо.

Робота з API криптобірж

Криптовалютні біржі надають відкриті програмні інтерфейси (API), за допомогою яких можна отримувати в режимі реального часу такі дані:

поточні ціни активів,

обсяг торгів,

історію змін вартості,

інформацію про нові лістинги,

книги ордерів тощо.

Більшість платформ обміну криптовалютами, як-от Binance, OKX та Bybit, надають REST API для отримання даних і WebSocket API для підключення до потоків з миттєвим оновленням. Python має готові бібліотеки для взаємодії з цими API, що істотно полегшує їх впровадження.

Вибір криптобіржі: Binance

Серед усіх платформ особливу увагу було зосереджено на біржі **Binance** з огляду на наступні фактори:

Найбільші обсяги торгів на ринку криптовалют, що дозволяє будувати аналітику на основі актуальних і масштабних даних.

Наявність зручної документації API, яка охоплює широкий спектр функціональності — від інформаційних запитів до виконання торгових ордерів (у рамках майбутнього розширення функціоналу бота).

Підтримка великої кількості торгових пар, включаючи нові проєкти, що дозволяє ефективно реалізувати функції виявлення нових трендів і можливостей для інвестування.

Стабільність та надійність платформи, що важливо при реалізації системи моніторингу у реальному часі.

Додаткові технології

SQLite — реляційна база даних, яка використовується для локального зберігання інформації про користувачів бота, налаштування алертів, улюблені монети тощо. Вона легко інтегрується з Python, не вимагає налаштування сервера та підходить для невеликих і середніх проєктів.

Асинхронні бібліотеки Python (asyncio, aiohttp) — застосовуються для одночасної обробки декількох джерел інформації (біржі Binance, OKX, Bybit), що забезпечує ефективну роботу в реальному часі без затримок.

Таким чином, обраний технологічний стек дозволяє реалізувати функціональний, масштабований та безпечний чат-бот для моніторингу активності на криптобіржах з можливістю подальшого розширення.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

2.1 Архітектура системи та взаємодія компонентів (бот – БД – API бірж)

Функціонал чат-бота, зосередженого на перевірці фінансових активів та знаходженні алгоритмів на криптобіржах, базується на взаємодії ключових складників: інтерфейсу користувача в месенджері, програмного ядра бота, зовнішніх API криптобірж та внутрішньої бази даних.

Загальна архітектура системи. Система має модульну структуру і включає наступні компоненти:

Користувач (через месенджер) — надсилає команди або запити боту, наприклад: /price BTC, /addalert ETH 3000, або натискає кнопки меню.

Чат-бот (основний логічний модуль) — обробляє запити користувача, взаємодіє з базою даних та API криптобірж, а також повертає у відповідь повідомлення, таблиці або графіки.

База даних (SQLite) — зберігає персональні налаштування користувачів, активні алерти, історію запитів та результати скринінгу.

API криптобірж (наприклад, Binance API) — джерело актуальних ринкових даних: ціни активів, обсяги торгів, нові лістинги, глибина ринку тощо.

Сценарій взаємодії між компонентами

Користувач надсилає запит боту (наприклад, хоче дізнатися поточну ціну монети або додати ціновий алерт).

Бот приймає запит, обробляє його за допомогою логіки, прописаної у програмному коді (Python), та визначає, які дії необхідно виконати.

Якщо потрібні **дані з біржі**, бот надсилає HTTP-запит до API обраної криптобіржі (наприклад, Binance REST API) і очікує відповідь у форматі JSON.

У випадку дій, що потребують зберігання інформації (наприклад, створення алерту), бот звертається до **локальної бази даних SQLite**, додаючи або змінюючи відповідні записи.

Після обробки запиту **бот формує повідомлення** (текстове, з кнопками, або таблицю) та надсилає його користувачу.

Особливості реалізації

Асинхронність: система побудована на асинхронному виконанні запитів (через asyncio), що дозволяє обробляти одночасно кілька запитів користувачів та API.

Розмежування логіки: кожен модуль системи виконує чітко визначені функції — обробка повідомлень, робота з API, доступ до бази даних.

Гнучкість і масштабованість: архітектура дозволяє легко додавати підтримку нових бірж (наприклад, OKX або Bybit), нові типи алертів або додаткові фільтри для скринінгу активів.

2.2. Розробка логіки скринінгу та генерації сповіщень

Логіка сканування фінансових активів та формування сповіщень є ключовим компонентом функціональності чат-бота, оскільки вона забезпечує вчасне виявлення значних змін на криптовалютному ринку та сповіщення користувачів про ці події. Створення ефективного алгоритму дозволяє автоматизувати моніторинг ринкових подій, уникаючи постійної необхідності втручання користувача.

Основні етапи логіки роботи чат-бота

Ініціалізація користувача та налаштування алертів

Користувач запускає бота та отримує стартове меню з функціями.

За допомогою команди /addalert або кнопки в меню користувач може задати монету та цільову ціну, при досягненні якої потрібно надіслати сповіщення.

Дані зберігаються у локальній базі даних SQLite разом із ідентифікатором користувача.

Періодичний скринінг ринку

Бот використовує API криптобірж (наприклад, Binance API) для отримання актуальних цін фінансових активів з інтервалом, заданим у конфігурації (наприклад, кожні 5 секунд).

Дані отримуються у форматі JSON, після чого парсяться і аналізуються за заздалегідь визначеними умовами.

Виявлення умов для сповіщення

Для кожного активного алерту бот порівнює поточну ринкову ціну з цільовим значенням.

Якщо умова виконана (наприклад, ціна BTC перевищила 70 000 USDT), бот формує повідомлення з деталями сповіщення.

Генерація та надсилання сповіщень

Повідомлення включає назву монети, поточну ціну, відсоток зміни за 24 години та час досягнення порогу.

Сповіщення надсилається користувачу через месенджер автоматично з кнопкою «Видалити алерт» або «Додати новий».

Очищення або повторне використання алертів

Після надсилання сповіщення бот або автоматично видаляє алерт з бази, або переводить його в архів для подальшої обробки, залежно від налаштувань користувача.

Додаткові сценарії використання

Масовий скринінг монет — за допомогою алгоритмів виявлення аномальних змін (наприклад, ≥10% зростання/падіння за останню годину) бот надсилає огляд монет, що демонструють нестандартну динаміку.

Крос-біржовий скринінг — бот може порівнювати ціни однієї й тієї ж монети на різних біржах (наприклад, Binance, OKX, Bybit) для виявлення можливостей арбітражу.

Індикативні фільтри — користувач може задати додаткові умови фільтрації: обсяг торгів, зміна ціни за добу, рівень волатильності.

2.3. Реалізація мессенджер-інтерфейсу користуча

Користувацький інтерфейс в чат-боті є критично важливим для забезпечення комфортної взаємодії з програмою. З огляду на особливості цільової аудиторії (люди, що цікавляться криптовалютами), інтерфейс месенджера повинен бути інтуїтивно зрозумілим, легким у використанні та забезпечувати швидкий доступ до потрібної інформації за допомогою кількох кліків або команд.

Основні реалізовані функції

1. Меню команд та стартова взаємодія

При першому запуску бот надсилає привітальне повідомлення з поясненням доступних функцій.

Головне меню реалізоване через кнопки (ReplyKeyboardMarkup) з командами:

📈 Перевірити ціну монети

🔔 Додати алерт

📋 Список алертів

❌ Видалити алерт

⚙ Налаштування

2. Запит ціни на актив

За запитом користувача (наприклад, через кнопку або команду /price) бот запитує назву монети (наприклад, BTC, ETH).

