ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «АВТОМАТИЗОВАНА СИСТЕМА ОНЛАЙН МОНІТОРИНГУ БІРЖОВИХ ЦІН КРИПТОВАЛЮТ НА ОСНОВІ МОВИ ПРОГРАМУВАННЯ PYTHON»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Олексій МАЩЕНКО*

Виконав:
здобувач вищої освіти Олексій МАЩЕНКО
група ІСДМ-61

Керівник: Оксана ТКАЛЕНКО
науковий ступінь, вчене звання к.т.н., доцент

Рецензент:
науковий ступінь, вчене звання _____
Ім’я, ПРІЗВИЩЕ

Київ – 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ

_____Каміла СТОРЧАК
«____» _____20___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ СТУДЕНТУ
Мащенку Олексію Юрійовичу
(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: «Автоматизована система онлайн моніторингу біржових цін криптовалют на основі мови програмування Python»

керівник кваліфікаційної роботи Оксана ТКАЛЕНКО, к.т.н., доцент
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «15» жовтня 2024 року № 320.

2. Строк подання кваліфікаційної роботи: 27 грудня 2024 року.

3. Вихідні дані до кваліфікаційної роботи:
Бібліотеки Python: PyQt6, SQLite, Plotly, PyQtGraph, yfinance, aiogram, lightweight-charts, requests, threading, binance, pandas.
Джерела даних: API криптовалютних бірж (Binance).
науково-технічна література з питань, пов'язаних з наукою про дані.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Методи збору, обробки та аналізу даних для моніторингу ринкових цін.

2. Дослідження підходів до автоматизованого моніторингу фінансових ринків.
3.Розробка та впровадження системи онлайн-моніторингу біржових цін криптовалют із використанням актуальних технологій та мови програмування Python.

5. Перелік ілюстративного матеріалу: *презентація*
1. Приклади систем моніторингу
2. Принцип роботи API.
3. Принцип веб-сокет з'єднання.
4. Робота з даними та типи даних.
5. Дизайн графічного інтерфейсу.
6. Архітектура системи.
7. Приклади використання системи:.
8. Демонстрація роботи системи.

6. Дата видачі завдання: 15 жовтня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	15.10 – 23.10.24	
2	Дослідження методів збору, обробки та аналізу даних в моніторингових системах	24.10 – 01.11.24	
3	Дослідження підходів до автоматизації моніторингу фінансових ринків	02.11 – 05.11.24	
4	Проектування системи онлайн моніторингу	06.11 – 14.11.24	
5	Розробка програмного забеспечення з використанням актуальних технологій	15.11 – 31.11.24	
6	Тестування системи онлайн моніторингу біржових цін криптовалют	02.12 – 11.12.24	
7	Оформлення роботи: вступ, висновки, реферат	12.12 – 20.12.24	
8	Розробка демонстраційних матеріалів	21.12 – 25.12.24	

Здобувач вищої освіти _____ Олексій МАЩЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник роботи
кваліфікаційної роботи _____ Оксана ТКАЛЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 80 сторінок, 44 рисунка, 2 таблиці, 30 джерел.

Мета роботи – розробка автоматизованої системи моніторингу біржових цін криптовалют із використанням сучасних технологій збору, обробки та візуалізації даних для підвищення ефективності аналізу ринку та спрощення прийняття рішень.

Об'єкт дослідження – процеси моніторингу, аналізу та візуалізації ринкових даних у системах обробки інформації.

Предмет дослідження – технології та інструменти моніторингу, включаючи API, веб-сокети, бази даних, засоби візуалізації та алгоритми автоматизації аналізу даних.

Короткий зміст роботи: Досліджено технології та методи збору й обробки даних у сучасних системах моніторингу. Розглянуто принципи роботи API та web-socket з'єднань для отримання ринкових даних у реальному часі. Проаналізовано роль баз даних у збереженні великих обсягів інформації, обґрунтовано вибір SQLite. Розроблено графічний інтерфейс на базі PyQt6 із використанням CSS для зручного налаштування параметрів системи та візуалізації ринкових графіків. Створено Telegram-бота з використанням бібліотеки aiogram для надсилання сповіщень про зміну ринкових умов. Розроблено функціональну систему, яка збирає, обробляє й аналізує дані з бірж криптовалют, а також автоматично інформує користувачів про виконання заданих умов. Систему була протестована на реальних даних.

КЛЮЧОВІ СЛОВА: СИСТЕМА МОНІТОРИНГУ, КРИПТОВАЛЮТА, БАЗА ДАНИХ, ВЕБ-СОКЕТ, API, PYTHON, ГРАФІЧНИЙ ІНТЕРФЕЙС, СПОВІЩЕННЯ, ФІНАНСОВИЙ АНАЛІЗ.

ABSTRACT

The text part of the master's qualification work: 80 pages, 44 pictures, 2 tables and 30 sources.

The purpose of the work – to develop an automated system for monitoring cryptocurrency exchange rates using modern technologies for data collection, processing, and visualization to improve market analysis efficiency and simplify decision-making.

Object of research – processes of monitoring, analyzing, and visualizing market data in information systems.

Subject of research – monitoring technologies and tools, including API, web sockets, databases, visualization tools, and algorithms for automating data analysis.

Summary of the work: The research explores technologies and methods for data collection and processing in modern monitoring systems. The principles of API and web socket connections for obtaining real-time market data are considered. The role of databases in storing large volumes of information is analyzed, and the choice of SQLite is justified. A graphical interface based on PyQt6 with CSS is developed for convenient system parameter configuration and market chart visualization. A Telegram bot is created using the Aiogram library to send notifications about market condition changes. A functional system has been developed to collect, process, and analyze data from cryptocurrency exchanges and automatically inform users when predefined conditions are met. The system was tested on real data.

KEYWORDS: MONITORING SYSTEM, CRYPTOCURRENCY, DATABASE, WEB SOCKET, API, PYTHON, GRAPHICAL INTERFACE, NOTIFICATIONS, FINANCIAL ANALYSIS.