Після введення символу монети, бот отримує актуальні дані з API криптобіржі (наприклад, Binance) і надсилає повідомлення з інформацією:

📊 BTC/USDT

Поточна ціна: 67,540.12 USDT

Зміна за 24 години: +2.45%

Повідомлення також може містити графік або кнопки для додаткових дій.

3. Додавання алертів

Користувач запускає команду /addalert або натискає відповідну кнопку.

Бот запитує монету (наприклад, ETH) та цільову ціну.

Після отримання даних, бот зберігає інформацію до бази даних і підтверджує створення:

🔔 Алерт для ETH створено!

Ми сповістимо вас, коли ціна досягне 3500 USDT.

4. Перегляд і видалення алертів

Користувач може в будь-який момент перевірити список активних алертів командою /listalerts.

Для кожного алерту відображається:

Назва монети

Цільова ціна

Поточна ціна

Видалення здійснюється через команду /deletealert .

5. Генерація сповіщень

При досягненні монетою цільового рівня бот надсилає автоматичне повідомлення користувачу:

🚨 ETH досягла ціни 3500 USDT!

Поточна ціна: 3501.24 USDT

6 . Налаштування інтервалу перевірки

Через меню налаштувань користувач може обрати частоту перевірки цін: кожні 15 сек, 30 сек, 1 хв.

Це дозволяє адаптувати бота під потреби трейдера або довгострокового інвестора.

2.4. Зберігання даних та умов моніторингу (структура бази даних)

Надійне сховище даних становить основу архітектури чат-бота, який взаємодіє з фінансовими активами. Воно зберігає інформацію про користувачів, конфігурації моніторингу, цільові ціни та історію сповіщень. Для реалізації обрано реляційну базу даних SQLite, що гарантує прийнятну продуктивність, простоту в налаштуванні та сумісність з Python-оточенням.

Основні цілі зберігання даних:

Зберігання інформації про користувачів (ідентифікатори, налаштування).

Фіксація умов моніторингу (цільова ціна, назва монети, напрямок зміни).

Відстеження активних та виконаних алертів.

Забезпечення масштабованості та можливості майбутньої інтеграції з іншими біржами або сервісами.

Структура бази даних

У рамках реалізації було створено кілька основних таблиць:

Таблиця 1. users

Містить базову інформацію про користувачів, які взаємодіють з ботом.

Поле	Тип	Опис
id	INTEGER	Унікальний ідентифікатор користувача
chat_id	TEXT	Telegram ID (або інший ID з месенджера)
username	TEXT	Ім'я користувача (нікнейм)
check_interval	INTEGER	Інтервал перевірки цін (у секундах)
created_at	DATETIME	Дата додавання користувача

Таблиця 2. alerts

Зберігає умови моніторингу та активні алерти.

Поле	Тип	Опис
id	INTEGER	Унікальний ідентифікатор алерту
user_id	INTEGER	Зовнішній ключ до таблиці users
symbol	TEXT	Назва монети (наприклад, BTC, ETH)
target_price	REAL	Цільова ціна, при досягненні якої спрацьовує алерт
direction	TEXT	Напрямок моніторингу: above або below
is_active	BOOLEAN	Статус алерту (активний/виконаний)
created_at	DATETIME	Дата створення алерту

Таблиця 3. notifications

Фіксує надсилання повідомлень про досягнення цільових значень.

Поле	Тип	Опис
id	INTEGER	Унікальний ідентифікатор запису
alert_id	INTEGER	Зовнішній ключ до таблиці alerts
notified_at	DATETIME	Дата та час спрацювання алерту
price_trigger	REAL	Ціна, при якій було надіслано сповіщення

Взаємодія бота з базою даних. Чат-бот взаємодіє з базою через модуль sqlite3, використовуючи ORM-або SQL-запити для:

Додавання нових користувачів при старті бота.

Збереження нових алертів при їх створенні.

Пошуку алертів для регулярної перевірки (виконується через async loop).

Вимкнення алертів після надсилання сповіщення.

Перегляду історії алертів користувачем.

Захист і надійність зберігання. Для забезпечення коректної роботи з базою:

Усі запити реалізовані з обробкою винятків (try-except) для запобігання помилкам доступу.

Регулярне резервне копіювання файлу БД може бути реалізоване на сервері або за допомогою сторонніх сервісів (наприклад, cron + cloud).

У перспективі можлива заміна SQLite на PostgreSQL або іншу СУБД при розширенні функціоналу або збільшенні кількості користувачів.

РОЗДІЛ 3. ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА ОЦІНЮВАННЯ

3.1. Тестування бота на різних платформах (мобільна, десктоп)

Тестування чат-боту є критично важливим заходом для гарантування його коректної роботи, надійності та зручності взаємодії. Враховуючи, що сучасні месенджери доступні на широкому спектрі пристроїв (смартфонах, планшетах, ноутбуках і настільних комп'ютерах), необхідно переконатися, як бот працює на кожному з них, а також в різних операційних системах, на кшталт iOS, Android та Windows.

Основні етапи тестування

1. Тестування на мобільних пристроях

Тестування на iOS та Android:

Було здійснено перевірку роботи бота у мобільних додатках месенджера на операційних системах iOS і Android.

Основні аспекти тестування:

Швидкість реакції бота: оцінювалась затримка між відправленням повідомлення користувачем і відповіддю від бота.

Коректність інтерфейсу: перевірялося відображення елементів, таких як кнопки та повідомлення, відповідно до розмірів екрана.

Зручність взаємодії: оцінювалась інтуїтивність і доступність елементів керування, таких як меню, кнопки вибору, функції запису, редагування профілю тощо.

2. Тестування на десктопних платформах (ПК)

Тестування через десктопну або веб-версію месенджера:

Бот перевірявся на стаціонарних комп’ютерах через браузер та настільну програму месенджера.

Основні перевірки:

Відображення інтерфейсу: перевірка чіткості та розміщення елементів на великому екрані.

Логіка взаємодії: тестування переходів між етапами — наприклад, відправка повідомлень, встановлення алертів.

Стабільність: оцінювалась здатність бота обробляти послідовні запити без зависань чи помилок.

3. Тестування кросплатформеності

Перевірка однакової функціональності на всіх пристроях:

Було здійснено тестування на різних типах пристроїв (смартфонах, планшетах, ПК), щоб переконатися в однаковій поведінці бота.

Оцінювались такі аспекти:

Синхронізація даних: чи зберігається інформація про користувача, історія алертів на встановлені рівні.

Уніфікований інтерфейс: чи однаково виглядає інтерфейс, повідомлення.

Сумісність із версіями месенджера: перевірка роботи бота у різних версіях клієнтів месенджера.

Основні критерії оцінки якості:

Швидкодія: час відповіді бота на запит користувача.

Точність: правильність логіки взаємодії та достовірність наданої інформації.

Кросплатформеність: стабільна робота на різних пристроях і ОС.

Юзабіліті: зручність використання інтерфейсу.

3.2. Аналіз помилок та шляхи їх усунення

Виправлення недоліків, виявлених під час тестування, є ключовим етапом забезпечення надійної, захищеної та ефективної роботи чат-бота, створеного для пошуку учасників та алгоритмів на криптобіржах. Під час перевірки функціоналу на різних платформах і у різних сценаріях використання було виявлено низку технічних проблем, що вимагали оперативного усунення.