ЗМІСТ

ВСТУП 9
РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ СИСТЕМ ОНЛАЙН МОНІТОРИНГУ 12
1.1 Потреба в автоматизованих системах моніторингу 12
1.2 Огляд порталу CoinMarketCap для моніторингу криптовалют 13
1.3 Огляд аналітичної платформи TradingView 15
1.4 Огляд інформаційної системи CoinGecko для моніторингу криптовалют 19
1.5 Огляд системи моніторингу CoinTracking 21
1.6 Огляд біржі Binance 22
РОЗДІЛ 2 АНАЛІЗ ТЕХНОЛОГІЙ ТА ВИБІР ІНСТРУМЕНТІВ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ МОНІТОРИНГУ КРИПТОВАЛЮТ 26
2.1 Клієнт-серверна архітектура 26
2.2 Технологія API 31
2.3 Технологія веб-сокетів 34
2.4 Технології для створення графічного інтерфейсу 36
2.5 Технології баз даних 40
2.6 Вибір мови програмування 44
РОЗДІЛ 3 РОЗРОБКА СИСТЕМИ АВТОМАТИЗОВАНОГО ОНЛАЙН МОНІТОРИНГУ ЦІН КРИПТОВАЛЮТ 47
3.1 Постановка задачі 47
3.2 Проектування системи 50
3.3 Вибір фреймворків для розробки системи 55
3.4 Дизайн графічного інтерфейсу 60
3.6 Використання бази даних SQLite 65
3.7 Проблеми сумісності 71
3.8 Розробка головних компонентів системи 72
3.9 Тестування системи 76
3.10 Можливості застосування та покращення системи 78
ВИСНОВКИ 80
ПЕРЕЛІК ПОСИЛАНЬ 82
ДОДАТОК А 85
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація) 1

ВСТУП

З розвитком інформаційних технологій зростає інтерес до цифрових активів, біржових операцій і криптовалют. Це зумовлює підвищений попит на системи, здатні забезпечувати оперативний моніторинг і аналіз ринкових цін у реальному часі.

Зазвичай фінансові біржі надають користувачам всю необхідну інформацію, про вибраний торговий актив. Це може бути просто графік ціни, який містить історичні дані та іншу корисну інформацію, обсяги торгів за певний період, глибину ринку, де детально описується обсяг заявок на покупку та продаж. Також біржі надають можливість користувачам вмикати різні індикатори та користуватися іншими корисними функціями. Однак обсяг інформації навіть для досвідченого торгового аналітика надмірний і активний моніторинг навіть однієї торгової пари потребує постійної уваги та концентрації.

Актуальність теми: автоматизовані системи онлайн моніторингу здатні вирішити цю проблему. Такі системи можуть одночасно працювати з великим обсягом даних, обробляти їх, відсіювати непотрібне, проводити агрегацію даних, перевіряти виконання чи невиконання умови та при потребі відправляти повідомлення чи автоматично виконувати певну дію, що значно економить час для користувача та дає можливість працювати з великим обсягом ринкових даних, обираючи з нього тільки те, що потрібно.

Автоматизовані системи широко застосовуються в даній галузі та активно розвиваються.

Метою даної роботи є розробка власної автоматизованої системи для онлайн-моніторингу біржових цін криптовалют із використанням сучасних веб-технологій. У межах роботи здійснено аналіз існуючих подібних систем, досліджено їхні підходи та реалізації, а також створено програмний продукт, що демонструє принципи роботи таких інструментів.

Об'єкт дослідження – процеси моніторингу, аналізу та візуалізації ринкових даних у системах обробки інформації.

У сучасних системах моніторингу використовуються різні технології для збору, обробки та аналізу даних у реальному часі. Залежно від специфіки та обсягу даних, використовуються такі інструменти як API, веб-сокети, інструменти для зберігання великих обсягів даних – бази даних, інструменти для візуалізації даних та створення графічного інтерфейсу, якщо це необхідно, бо це допоможе користувачеві швидко розібратися в тому як налаштувати систему. А також інструменти для сповіщень користувача. Це можуть бути різні месенджери, що надають можливість розробникам використовувати свій системний інтерфейс при розробці власного програмного продукту, так і, наприклад, за допомогою електронник листів чи іншими способами в залежності від вимог та поставлених задач.

Для досягнення поставленої мети необхідно виконати наступні *завдання*:

- Дослідити технології збору та обробки даних, включаючи API та web-socket для підключення до зовнішніх сервісів і отримання ринкових даних у реальному часі.
- Проаналізувати різні доступні типи баз даних для зберігання великих обсягів даних.
- Розробити графічний інтерфейс користувача за допомогою PyQt6 і CSS для зручної візуалізації даних і налаштування параметрів системи.
- Створити систему сповіщень за допомогою Telegram-бота з бібліотекою aiogram для оперативного інформування користувачів про зміни цін.
- Провести тестування і оцінити ефективність розробленої системи на реальних даних біржі та скласти звіт.

Практичне значення роботи полягає в розробці автоматизованої системи для онлайн моніторингу біржових цін криптовалют, яка дозволяє ефективно збирати, обробляти та аналізувати ринкові дані. Ця система буде мати графічний інтерфейс, який можна використати для налаштування потрібної торгової пари та умови, при якій буде виконуватися вибрана дія, наприклад, буде приходити повідомлення на телеграм бота з відповідним текстом та сповіщати користувача про виконання умови в режимі онлайн. Така система забезпечує користувачів

можливістю відслідковувати актуальні зміни на ринку, можливість застосовувати різні аналітичні індикатори та отримувати сповіщення про важливі події, що може значно знизити необхідність постійно перевіряти термінал біржі на певні змін.

Розроблений інструмент орієнтований як на досвідчених трейдерів і аналітиків, так і на початківців. Він допомагає користувачам зменшити витрати часу на аналіз, знизити ризик пропуску важливих змін і підвищити ефективність прийняття рішень. Система також має потенціал для інтеграції в автоматизовані торгові рішення, що робить її корисною в різних сценаріях використання.

Апробація результатів магістерської роботи:

1. Мащенко О.Ю - «Система моніторингу біржових цін криптовалют». Тези доповіді на II Всеукраїнській науково-технічній конференції «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу». – Київ, 18 листопада 2024 року.
2. Мащенко О.Ю - «Веб-технології в системах моніторингу фінансових даних». Тези доповіді на VII Всеукраїнській науково-технічній конференції «Комп’ютерні технології: інновації, проблеми, рішення». – Київ, 2-3 грудня 2024 року.

1 ОГЛЯД ІСНУЮЧИХ СИСТЕМ ОНЛАЙН МОНІТОРИНГУ

1.1 Потреба в автоматизованих системах моніторингу

Автоматизовані системи моніторингу є важливою складовою сучасної інфраструктури для збору, обробки та аналізу даних в реальному часі. Вони використовуються в багатьох галузях, і звичайно в такій як фінансові технології.

Ці системи дозволяють оптимізувати процеси, зменшити ризики та покращити ефективність роботи організацій чи конкретних користувачів. Якщо розглядати конкретно ринок криптовалют то саме в цій галузі найбільше потрібні подібні системи моніторингу в системі реального часу, адже зміни відбуваються постійно і зазвичай суттєво, що вимагає від фінансового аналітика постійно слідкувати за ринком. Ринок криптоактивів характеризується високою волатильністю (постійним коливанням ціни з високою амплітудою). Для криптовалюти ситуація, коли ціна за годину виросла на п’ять відсотків а вже через дві впала на десять абсолютно нормально, тому трейдер повинен постійно слідкувати за цим і не пропустити потрібну йому ситуацію. На відміну від класичних бірж, де торгують акціями компаній, фінансовими облігаціями, валютами чи товарними ресурсами, криптобіржі не мають часу відкриття чи закриття і працюють цілодобово, що практично унеможливлює цілодобовий контроль з боку користувача. Автоматизована система моніторингу може працювати постійно, оброблюючи великі обсяги інформації, і або відправляти сповіщення, або вона можу бути частиною більшої системи і подавати сигнал на виконання певної операції.

Криптовалютні біржі генерують величезну кількість даних в реальному часі, що включають ціни, обсяги торгів, зміну котирувань, новини, а також інформацію про угоди та транзакції. Для трейдерів важливо мати доступ до всіх цих даних в зручному форматі для прийняття швидких рішень. Автоматизовані системи моніторингу можуть обробляти ці дані, проводити агрегацію, структурувати їх і надавати користувачам зручні інтерфейси для перегляду та аналізу. Такі системи часто є складовою більших масштабованих систем обробки даних та прийняття

рішення, що використовуються для торгівлі. Такі система мають вбудовані інструменти для технічного аналізу, включаючи різноманітні графіки, осцилятори, трендові лінії та інші показники, що допомагають формувати торгові стратегії, але без інтегрованої системи моніторингу вони не можуть отримувати дані в режимі онлайн і стають неефективними. Автоматизовані системи зменшують вплив людського фактору і зменшують вірогідність помилки, а також значно економлять час, що дозволяє зосередитися користувачу на інших важливих аспектах торгівлі на біржі.

Подібні системи широко застосовуються вже зараз та активно розвиваються, збільшуючи свої можливості, розширюючи наявних функціонал. Більшість з них мають зручний у використанні та інтуїтивно зрозумілий графічний інтерфейс, що дозволяє швидко розібратися навіть новачку. Деякі системи мають свій безкоштовний варіант, але більшість доступні по платній підписці.

1.2 Огляд порталу CoinMarketCap для моніторингу криптовалют

CoinMarketCap — це один із найбільших та найпопулярніших порталів для моніторингу криптовалют, що надає аналітичні дані про ціни, обсяги торгів, ринкову капіталізацію та інші ключові показники криптовалют. Він був створений у 2013 році, і відтоді став основним джерелом інформації для трейдерів, інвесторів та користувачів, які цікавляться ринком криптовалют. CoinMarketCap забезпечує доступ до різноманітних даних про понад 8 000 криптовалют і використовує для цього дані з різних доступних фінансових бірж, та надає доступ до широкого набору функцій.

Платформа надає актуальні дані про ціни більшості популярних криптовалют. Ціни оновлюються у реальному часі, що важливо для трейдерів, які працюють з активами, такими як криптовалюта.

CoinMarketCap пропонує різні технічні показники типи графіків для відображення змін у цінах та інших ринкових даних. Також користувачі можуть отримати історичну інформацію про ціни, обсяги торгів, ринкову капіталізацію

криптовалют, що допомагає оцінити тренди на ринку за минулі періоди. Також ця інформаційна система має функціонал для створення особистих портфелів криптовалют, що дозволяє користувачам відстежувати власні інвестиції та отримувати сповіщення про зміни в ринкових умовах.

Ця платформа надає доступ до API (має власний програмний інтерфейс), що дозволяє розробникам інтегрувати дані з даного сайту у свої додатки чи системи моніторингу, що дає змогу автоматизувати процес збору і аналізу даних про ринок криптовалют.

Є функціонал, який надає користувачам можливість слідкувати за новинами, що впливають на ринок криптовалют, а також отримувати прогнози про майбутні зміни вартості активів.

Телеграм-сповіщення також є можливість використовувати для отримання даних з даного інформаційного порталу. Тож CoinMarketCap також підтримує інтеграцію з Telegram, що дає змогу користувачам отримувати повідомлення про зміни на ринку прямо в месенджері.