1. Некоректне сприйняття текстових команд на старих версіях месенджера

Проблема: У застарілих версіях деяких месенджерів користувачі стикалися з тим, що текстові команди не розпізнавались або бот не реагував на них належним чином.

Рішення: Оновлено механізм обробки вхідних повідомлень. Зокрема, додано підтримку альтернативних форматів команд (наприклад, з додатковими пробілами, різним регістром літер) та реалізовано fallback-механізми для обробки неочікуваного вводу. Це дозволило забезпечити стабільну роботу навіть у застарілих версіях клієнтів.

2. Проблеми з інтеграцією із зовнішніми системами

Проблема: Під час синхронізації даних з зовнішніми сервісами виникали затримки або помилки у передачі інформації, що впливало на актуальність отримуваних даних.

Рішення: Оптимізовано структуру API-запитів і реалізовано багаторівневу обробку помилок. Покращено механізм підтвердження синхронізації, що забезпечило стабільний обмін інформацією про учасників, алгоритми та інші операції.

Узагальнення та шляхи вирішення

Для підвищення надійності та ефективності роботи месенджер-бота були впроваджені такі заходи:

Оновлення логіки обробки команд для сумісності з різними версіями месенджерів та коректної взаємодії з зовнішніми API і базою даних.

Проведення повторного тестування після кожного виправлення, що дозволило перевірити працездатність оновленого функціоналу у реальних умовах.

Випуск оновлень, що включали виправлення виявлених помилок, оптимізацію роботи бота та покращення зручності взаємодії.

Покращення зворотного зв’язку з користувачем — впроваджено інформативні повідомлення про результати обробки команд, наявність помилок чи успішність операцій.

3.3. Приклади роботи бота: сценарії взаємодії

У процесі тестування чат-бота було ретельно перевірено його роботу відповідно до вимог, сформульованих на етапі проєктування. Метою тестування було підтвердження коректності виконання ключових сценаріїв взаємодії

користувача з ботом, що дозволяють здійснювати пошук учасників криптобірж, аналіз алгоритмів торгівлі, налаштування фільтрів та отримання сповіщень.

1. Початок роботи з ботом

Функціональність: Користувач запускає чат-бота та вводить команду старту (/start).

Очікуваний результат: Бот вітає користувача та пропонує перелік доступних опцій (перегляд бірж, пошук учасників, додавання фільтрів, перегляд алгоритмів).

Результат тестування: Початкове меню працює стабільно, відображається у зрозумілому вигляді, бот оперативно реагує на стартову команду.

2. Пошук торгових алгоритмів

Функціональність: Користувач вводить команду пошуку алгоритмів (наприклад, алгоритми скальпінг).

Очікуваний результат: Бот повертає перелік знайдених алгоритмів з характеристиками (біржа, монета, обсяг торгів, волатильність).

Результат тестування: Алгоритми знаходяться коректно, відповідають заданим критеріям, інформація надається з використанням даних API криптобірж.

3. Оповіщення про активність

Функціональність: Користувач активує функцію сповіщень (наприклад, увімкнути сповіщення про зростання об'єму торгів).

Очікуваний результат: При настанні відповідної умови бот надсилає сповіщення з коротким описом події та висновками.

Результат тестування: Оповіщення надсилаються вчасно, зміст — інформативний, дозволяє швидко зорієнтуватися у ринковій ситуації.

Підсумок. **Всі ключові функції чат-бота протестовано та підтверджено їх стабільну роботу. Взаємодія побудована через текстові команди без використання кнопок, що дозволяє зберігати універсальність бота та забезпечити сумісність із різними версіями клієнтів месенджера. Бот**

демонструє високий рівень функціональності, зручності та адаптивності до запитів користувача.

3.4. Інструкція користувача

Для ефективного використання чат-бота, створеного для моніторингу активності на криптобіржах, рекомендується дотримуватись наступної інструкції:

1. Запуск бота

🔎 Знайдіть чат-бота в месенджері за його назвою через функцію пошуку.

▶ Натисніть кнопку **«Start»** або введіть команду /start, щоб розпочати роботу.

2. Головне меню

Після запуску бот запропонує перелік основних функцій:

🔍 **Пошук учасників біржі**

📊 **Аналіз алгоритмів**

⚙ **Налаштування фільтрів**

🔔 **Отримати сповіщення**

ℹ Допомога

Взаємодія відбувається через текстові команди, тому оберіть потрібну опцію, ввівши її назву або відповідну команду.

3. Пошук учасників біржі

З початком запуску вам почнуть приходити алерти про різкі рухи в цінах різних інструментів на біржах.

4. Аналіз торгових алгоритмів

Почніть стежити за сигналами перевіряючи, і вже аналізувавши все на ринку, брокеру тощо.

5. Налаштування фільтрів

Для точнішого пошуку введіть параметри фільтрації:

Ви можете вибрати % руху який вас цікавить

Також ви можете вказати оповіщення на рівні які вас цікавлять, та бот надішле вам сповіщення по досягненню цілі.

Приклад команди: фільтр: /addalert /ethusdt 2450

6. Отримання сповіщень

Активуйте функцію, в чат мессенджері.

Бот надсилатиме повідомлення у випадках:

різкого зростання або падіння курсу монети

алертів щодо важливих рівнів

Сповіщення допомагають **швидко реагувати на критичні зміни ринку**.

Порада. **Для ознайомлення з доступними командами введіть: допомога або /help. Бот відобразить перелік усіх функцій та приклади команд.**

ВИСНОВКИ

У процесі виконання даної роботи було розроблено функціонального чат-бота, здатного забезпечити оперативний доступ до аналітичної інформації про ринок криптовалют, зокрема щодо учасників криптобірж та торгових алгоритмів. Цей інструмент став результатом дослідження особливостей криптовалютного ринку, вивчення API криптобірж та реалізації сучасних технологічних рішень у сфері автоматизації обробки даних. Отримані результати свідчать про актуальність обраної теми, її прикладне значення, а також про потенціал для подальшого розвитку і вдосконалення системи.

Одним із основних досягнень є створення стабільного, багатофункціонального та зручного інструмента, що дозволяє користувачам швидко отримувати необхідну інформацію про ринок криптовалют. Бот забезпечує пошук учасників криптобірж за різними параметрами, аналіз торгових стратегій на основі даних з біржових API, а також надсилає сповіщення про важливі ринкові події. Таким чином, користувачі отримують доступ до комплексної інформації в режимі реального часу, що значно підвищує ефективність прийняття рішень у динамічному середовищі крипторинку.

У ході реалізації проєкту було приділено увагу питанням зручності інтерфейсу та сумісності бота з різними платформами. Було проведено тестування роботи системи у популярних месенджерах, яке показало високу стабільність функціонування, швидке реагування на запити користувачів та коректне відображення отриманих результатів. Виявлені технічні помилки усувалися на етапі випробувань, що дозволило забезпечити високу якість продукту на завершальному етапі.

Особливу увагу було приділено автоматизації обробки великих обсягів біржової інформації. Це дозволило зменшити витрати часу на аналіз, підвищити точність результатів, а також створити умови для масштабування функціоналу в майбутньому. Автоматичний збір даних, обробка запитів, фільтрація інформації за обраними критеріями та формування повідомлень про ринкові

зміни — усе це реалізовано з урахуванням сучасних вимог до цифрових продуктів у фінансовій сфері.

Інноваційним аспектом проєкту стало впровадження механізму сповіщень, який дозволяє користувачам оперативно дізнаватися про значні коливання на ринку, зміну цін активів, активність трейдерів або появу нових алгоритмів торгівлі. Це дає змогу оперативно реагувати на ринкові сигнали, знижуючи ризики та підвищуючи ефективність дій.