Приклад роботи телеграм бота CoinMarketCap Price Bot:

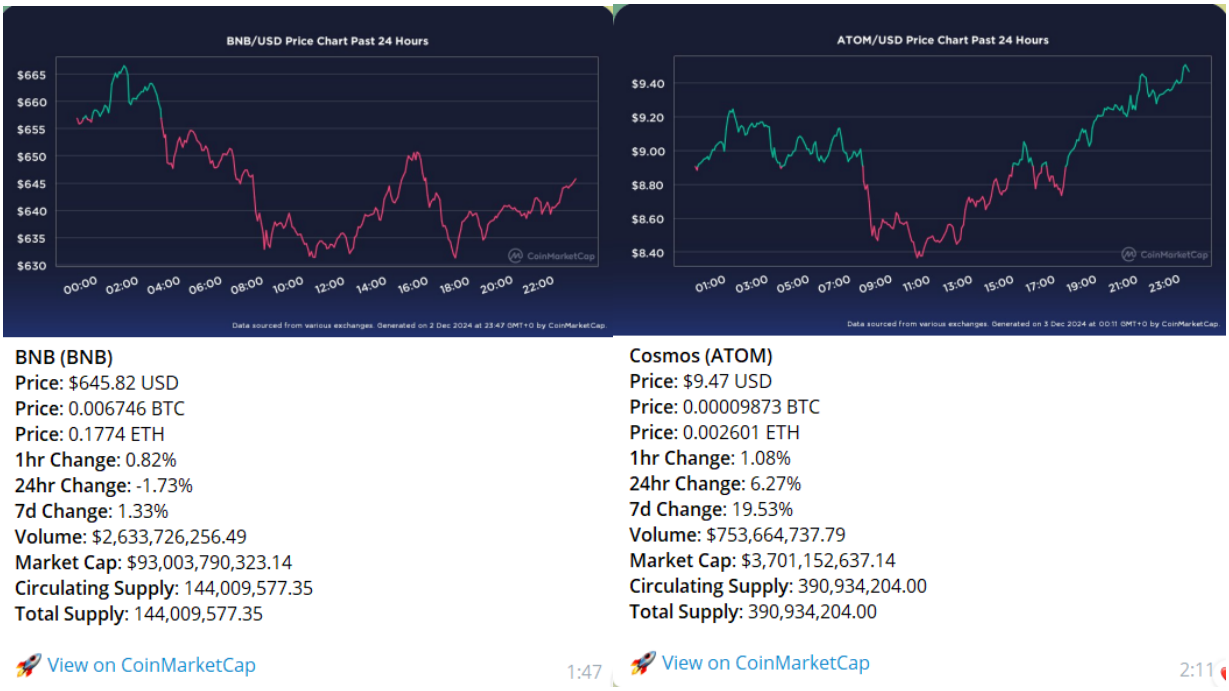


Рис. 1.1 Приклад відповіді від бота CoinMarketCap

Він має достатньо обмежений функціонал, але доступний безкоштовно. Як видно на рис. 1.1 бот присилає інформацію про конкретну обрану торгову пару, а також актуальну розгорнуту інформацію про нього. Тут вказана ціна в доларах, ціна в біткоїні та ефірі, відсоток змін за останній час та за останню добу. Також такі показники як об’єм капіталізації активу та фактичну фінансову підтримку.

Можна також підписатися на щогодинну розсилку подібних повідомлень, але тут не можна встановити якісь умови при яких повідомлення прийде раніше. Наприклад, немає можливості встановити рубіжну ціну при якій прийде сповіщення, або неможна налаштувати реакцію на зміну ціни в відсотках чи іншим способом. Тому як для автоматизованої системи цей бот використовувати не дуже зручно, а сам портал потребує постійної уваги, щоб слідкувати за змінами на ринку.

Проте є важливим інструментом, оскільки надає зручний доступ до величезного обсягу даних в реальному часі, а також має власний інтерфейс API, що дає можливість розробникам використовувати ці ресурси в розробці власних додатків.

1.3 Огляд аналітичної платформи TradingView

TradingView — це одна з найпопулярніших платформ для технічного аналізу фінансових ринків, зокрема криптовалют. Вона була створена у 2011 році і активно розвивалася, нарощуючи власний функціонал, та ставала все популярнішою серед трейдерів та фінансових аналітиків, адже пропонує широкий спектр можливостей для технічного аналізу, побудови графіків, аналізу ринку та створення автоматизованих стратегій. Для цієї задачі навіть було створену власну мову для створення скриптів і стратегій — Pine Script. Це спеціалізована мова, яка оптимізована для побудови фінансових індикаторів і торгових стратегій. Платформа побудована як вебзастосунок із використанням хмарних технологій, що забезпечує доступність з будь-якого пристрою без необхідності встановлення

програмного забезпечення. Вона інтегрує клієнт-серверну архітектуру, яка дозволяє проводити розрахунки на стороні сервера.

TradingView підтримує інтерактивні графіки, які легко налаштовуються. Користувачі можуть додавати індикатори, трендові лінії та інші візуальні елементи прямо на графіку.

Платформа підтримує одночасну обробку великого обсягу даних. Вона інтегрується з сотнями брокерських сервісів, включаючи криптовалютні біржі, і пропонує доступ до глобальних ринків. Цей сервіс можна використовувати як систему моніторингу. Хоч ця інформаційна система в основному зосереджена та технічному аналізі, але також надає доступ до великого обсягу фінансової інформації та певні можливості автоматизації. Наприклад, якщо відкрити сайт tradingview.com і знайти потрібну торгову пару то можна налаштувати сповіщення. Хоча воно буде активним тільки якщо не закривати браузер.

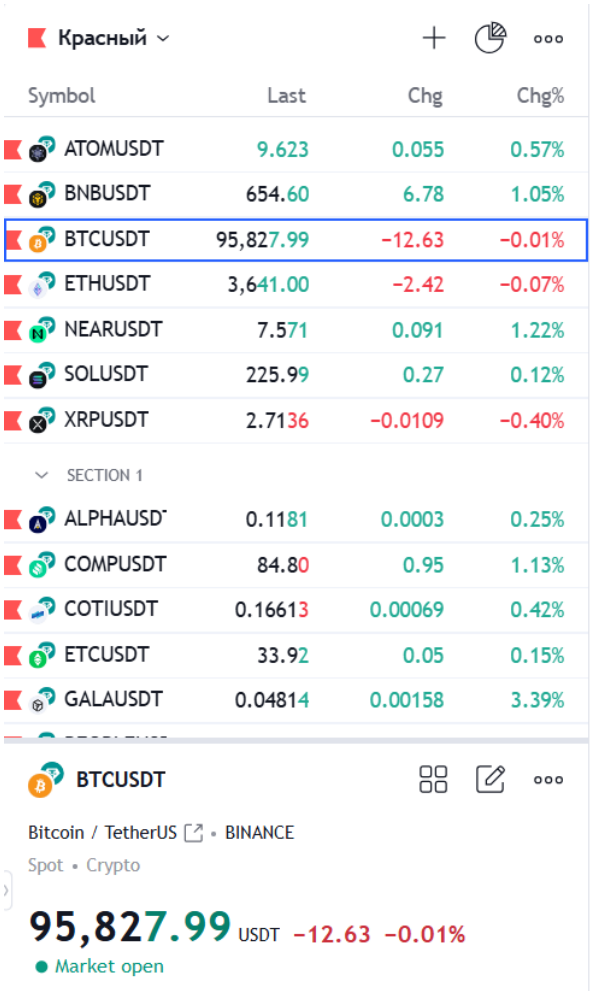


Рис. 1.2 Інформаційна стрічка на порталі TradingView

Таким чином можна отримувати ціни в режимі онлайн.

Якщо авторизуватися на даному порталі то можна отримати можливість персоналізувати інформаційну стрічку, в якій буде відображено обрані пари криптовалютних активів, їх поточну ціну, яка підтягується з обраної біржі, а також зміну їх ціни в абсолютному і відсотковому значенні. Обрані пари можна позначити прапорцем, щоб було легше знайти в списку.

TradingView має власну екосистему, яка окрім кросплатформленного веб-сервісу має також мобільний додаток, що і надає вже більші можливості по автоматизації онлайн моніторингу. Цей додаток можна знайти в Google Play, він безкоштовний. Мобільний застосунок TradingView надає можливість отримувати push-повідомлення.

Приклад використання програми:

Рис. 1.3 Приклад роботи мобільного застосунку TradingView

Пуш-повідомлення (push notifications) - це короткі автоматичні повідомлення, які надсилаються на пристрої користувачів (найчастіше на мобільні пристрої) від

мобільних додатків, веб-сайтів або систем. Вони з’являються у вигляді спливаючих вікон, банерів або звукових сигналів, навіть коли користувач не взаємодіє безпосередньо з програмою або сайтом. Саме такі повідомлення і забезпечують автоматизований процес моніторингу цін криптовалют, так як все що потрібно зробити це встановити умову та почекати доки вона виконається і прийде повідомлення.

Приклад push-повідомлення:

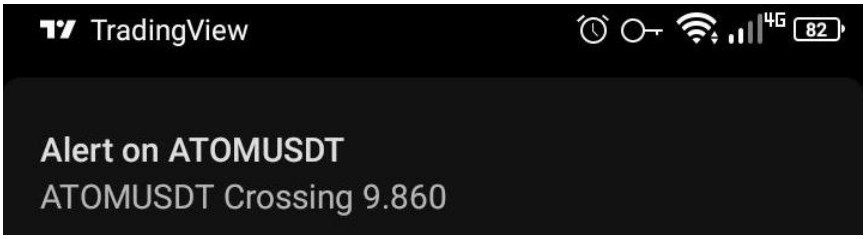


Рис. 1.4 Приклад push-повідомлення

Такі повідомлення мають назву активу, за яким слідкували та умову, що була виконана. Вони будуть приходити і тоді, коли програма працює у фоновому режимі, а також можна використовувати десктопний застосунок і налаштування будуть синхронізовані, якщо користувач і там і там авторизований.

Окрім того є можливість використовувати віджет для головного екрану мобільного телефону.

TradingView розроблено на JavaScript та використовує сучасні фреймворки для роботи з графікою. Портал надає API та SDK (Software Development Kit), які дозволяють інтегрувати функціонал платформи у сторонні програми та веб-сайти. SDK Charting Library – це набір інструментів і бібліотек, призначений для розробки програмного забезпечення з використанням даних та функціоналу даної платформи платформи

Документація TradingView доступна для розробників із прикладами та інструкціями, що дозволяє створювати кастомізовані системи під вузьконаправлені задачі.

1.4 Огляд інформаційної системи CoinGecko для моніторингу криптовалют

CoinGecko — це популярний портал для моніторингу криптовалют, який надає детальну інформацію про різноманітні криптоактиви, ринки та їхню динаміку в реальному часі. Платформа була створена і доступна з 2014 року, швидко здобула популярність завдяки своєму відкритому доступу до даних і детальній аналітиці.

Ця платформа схожа на CoinMarketCap, але має трішки менший об’єм доступної інформації і підтримує меншу кількість торгових інструментів, проте не має набридливої реклами, що зробило цей портал достатньо популярним.

Основний веб-портал розроблений для забезпечення швидкого доступу до даних користувачів, зокрема через функціонал пошуку, фільтрації та сортування. Застосовуються сучасні веб-технології для побудови інтуїтивного інтерфейсу. Має мікросервісну архітектуру, що дозволяє обробляти значні обсяги запитів і взаємодіяти з різними джерелами даних. Дані про ринки криптовалют, транзакції та ціни збираються з різних бірж і узагальнюються для аналітики. Всього CoinGecko взаємодіє з понад 400 бірж і торгових майданчиків, включаючи і криптобіржі. Для розробників портал пропонує детальну документацію API, яка містить приклади використання та опис функцій. Наприклад, за допомогою API можна створювати власні панелі моніторингу криптовалют або інтегрувати дані в інші фінансові системи.

Цей портал не має власного SDK, тому доведеться взаємодіяти з її даними через HTTP-запити (Hypertext Transfer Protocol) та розробляти методи для взаємодії та обробки отриманої від API інформації, яка прийде або в JSON (JavaScript Object Notation) або в форматі XML (Extensible Markup Language).