Також важливою частиною проєкту стала побудова структури даних, яка забезпечує гнучкість, масштабованість і надійність системи. Застосовано модульний підхід до програмної архітектури, що дозволяє легко інтегрувати нові функції або оновлювати наявні компоненти. Використання API криптобірж забезпечує актуальність даних, а реалізація відповідного механізму обробки запитів гарантує точність та швидкість виконання функцій.

Аналіз результатів підтвердив доцільність створення чат-бота як ефективного рішення для трейдерів, аналітиків, дослідників і ентузіастів криптовалют. Завдяки інтеграції з біржами бот дозволяє отримувати аналітику без необхідності вручну переглядати численні джерела. Автоматизація процесів і зручність інтерфейсу значно підвищують якість взаємодії з крипторинком.

Крім досягнутих результатів, у процесі роботи було виявлено перспективні напрями для подальшого розвитку системи. По-перше, доцільним є розширення переліку підтримуваних криптобірж та джерел даних, що забезпечить ще ширше охоплення ринку. Це дозволить користувачам працювати з більш повною картиною ринкової ситуації, отримуючи дані з різних географічних регіонів та платформ.

По-друге, впровадження алгоритмів машинного навчання може суттєво покращити якість аналізу та точність прогнозів торгових стратегій. Бот зможе виявляти закономірності у діях трейдерів, оптимізувати рекомендації на основі історичних даних та формувати індивідуальні підказки для кожного користувача. Це зробить продукт ще більш персоналізованим та ефективним.

По-третє, розвиток функціоналу для самостійного створення алертів користувачами відкриє нові можливості для гнучкого налаштування системи. Кожен користувач зможе задати власні критерії для сповіщень, залежно від своєї стратегії або інтересів. Такий підхід дозволяє задовольнити потреби як новачків, так і досвідчених учасників ринку.

Не менш важливим напрямом удосконалення є підвищення рівня безпеки, зокрема через впровадження сучасних методів шифрування та захисту персональних даних. З урахуванням високої вартості та чутливості біржової інформації, питання конфіденційності та захисту є ключовими для подальшого розвитку системи.

Також перспективним є впровадження мультимовної підтримки, що дозволить значно розширити аудиторію користувачів у глобальному масштабі. Підтримка кількох мов сприятиме доступності бота для користувачів із різних країн та культур, роблячи його універсальним інструментом у світі криптовалют.

Таким чином, підсумовуючи вищевикладене, можна зробити висновок, що розроблений чат-бот є практично значущим, технологічно досконалим і гнучким рішенням для роботи з крипторинком. Його впровадження дозволяє оптимізувати процеси аналізу, прийняття рішень та взаємодії з біржовими даними. Проєкт має великий потенціал для подальшого розвитку, розширення функціональності та комерційного застосування.

Розробка такого інструменту є кроком у напрямку цифровізації фінансових послуг, створення інтелектуальних систем підтримки рішень та підвищення доступності фінансової аналітики для широкого кола користувачів. У результаті виконаної роботи було не лише досягнуто поставлених цілей, а й закладено надійну основу для подальших досліджень і вдосконалення у сфері криптовалютного програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Глушаков І. В. Програмування мовою Python : навч. посіб. / І. В. Глушаков. – Харків : ХНУРЕ, 2021. – 216 с.
2. Шматков Є. О. Розробка програмних застосунків у середовищі Telegram / Є. О. Шматков // Вісник ЖДТУ. – 2020. – №3(93). – С. 56–61.
3. Офіційна документація Telegram Bot API. – https://core.telegram.org/bots/api
4. Офіційна документація python-telegram-bot. – https://docs.python-telegram-bot.org/
5. Binance API Documentation. – https://binance-docs.github.io/apidocs/spot/en/
6. SQLite Documentation. – https://www.sqlite.org/docs.html
7. Крамарьов С. О. Основи баз даних : навч. посіб. / С. О. Крамарьов. – Львів : ЛНУ ім. Івана Франка, 2022. – 184 с.
8. Martelli A., Ravenscroft A., Ascher D. Python Cookbook. – 3rd ed. – O’Reilly Media, 2013. – 706 p.
9. Grinberg M. Flask Web Development: Developing Web Applications with Python. – 2nd ed. – O’Reilly Media, 2018. – 258 p.
10. Жуйков Д. А. Побудова телеграм-бота з використанням Python // Наукові записки. Серія: Комп’ютерні науки. – 2021. – №1(35). – С. 34–39.
11. Глушко О. В. Автоматизація збору біржової інформації за допомогою API // Вісник КНУ. Серія «Прикладна математика». – 2020. – №28. – С. 91–95.
12. Пономарьов О. М. Технології розробки програмних застосунків: навч. посібник. – Київ : КНЕУ, 2020. – 312 с.
13. Allen B. Web Scraping with Python: Collecting Data from the Modern Web. – 2nd ed. – O’Reilly Media, 2017. – 330 p.
14. Stack Overflow – Python-telegram-bot tag. – https://stackoverflow.com/questions/tagged/python-telegram-bot
15. GitHub – python-telegram-bot: A library that provides a pure Python interface for the Telegram Bot API. – https://github.com/python-telegram-bot/python-telegram-bot

ДОДАТКИ

Додаток А. Вихідний код

```
import json
from datetime import datetime, timedelta
from telegram import Bot
from telegram.ext import Application, CommandHandler
import time
import threading
from threading import Thread
import asyncio
import requests
import sqlite3
import logging

# Налаштування логування для відстеження інформації та помилок
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

CHECK_INTERVAL = 10  # Інтервал перевірки цін у секундах

print(f"333")  # Виведення для тесту

async def main():
    """ Головна асинхронна функція для перевірки цін """
    logging.info("Запуск головної функції")  # Логування запуску функції
    print(f"222")  # Виведення для тесту
    while True:
        new_prices = await get_current_prices()  # Отримання нових цін
        if new_prices:
            await check_price_changes(new_prices)  # Перевірка зміни цін
        await asyncio.sleep(CHECK_INTERVAL)  # Очікування перед наступною перевіркою

def get_current_price(symbol):
```

```
    """ Отримання поточної ціни для заданого символу """
    url = f'https://fapi.binance.com/fapi/v1/ticker/price?symbol={symbol.upper()}'
    logging.info(f"Отримання ціни для {symbol} за URL: {url}")  # Логування запиту до Binance


    try:
        response = requests.get(url)  # Виконання HTTP запиту до API Binance
        response.raise_for_status()  # Перевірка на помилки HTTP
        data = response.json()  # Парсинг відповіді JSON


        if 'price' in data:  # Якщо ціна присутня в даних
            price = float(data['price'])  # Конвертація ціни в число з плаваючою точкою
            logging.info(f"Поточна ціна {symbol}: {price}")  # Логування поточної ціни
            return price
        else:
            logging.warning(f"Не знайдено ціну у відповіді Binance для {symbol}")  #
Попередження, якщо ціна не знайдена
            return None
    except requests.RequestException as e:
        logging.error(f"Помилка запиту до Binance: {e}")  # Логування помилок запиту
        return None


def create_connection():
    """ Створення підключення до бази даних """
    try:
        conn = sqlite3.connect('crypto_bot.db')  # Підключення до бази даних SQLite
        logging.info("Підключено до бази даних crypto_bot.db")  # Логування успішного
підключення
        return conn
    except sqlite3.Error as e:
        logging.error(f"Помилка підключення до БД: {e}")  # Логування помилок підключення
        return None
```

```
def check_alerts():
    """ Перевірка спрацьовування алертів """
    logging.info("Перевірка активних алертів")  # Логування початку перевірки алертів
    conn = create_connection()  # Створення підключення до БД
    if conn is None:
        logging.error("Не вдалося підключитися до бази даних")  # Логування помилки підключення
        return

    cursor = conn.cursor()
    cursor.execute("SELECT * FROM alerts")  # Отримання всіх алертів з БД
    alerts = cursor.fetchall()  # Завантаження алертів

    for alert in alerts:
        alert_id = alert[0]  # ID алерта
        symbol = alert[1].lstrip('/')  # Символ монети (без слешу)
        alert_price = alert[2]  # Встановлена ціна алерта
        chat_id = alert[3]  # ID чату Telegram для сповіщення

        logging.info(f"Перевірка алерта {alert_id} для {symbol} на {alert_price}$")  # Логування перевірки конкретного алерта
        current_price = get_current_price(symbol)  # Отримання поточної ціни

        if current_price is not None and round(current_price, 4) == round(alert_price, 4):  # Перевірка чи досягнута ціна
            message = f"🚨 {symbol} досяг {alert_price}$! Поточна ціна: {current_price}$."  # Формування повідомлення
            logging.info(f"Алерт спрацював: {message}")  # Логування спрацьовування алерта
            send_telegram_message(chat_id, message)  # Надсилання повідомлення в Telegram

            delete_alert(alert_id)  # Видалення алерта після спрацювання

    conn.close()  # Закриття з'єднання з БД
```

```
def delete_alert(alert_id):
    """ Видалення алерта з бази даних після його спрацювання """
    logging.info(f"Видалення алерта {alert_id}")  # Логування видалення алерта
    conn = create_connection()  # Створення підключення до БД
    if conn is None:
        logging.error("Не вдалося підключитися до бази даних для видалення алерта")  # Логування помилки підключення
        return

    cursor = conn.cursor()
    cursor.execute("DELETE FROM alerts WHERE id = ?", (alert_id,))  # Видалення алерта з БД
    conn.commit()  # Застосування змін до бази даних
    conn.close()  # Закриття з'єднання з БД
    logging.info(f"Алерт {alert_id} успішно видалено")  # Логування успішного видалення



def send_telegram_message(chat_id, message):
    """ Надсилає повідомлення у Telegram """
    bot_token = 'YOUR_BOT_TOKEN'  # Токен бота для доступу до Telegram API
    url = f'https://api.telegram.org/bot{bot_token}/sendMessage'  # URL для надсилання повідомлення
    payload = {
        'chat_id': chat_id,  # ID чату, куди надсилати повідомлення
        'text': message  # Текст повідомлення
    }
    logging.info(f"Надсилання повідомлення у Telegram: {message}")  # Логування повідомлення
    try:
        requests.post(url, data=payload)  # Надсилання POST запиту для повідомлення
    except requests.RequestException as e:
        logging.error(f"Помилка надсилання повідомлення у Telegram: {e}")  # Логування помилок запиту
```

```
def check_alerts_periodically():
    """ Запускає перевірку алертів у фоновому потоці """
    while True:
        logging.info("Перевірка алертів у фоновому режимі")  # Логування початку перевірки
        check_alerts()  # Перевірка алертів
        time.sleep(3)  # Затримка між перевірками


alert_thread = threading.Thread(target=check_alerts_periodically)  # Створення потоку для фонової перевірки
alert_thread.daemon = True  # Потік буде завершено при завершенні основної програми
alert_thread.start()  # Запуск фонової перевірки


def add_alert(symbol, price, chat_id):
    """ Додає новий алерт у базу даних """
    conn = create_connection()  # Створення підключення до БД
    cursor = conn.cursor()
    cursor.execute("INSERT INTO alerts (symbol, price, chat_id) VALUES (?, ?, ?)", (symbol, price, chat_id))  # Додавання алерта
    conn.commit()  # Застосування змін до БД
    conn.close()  # Закриття з'єднання
    logging.info(f"Алерт додано: {symbol} - {price}$ для чату {chat_id}")  # Логування успішного додавання


def check_alert(symbol, price):
    """ Перевіряє, чи існує заданий алерт """
    conn = create_connection()  # Підключення до БД
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM alerts WHERE symbol = ? AND price = ?", (symbol, price))  # Пошук алерта в БД
    alert = cursor.fetchone()  # Отримання першого знайденого алерта
    conn.close()  # Закриття з'єднання
    return alert  # Повернення алерта, якщо знайдений
```

```
def is_valid_symbol(symbol):
    """ Перевіряє, чи існує символ на Binance """
    symbol = symbol.lstrip('/')  # Видалення слешу на початку, якщо він є
    url = f'https://fapi.binance.com/fapi/v1/ticker/price?symbol={symbol.upper()}'  # URL для перевірки символу
    logging.info(f"Перевірка валідності символу: {symbol}")  # Логування перевірки
    try:
        response = requests.get(url)  # HTTP запит для перевірки символу
        data = response.json()  # Парсинг відповіді
        return 'price' in data  # Якщо є поле 'price', то символ валідний
    except requests.RequestException as e:
        logging.error(f"Помилка перевірки символу {symbol}: {e}")  # Логування помилки
        return False

async def add_alert_command(update, context):
    """ Команда для додавання алерта через Telegram """
    if len(context.args) != 2:  # Перевірка правильності формату команди
        await update.message.reply_text("Правильний формат: /addalert <символ> <ціна>")
        return

    symbol = context.args[0].upper()  # Символ монети, переведений до верхнього регістру
    price = float(context.args[1])  # Ціна, на яку встановлюється алерт
    chat_id = update.message.chat_id  # ID чату користувача в Telegram

    if not is_valid_symbol(symbol):  # Перевірка валідності символу
        await update.message.reply_text(f"Символ {symbol} не знайдений на Binance. Введіть правильний символ.")
        return

    add_alert(symbol, price, chat_id)  # Додавання алерта в БД
```

```
    await update.message.reply_text(f"Алерт для {symbol} встановлений на ціну {price}$.")  # Сповіщення користувача


async def check_alert_command(update, context):
    """ Команда для перевірки наявності алерта через Telegram """
    if len(context.args) != 2:  # Перевірка правильності формату команди
        await update.message.reply_text("Правильний формат: /checkalert <символ> <ціна>")
        return

    symbol = context.args[0].upper()  # Символ монети
    price = float(context.args[1])  # Ціна для перевірки

    conn = create_connection()  # Підключення до БД
    cursor = conn.cursor()

    cursor.execute("SELECT * FROM alerts WHERE symbol = ? AND price = ?", (symbol, price)) # Перевірка наявності алерта
    alert = cursor.fetchone()  # Отримання першого знайденого алерта

    conn.close()  # Закриття з'єднання

    if alert:
        await update.message.reply_text(f"Алерт для {symbol} встановлений на ціну {price}$.")  # Повідомлення про наявність алерта
    else:
        await update.message.reply_text(f"Не знайдено алерта для {symbol} на ціну {price}$.")  # Повідомлення про відсутність алерта


TOKEN = "7567505791:AAE5Yqbd7gG9ydsx2_inJJmnS1Ln6rmh2Ts"  # Токен бота Telegram


application = Application.builder().token(TOKEN).build()  # Створення об'єкта додатку для Telegram бота
```

```
application.add_handler(CommandHandler("addalert", add_alert_command))  # Обробник команди для додавання алерта
application.add_handler(CommandHandler("checkalert", check_alert_command))  # Обробник команди для перевірки алерта


def tg_start_pool():
    """ Запускає Telegram бот на постійному polling для отримання повідомлень від користувачів """
    application.run_polling()  # Запуск polling бота для отримання оновлень


tg_pool_thread = None  # Змінна для зберігання потоку, що обробляє polling


CHAT_ID_FILE = "chat_ids.json"  # Файл для збереження ID чатів
try:
    with open(CHAT_ID_FILE, "r") as file:
        config = json.load(file)  # Читання конфігурації з JSON файлу
except (FileNotFoundError, json.JSONDecodeError, ValueError) as e:
    print(f"Помилка завантаження JSON: {e}")  # Обробка помилок при читанні JSON
    config = {"TOKEN": "", "CHAT_ID": []}  # Створення порожньої конфігурації
    with open(CHAT_ID_FILE, "w") as file:
        json.dump(config, file)  # Запис порожньої конфігурації до файлу


TOKEN = config.get("TOKEN", "")  # Отримання токену з конфігурації
CHAT_IDS = config.get("CHAT_ID", [])  # Отримання чат ID з конфігурації


if not TOKEN:
    raise ValueError("Токен не знайдено або він порожній!")  # Перевірка на наявність токену


BASE_URL = "https://fapi.binance.com"  # Основний URL для API Binance
ENDPOINT = "/fapi/v1/ticker/price"  # API ендпоінт для отримання цін
THRESHOLDS = [0.01, 0.04, 0.09, 0.20]  # Пороги для змін ціни, які будуть відстежуватись
CHECK_INTERVAL = 1.5  # Інтервал перевірки цін у секундах
```

```
bot = Bot(token=TOKEN)  # Ініціалізація Telegram бота за допомогою токену

price_history = {}  # Зберігання історії цін для кожного символу

async def get_current_prices():
    """ Отримує поточні ціни для всіх монет на Binance """
    try:
        response = requests.get(BASE_URL + ENDPOINT, timeout=5)  # Запит до API Binance
        print(f"Запит до API Binance, статус-код: {response.status_code}")
        if response.status_code == 200:
            prices = response.json()  # Парсинг відповіді
            return {item['symbol']: float(item['price']) for item in prices}  # Формування словника цін
        else:
            print(f"Помилка {response.status_code}: {response.text}")  # Логування помилки
            return {}
    except requests.exceptions.RequestException as e:
        print(f"Помилка при запиті до API: {e}")  # Логування помилки запиту
        return {}

async def check_price_changes(new_prices):
    """ Перевіряє зміни цін і надсилає сповіщення, якщо зміни перевищують задані пороги """
    now = datetime.now()  # Поточний час
    messages = []  # Список для повідомлень

    for symbol, new_price in new_prices.items():  # Для кожної монети
        data = price_history.get(symbol)  # Отримання історії цін для символу

        if data is None:
            price_history[symbol] = {'open_price': new_price, 'last_checked': now}  # Ініціалізація
історії
            continue
```

```
        open_price = data['open_price']  # Початкова ціна монети
        last_checked = data['last_checked']  # Час останньої перевірки

        if now - last_checked >= timedelta(minutes=5):  # Якщо з часу останньої перевірки пройшло більше 5 хвилин
            price_history[symbol] = {'open_price': new_price, 'last_checked': now}  # Оновлюємо історію
            continue

        change = (new_price - open_price) / open_price  # Обчислення зміни ціни
        abs_change = abs(change)  # Абсолютна зміна ціни

        for threshold in THRESHOLDS:  # Для кожного порогу
            if abs_change >= threshold and not data.get(f'notified_{threshold}', False):  # Якщо зміна перевищує поріг
                direction = "🟢" if change > 0 else "🔴"  # Визначаємо напрямок зміни
                change_percent = abs_change * 100  # Переведення зміни в проценти
                message = f"{direction} {symbol} BINANCE 5m {change_percent:.1f}%"  # Формування повідомлення
                messages.append(message)  # Додавання повідомлення
                price_history[symbol][f'notified_{threshold}'] = True  # Оновлення статусу повідомлення
                logging.info(f"Зміна ціни {symbol}: {change_percent:.1f}% (поріг {threshold})")  # Логування зміни ціни
                break
        else:
            price_history[symbol]['last_checked'] = now  # Оновлення часу перевірки

    await send_notifications(messages)  # Надсилання повідомлень

async def send_notifications(messages):
    """ Надсилає повідомлення в Telegram для кожного чату """
```

```
    if messages:
        full_message = "\n".join(messages)  # Об'єднання всіх повідомлень
        for chat_id in CHAT_IDS:  # Для кожного ID чату
            try:
                await bot.send_message(chat_id=chat_id, text=full_message)  # Надсилання повідомлення
                logging.info(f"Сповіщення відправлено до {chat_id}: {full_message}")  # Логування успіху
            except Exception as e:
                logging.error(f"Помилка надсилання до {chat_id}: {e}")  # Логування помилки


if __name__ == "__main__":
    logging.info("Запуск основного потоку")  # Логування запуску програми
    import nest_asyncio
    nest_asyncio.apply()  # Використання nest_asyncio для підтримки асинхронних функцій в середовищі, яке не підтримує їх нативно


    tg_pool_thread = threading.Thread(target=tg_start_pool)  # Створення потоку для запуску Telegram бота
    tg_pool_thread.daemon = True  # Встановлення потоку як daemon, щоб він завершувався разом з основним процесом
    tg_pool_thread.start()  # Запуск потоку


    asyncio.run(main())  # Запуск головної асинхронної функції
    logging.info("Основний процес завершений")  # Логування завершення процесу
```