Так само, як і попередній портал, цей має власний мобільний додаток. Мобільний застосунок CoinGecko доступний для платформ Android та iOS, дозволяючи користувачам ефективно відстежувати ціни криптовалют,

переглядати графіки та налаштовувати сповіщення. Застосунок надає доступ до понад 6,000 криптовалют та NFT, забезпечуючи актуальні дані в режимі реального часу. Тому його можна активно використовувати як автоматизовану систему онлайн моніторингу. Для встановлення на Android потрібно мати систему версії 6.0 і вище.

Так само як і TradingView надає можливість налаштувати список валют, які потрібно відстежувати.

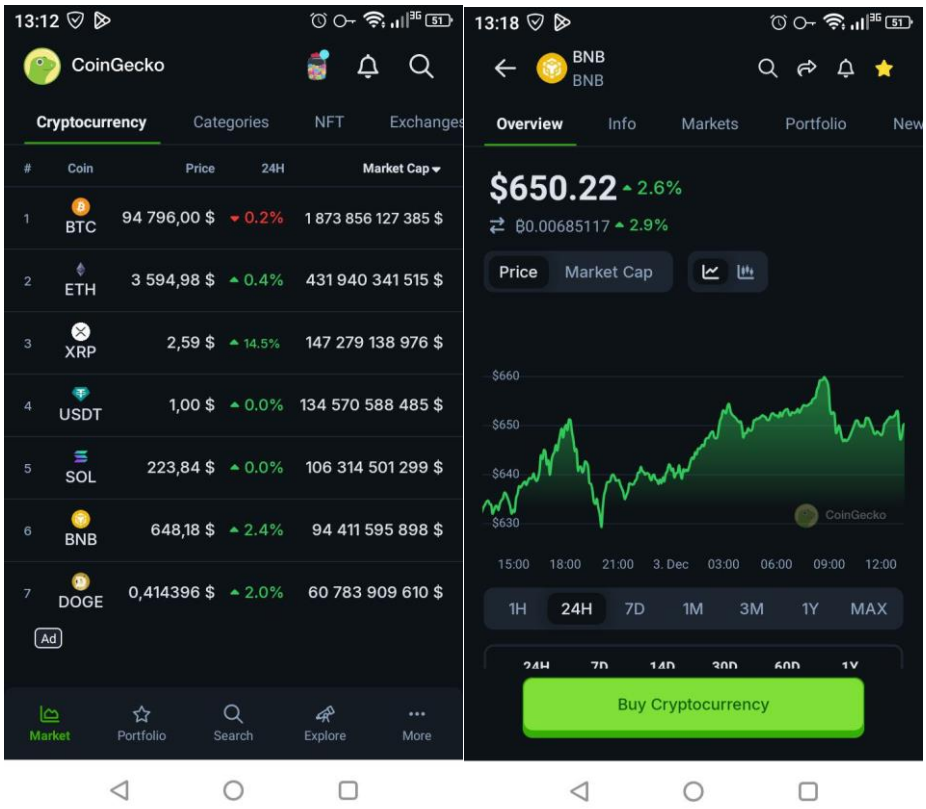


Рис. 1.5 Приклад роботи мобільного застосунку CoinGecko

В списку інформації можна отримати дані про ціну, зміну в відсотках за останні 24 години та ринкову капіталізацію. Якщо перейти до конкретної торгової пари то можна відкрити графік і отримати додаткову історичну інформацію про коливання ціни за вибраний період.

Навідміну від застосунку TradingView тут немає можливості використовувати графік для технічного аналізу (будувати трендові лінії, сітки Фібоначчі чи робити інші позначки на графіку). Але в CoinGecko простіше налаштувати сповіщення і для цього відведено окреме вікно програми.

Тут є можливість вибрати криптовалюту зі списку доступних та встановити цільову ціну, при досягненні якої прийде push-повідомлення від додатку. Також можна скористатися додатковими кнопками швидкого заповнення і поставити ціль на 5, 10 та 20 відсотків коливання ціни від поточної. Можна налаштувати це як одноразове повідомлення та як багаторазове.

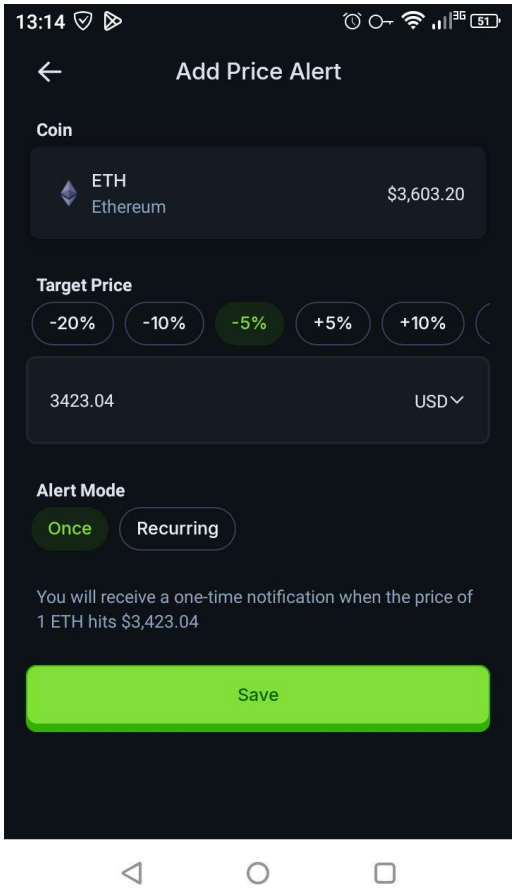


Рис. 1.6 Налаштування оповіщення в CoinGecko

Отже додаток CoinGecko надає більші можливості по налаштуванню автоматизованого моніторингу.

1.5 Огляд системи моніторингу CoinTracking

CoinTracking – це популярна платформа для управління криптовалютами, яка дозволяє аналізувати портфель, відстежувати угоди, обчислювати податки та

забезпечувати загальну аналітику. Вона створена для трейдерів та інвесторів, щоб спростити процес управління активами на основі даних із бірж та блокчейн-мереж.

Ця інформаційна система загалом розроблена для управління власними активами, має платний функціонал, що дозволяє застосовувати потужні інструменти для аналітики та ефективному розподілу власних інвестицій і надає також можливість моніторингу змін цін на крипторинку.