Додаток А.1 – Додавання стовпця created_at до таблиці alerts

```
import sqlite3  # Імпорт бібліотеки для роботи з SQLite базою даних


DB_FILE = "crypto_bot.db"  # Назва файлу бази даних


def add_created_at_column():
    """ Додає стовпець created_at до таблиці alerts, якщо він ще не існує """
    conn = sqlite3.connect(DB_FILE)  # Підключення до бази даних
```

```
    cursor = conn.cursor()

    # Перевірка, чи є вже стовпець created_at в таблиці alerts
    cursor.execute("PRAGMA table_info(alerts);")
    columns = cursor.fetchall()
    column_names = [column[1] for column in columns]

    if 'created_at' not in column_names:
        try:
            cursor.execute("""
                ALTER TABLE alerts ADD COLUMN created_at TIMESTAMP
            """)  # Додаємо стовпець без значення за замовчуванням
            conn.commit()  # Застосування змін до БД
            print("Стовпець created_at успішно додано в таблицю alerts.")
        except sqlite3.OperationalError as e:
            print(f"Помилка: {e}")
    else:
        print("Стовпець 'created_at' вже існує в таблиці alerts.")

    conn.close()  # Закриття з'єднання

if __name__ == "__main__":
    add_created_at_column()  # Викликаємо функцію для додавання стовпця
```