Якщо створити власний обліковий запис на цьому порталі то є можливість за допомогою API ключів приєднати свій профіль на біржі і отримати доступ до статистики власних угод і наявних в даний момент активів. Можна налаштувати мобільний додаток як систему моніторингу власних активів та слідкувати за змінами цін за допомогою повідомлень. Цей портал підтримує інтеграцію з багатьма криптобіржами і дозволяє здійснювати операції на них за допомогою власного програмного інтерфейсу.

CoinTracking не має власного SDK, але надає розробнику доступ до власного API. З його допомогою можна завантажувати дані про транзакції та отримувати цінову інформацію в реальному часі.

Якщо порівнювати з попередніми системами такими як TradingView чи CoinMarketCap то ця система зосереджена більше на моніторингу власних активів і допомагає користувачу слідкувати за змінами у вартості інвестицій. Таким чином система автоматизованого онлайн моніторингу інтегрована в більш функціональну систему, яка виконує вже більший набір функцій.

1.6 Огляд біржі Binance

Binance – одна з найбільших криптовалютних бірж у світі, заснована в 2017 році. Вона надає можливість торгівлі цифровими активами із доступом до спотових, маржинальних, ф'ючерсних та опціонних ринків.

Спотовий ринок - це фінансовий ринок, на якому торгівля активами (наприклад криптовалютами) відбувається з негайним розрахунком і поставкою, тобто миттєво.

Маржинальний ринок – тут використовується кредитне плече, щоб збільшити об’єм продукції і тим самим збільшити дохід і ризик на операцію.

Ф'ючерсний ринок - угоди укладаються для виконання в майбутньому за зафіксованою ціною, це дає можливість заробляти як на рості так і на падінні ціни і є можливість використовувати кредитне плече.

Біржа відома своїм великим обсягом ліквідності, високою швидкістю обробки транзакцій і широким функціоналом. Binance побудована на масштабованій хмарній інфраструктурі, яка дозволяє обробляти понад 1.4 мільйона транзакцій на секунду. В основі її роботи лежить мікросервісна архітектура, API-орієнтованість і багатошарова система захисту даних. Біржа підтримує інтеграцію з різними фінансовими системами, та пропонує API для доступу до різних функцій, а також доступ до ринкових даних у реальному часі. API підтримує REST-запити та WebSocket для отримання стрімінгових даних (даних в реальному часі).

Binance надає кілька SDK для розробників, що дозволяють безшовну інтеграцію з їх платформою. Ці SDK охоплюють різні функції, такі як криптовалютна торгівля, управління гаманцями та Web3 функціональність. Це полегшує доступ до всіх можливостей біржі і дозволяє створювати торгові боти для автоматизації транзакцій, а також автоматизовані системи моніторингу.

SDK Binance для мови програмування Python (python-binance) дозволяє розробникам взаємодіяти з біржею Binance через їх API для доступу до різноманітних функцій. Таких як вибірка історії цін криптовалют, отримання інформації про торгові пари, цінові котирування в реальному часі, обсяги торгів, а також доступ до інформації про ордери та глибину ринку. Є можливість здійснювати торгові операції на біржі, таких як розміщення лімітних, ринкових та стоп-ордерів, що дає можливість розробникам створювати повноцінні торгові системи.

Також фреймворк python-binance надає доступ до управління аккаунтами. Перевірка балансу, зняття коштів, переведення активів на інші гаманці та дає доступ до історії транзакцій.

Цей SDK також має інтеграцію з технологією WebSocket. Це дає можливість встановлювати з’єднання використовуючи відповідні методи цієї бібліотеки. Цей функціонал корисно застосовувати, коли потрібно розробити систему моніторингу з високою частотою оновлення даних. Біржа Binance надає технічну документацію та підтримку, що значно спрощує розробку програмного забезпечення. Саме тому я обрав цю біржу для створення власної системи моніторингу.

Існують неофіційні Telegram-боти для інтеграції з Binance, такі як Binance Trade Telegram Bot. Ці боти дозволяють отримувати оновлення про ринок, керувати портфелем та здійснювати базові торгові операції прямо з месенджера. Вони зазвичай вимагають налаштування API-ключів.

Мобільний застосунок Binance доступний для Android та iOS. Він включає всі основні функції доступні на біржі, а також є можливість використовувати автоматичні оповіщення.

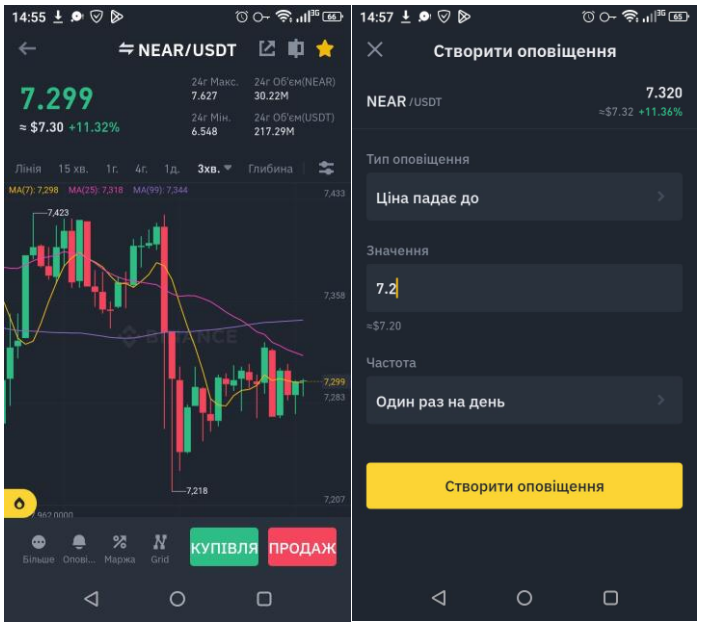


Рис. 1.7 Приклад використання мобільного застосунку Binance

Цей додаток має більші можливості по керування власними криптоактивами та можливістю моніторингу цін ніж ті, що були розглянуті до цього. Також вибір умов сповіщення більший. І цей додаток має українську локалізацію.