Додаток А.2 – Видалення алертів старших за 48 годин і перевірка записів у таблиці alerts

```
import sqlite3
from datetime import datetime, timedelta

DB_FILE = "crypto_bot.db"

def delete_old_alerts():
```

```
    """ Видаляє алерти, яким більше 48 годин """
    conn = sqlite3.connect(DB_FILE)
    cursor = conn.cursor()

    # Теперішній час
    now = datetime.now()

    # Запит для видалення алертів, створених більше ніж 48 годин тому
    cursor.execute("""
        DELETE FROM alerts
        WHERE created_at < ?;
    """, (now - timedelta(hours=48),))

    conn.commit()  # Застосовуємо зміни до бази даних
    conn.close()  # Закриваємо підключення

    print("Старі алерти видалені успішно!")

def check_alerts_in_db():
    """ Перевірка наявності алертів в базі даних """
    conn = sqlite3.connect(DB_FILE)
    cursor = conn.cursor()

    cursor.execute("SELECT * FROM alerts")
    alerts = cursor.fetchall()

    if not alerts:
        print("❌ Алерти не знайдено в базі!")
    else:
        print("✅ Виявлені алерти в базі:")
        for alert in alerts:
```

```
        print(alert)


    conn.close()


if __name__ == "__main__":
    delete_old_alerts()  # Видаляємо старі алерти
    check_alerts_in_db()  # Перевіряємо наявність алертів в базі
```

Додаток А.3 Створення таблиць, додавання та перевірка алертів у базі даних

```
import sqlite3  # Імпорт бібліотеки для роботи з SQLite базою даних


def create_connection():
    """ Функція для створення підключення до бази даних """
    conn = sqlite3.connect('crypto_bot.db')  # Підключення до бази даних
    return conn


def get_alerts():
    """ Отримання всіх алертів з бази даних """
    conn = create_connection()  # Підключення до бази даних
    cursor = conn.cursor()  # Створення курсора для виконання запитів
    cursor.execute("SELECT * FROM alerts;")  # Запит для отримання всіх алертів
    alerts = cursor.fetchall()  # Отримання всіх результатів запиту
    conn.close()  # Закриття підключення до бази даних
    return alerts  # Повертаємо отримані алерти


def check_specific_alert(symbol, price):
    """ Перевірка наявності конкретного алерта для заданого символу та ціни """
    conn = create_connection()  # Підключення до бази даних
    cursor = conn.cursor()  # Створення курсора для виконання запитів


    # Запит для перевірки конкретного алерта
```

```
    cursor.execute("SELECT symbol, price FROM alerts WHERE symbol = ? AND price = ?",
(symbol, price))
    alert = cursor.fetchone()  # Отримуємо перший знайдений алерт (якщо є)

    conn.close()  # Закриття підключення до бази даних

    return alert  # Повертаємо знайдений алерт або None

def check_alert_command(update, context):
    """ Обробка команди /checkalert для перевірки наявності алерта """
    if len(context.args) != 2:  # Перевірка, чи правильно вказано аргументи
        update.message.reply_text("Правильний формат: /checkalert <символ> <ціна>")
        return

    symbol = context.args[0].upper()  # Символ монети (наприклад, "ETHUSDT")
    price = float(context.args[1])  # Ціна як float

    alert = check_specific_alert(symbol, price)  # Викликаємо функцію для перевірки алерта

    if alert:  # Якщо алерт знайдено
        update.message.reply_text(f"Алерт для {alert[0]} встановлений на ціну {alert[1]}$.")
    else:  # Якщо алерт не знайдено
        update.message.reply_text(f"Не знайдено алерта для {symbol} на ціну {price}$.")

def add_alert(symbol, price, chat_id):
    """ Додає новий алерт до бази даних """
    conn = create_connection()  # Створення підключення до бази даних
    cursor = conn.cursor()  # Створення курсора для виконання SQL-запиту
    cursor.execute("INSERT INTO alerts (symbol, price, chat_id) VALUES (?, ?, ?)", (symbol,
price, chat_id))  # Вставка нового алерта
    conn.commit()  # Збереження змін у базі даних
    conn.close()  # Закриття підключення до бази даних
```

```
def create_tables():
    """ Створення таблиць у базі даних (якщо ще не існують) """
    try:
        conn = sqlite3.connect('crypto_bot.db')  # Підключення до бази даних
        cursor = conn.cursor()  # Створення курсора для виконання запитів

        # Створення таблиці chat_ids (для зберігання chat_id користувачів)
        cursor.execute('''
        CREATE TABLE IF NOT EXISTS chat_ids (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            chat_id INTEGER UNIQUE
        );
        ''')
        conn.commit()  # Збереження змін у базі даних

        # Створення таблиці alerts (для зберігання алертів користувачів)
        cursor.execute('''
        CREATE TABLE IF NOT EXISTS alerts (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            symbol TEXT,
            price REAL,
            chat_id INTEGER,
            FOREIGN KEY(chat_id) REFERENCES chat_ids(id)
        );
        ''')
        conn.commit()  # Збереження змін у базі даних

        # Перевірка наявності стовпця created_at у таблиці alerts
        cursor.execute("PRAGMA table_info(alerts);")
        columns = cursor.fetchall()
```

```
        column_names = [column[1] for column in columns]  # Отримуємо імена стовпців

        if 'created_at' not in column_names:
            # Додаємо стовпець created_at (якщо його ще немає)
            cursor.execute("PRAGMA foreign_keys = OFF;")
            cursor.execute("ALTER TABLE alerts ADD COLUMN created_at TIMESTAMP;")
            cursor.execute("PRAGMA foreign_keys = ON;")
            conn.commit()  # Збереження змін у базі даних

            # Оновлюємо існуючі записи в alerts, додаючи поточний timestamp у створений стовпець
            cursor.execute("""
                UPDATE alerts
                SET created_at = CURRENT_TIMESTAMP
                WHERE created_at IS NULL;
            """)
            conn.commit()  # Збереження змін у базі даних

        # Створення таблиці price_history (для зберігання історії цін монет)
        cursor.execute('''
        CREATE TABLE IF NOT EXISTS price_history (
            symbol TEXT PRIMARY KEY,
            open_price REAL,
            last_checked DATETIME
        );
        ''')
        conn.commit()  # Збереження змін у базі даних

        print("Таблиці створено успішно!")

    except sqlite3.Error as e:
```

```
        print(f"Помилка: {e}")  # Виведення помилки при підключенні до бази даних або виконанні запиту
    finally:
        conn.close()  # Закриття підключення до бази даних


if __name__ == "__main__":  # Якщо цей файл є головним скриптом
    create_tables()  # Викликаємо функцію для створення таблиць у базі даних
```

Додаток А.4 Видалення алертів, старших за 48 годин, через SQL-функцію datetime

```
import sqlite3  # Імпорт бібліотеки для роботи з SQLite базою даних


DB_FILE = "crypto_bot.db"  # Назва файлу бази даних


def delete_old_alerts():
    """ Видаляє алерти, що не спрацювали протягом 48 годин """
    conn = sqlite3.connect(DB_FILE)  # Підключення до бази даних
    cursor = conn.cursor()


    cursor.execute("""
        DELETE FROM alerts
        WHERE created_at < datetime('now', '-48 hours')
    """)  # Видалення алертів, що старші за 48 годин
    conn.commit()  # Застосування змін до БД
    conn.close()  # Закриття з'єднання
    print("Старі алерти видалено успішно")


if __name__ == "__main__":
    delete_old_alerts()  # Виклик функції для видалення старих алертів
```

Додаток А.5 Оновлення стовпця created_at для існуючих алертів

```
import sqlite3  # Імпорт бібліотеки для роботи з SQLite базою даних


DB_FILE = "crypto_bot.db"  # Назва файлу бази даних


def update_alerts_created_at():
    """ Оновлює стовпець created_at для існуючих алертів """
    conn = sqlite3.connect(DB_FILE)  # Підключення до бази даних
    cursor = conn.cursor()


    cursor.execute("""
        UPDATE alerts
        SET created_at = CURRENT_TIMESTAMP
        WHERE created_at IS NULL
    """)  # Оновлюємо записи, де created_at ще не заповнено
    conn.commit()  # Застосування змін до БД
    conn.close()  # Закриття з'єднання
    print("Існуючі алерти оновлено з поточним часом")


if __name__ == "__main__":
    update_alerts_created_at()  # Виклик функції для оновлення алертів
```

Додаток Б. Скріншоти роботи бота

```
333
2025-05-20 17:01:06,109 - INFO - Перевірка алертів у фоновому режимі
2025-05-20 17:01:06,109 - INFO - Перевірка активних алертів
2025-05-20 17:01:06,111 - INFO - Підключено до бази даних crypto_bot.db
2025-05-20 17:01:06,114 - INFO - Перевірка алерта 19 для ETHUSDT на 2750.0$
2025-05-20 17:01:06,114 - INFO - Отримання ціни для ETHUSDT за URL: https://fapi.binance.com/fapi/v1/ticker/price?symbol=ETHUSDT
2025-05-20 17:01:06,933 - INFO - Поточна ціна ETHUSDT: 2474.74
2025-05-20 17:01:06,933 - INFO - Перевірка алерта 20 для ETHUSDT на 1.0$
2025-05-20 17:01:06,934 - INFO - Отримання ціни для ETHUSDT за URL: https://fapi.binance.com/fapi/v1/ticker/price?symbol=ETHUSDT
2025-05-20 17:01:07,076 - INFO - Запуск основного потоку
2025-05-20 17:01:07,078 - INFO - Запуск головної функції
222
2025-05-20 17:01:07,280 - INFO - HTTP Request: POST https://api.telegram.org/bot7567505791:AAE5Yqbd7gG9ydsx2_inJJmnS1Ln6rmh2Ts/getMe "HTTP/1.1 200 OK"
2025-05-20 17:01:07,313 - INFO - HTTP Request: POST https://api.telegram.org/bot7567505791:AAE5Yqbd7gG9ydsx2_inJJmnS1Ln6rmh2Ts/deleteWebhook "HTTP/1.1 200 OK"
2025-05-20 17:01:07,314 - INFO - Scheduler started
2025-05-20 17:01:07,314 - INFO - Application started
2025-05-20 17:01:07,659 - INFO - Поточна ціна ETHUSDT: 2474.95
Запит до API Binance, статус-код: 200
Запит до API Binance, статус-код: 200
2025-05-20 17:01:10,661 - INFO - Перевірка алертів у фоновому режимі
2025-05-20 17:01:10,661 - INFO - Перевірка активних алертів
2025-05-20 17:01:10,661 - INFO - Підключено до бази даних crypto_bot.db
2025-05-20 17:01:10,661 - INFO - Перевірка алерта 19 для ETHUSDT на 2750.0$
2025-05-20 17:01:10,661 - INFO - Отримання ціни для ETHUSDT за URL: https://fapi.binance.com/fapi/v1/ticker/price?symbol=ETHUSDT
2025-05-20 17:01:11,386 - INFO - Поточна ціна ETHUSDT: 2473.68
2025-05-20 17:01:11,386 - INFO - Перевірка алерта 20 для ETHUSDT на 1.0$
2025-05-20 17:01:11,386 - INFO - Отримання ціни для ETHUSDT за URL: https://fapi.binance.com/fapi/v1/ticker/price?symbol=ETHUSDT
2025-05-20 17:01:12,109 - INFO - Поточна ціна ETHUSDT: 2472.33
Запит до API Binance, статус-код: 200
Запит до API Binance, статус-код: 200
2025-05-20 17:01:15,111 - INFO - Перевірка алертів у фоновому режимі
2025-05-20 17:01:15,111 - INFO - Перевірка активних алертів
2025-05-20 17:01:15,111 - INFO - Підключено до бази даних crypto_bot.db
2025-05-20 17:01:15,111 - INFO - Перевірка алерта 19 для ETHUSDT на 2750.0$
2025-05-20 17:01:15,111 - INFO - Отримання ціни для ETHUSDT за URL: https://fapi.binance.com/fapi/v1/ticker/price?symbol=ETHUSDT
2025-05-20 17:01:15,836 - INFO - Поточна ціна ETHUSDT: 2472.33
2025-05-20 17:01:15,837 - INFO - Перевірка алерта 20 для ETHUSDT на 1.0$
2025-05-20 17:01:15,837 - INFO - Отримання ціни для ETHUSDT за URL: https://fapi.binance.com/fapi/v1/ticker/price?symbol=ETHUSDT
2025-05-20 17:01:16,564 - INFO - Поточна ціна ETHUSDT: 2472.33
Запит до API Binance, статус-код: 200
2025-05-20 17:01:17,422 - INFO - HTTP Request: POST https://api.telegram.org/bot7567505791:AAE5Yqbd7gG9ydsx2_inJJmnS1Ln6rmh2Ts/getUpdates "HTTP/1.1 200 OK"
Запит до API Binance, статус-код: 200
```

Рисунок Б.1 – Запуск бота та отримання повідомлення при зміні ціни на понад 1%

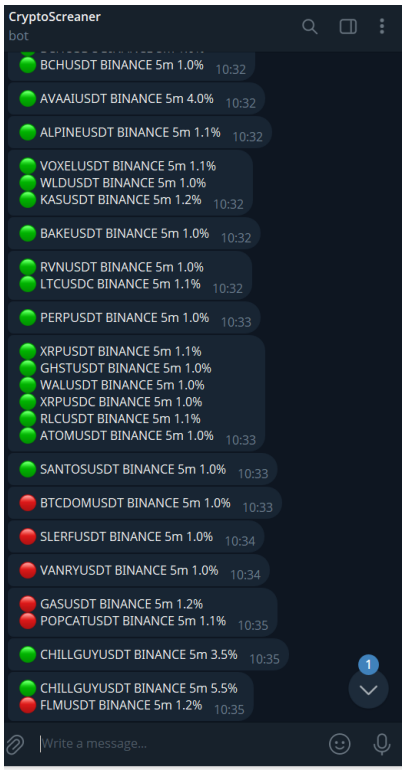

Рисунок Б.2 – Використання команди /addalert для додавання цінового алерта

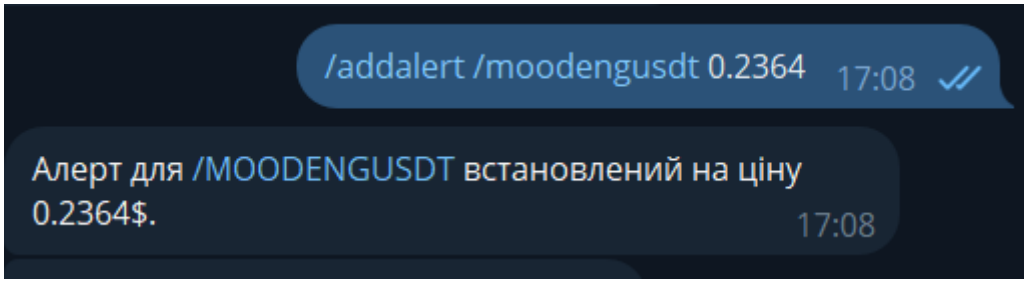

Рисунок Б.3 – Повідомлення про спрацювання алерта