Binance пропонує потужні інструменти для трейдерів та розробників, забезпечуючи широкий функціонал і надійність для роботи з криптовалютами.

2 АНАЛІЗ ТЕХНОЛОГІЙ ТА ВИБІР ІНСТРУМЕНТІВ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ МОНІТОРИНГУ КРИПТОВАЛЮТ

2.1 Клієнт-серверна архітектура

Клієнт-серверна архітектура стала основою для багатьох сучасних комп'ютерних систем. Її почали використовувати ще в 1960-х роках, коли з'явилася потреба у віддаленому доступі до ресурсів комп'ютера. Одним з перших моментів у розвитку цієї архітектури була поява концепції "термінального обслуговування", коли для доступу до центрального комп'ютера використовували прості термінали, які фактично не мали потужностей для обробки даних, але могли подавати запити на сервер. Це дозволило обчислювальним машинам ефективніше використовувати свої апаратні ресурси, а користувачам - взаємодіяти з ними на відстані.

В 1980-90-х роках, коли розвиток персональних комп'ютерів набрав обертів, стало очевидним, що з'являється потреба в нових архітектурах, що дозволяють більш ефективно розподіляти навантаження. Це призвело до популяризації архітектури "клієнт-сервер", де сервери виконували обробку даних, а клієнти лише взаємодіяли з користувачем, показуючи йому результати. У цей час стали розвиватися різноманітні серверні технології (наприклад, для баз даних), що дозволило досягти ефективнішого використання ресурсів.

Сучасна клієнт-серверна архітектура є основною моделлю організації комп'ютерних мереж, яка розподіляє функціональність між клієнтами і серверами. У такій архітектурі сервери надають ресурси або послуги, в той час як клієнти ініціюють запити до серверів, отримуючи від них необхідні дані або сервіси.

Основні компоненти клієнт-серверної архітектури:

- Клієнт - пристрій або програма, що ініціює запит до сервера. Клієнт може бути простим (наприклад, веб-браузер) або складним (наприклад, спеціалізоване програмне забезпечення для бізнес-аналізу).

- Сервер - програма або система, що приймає запити від клієнтів і надає необхідні ресурси або послуги. Сервери зазвичай обробляють ці запити, виконують складні обчислення і повертають відповіді.

На початку 2000-х років технології розподілених обчислень, а також веб-сервіси і мобільні застосунки, сприяли еволюції клієнт-серверної архітектури. Вони стали більш гнучкими, дозволяючи використовувати як тонкі, так і товсті клієнти, що залежало від конкретних вимог до системи.

Поширення доступного інтернету і розробка великої кількості веб-додатків значно підвищили важливість тонких клієнтів, оскільки браузери стали основним інтерфейсом для користувачів. Веб-сервіси почали займати важливе місце в підприємницьких процесах, і через це сервери стали обробляти величезну кількість запитів одночасно, що вимагало нових підходів до побудови архітектури.

- Тонкий клієнт - це клієнт, який має мінімальний обсяг локальної обробки даних. Всі складні обчислення і збереження даних виконуються на сервері. Тонкі клієнти, зазвичай, є простими пристроями або програмами, що не потребують потужних ресурсів. Зазвичай цей тип клієнта підходить для великих мереж з обмеженими ресурсами, де обробка даних відбувається на сервері, а клієнт лише відображає результати.
- Товстий клієнт - тип клієнта, який має більше локальної обробки даних. Вся основна логіка програми або навіть зберігання частини даних можуть здійснюватися на локальній машині. Товсті клієнти зазвичай мають більше ресурсів і складніші функції, ніж тонкі клієнти, і вони використовують сервер для зберігання даних або обробки лише певних операцій. Це дозволяє зменшити навантаження на сервер і поліпшити швидкість роботи користувача, оскільки багато процесів виконуються локально.

У системах онлайн моніторингу криптовалют, часто застосовуються тонкі клієнти, оскільки більшість обчислень та зберігання даних відбувається на сервері, а клієнт лише візуалізує результати для користувача (наприклад, веб-браузер або мобільний додаток). Це дозволяє забезпечити масштабованість

системи і знижує вимоги до локальних ресурсів користувачів. Однак у деяких випадках можуть використовуватися і товсті клієнти, які обробляють частину даних локально, щоб зменшити затримки та знизити навантаження на сервер.

У власні системі я буду використовувати товстий клієнт, так як біржа лише надає поточно актуальні дані та у мене немає доступу до обчислювальних ресурсів серверу, тож всі обчислення будуть проводитися на локальній машині. І результат обчислення, якщо це потрібно, буде передаватися на сервер для взаємодії.

Також товстий клієнт є пріоритетним вибором, якщо я хочу масштабувати свою систему онлайн моніторингу та інтегрувати її в системи автоматизованої торгівлі чи інші подібні системи, оскільки він дозволяє здійснювати складну обробку даних без необхідності постійно звертатися до сервера. Це дає можливість створювати більш гнучкі та адаптивні рішення, де локальна обробка інформації значно зменшує затримки і підвищує ефективність роботи системи. Товстий клієнт також забезпечує більший контроль над процесами, оскільки більшість обчислень та аналітики виконуються на боці користувача, що може бути критично важливо при інтеграції з високопродуктивними торговими платформами.

Мікросервісна архітектура — це стиль проектування програмного забезпечення, при якому система розбивається на набір малих, автономних служб (сервісів), які взаємодіють між собою через стандартизовані інтерфейси, зазвичай через API. Такий підхід дозволяє кожному сервісу виконувати свою специфічну задачу і бути незалежним від інших, що значно спрощує масштабування та модифікацію системи.

Мікросервісна архітектура з’явилася як відповідь на обмеження монолітних додатків, які використовувалися в класичних IT-системах. Розвиток технологій у 2000-х роках, зокрема потужні сервери та зростаючі вимоги до швидкості та надійності програмних систем, спонукав компанії шукати рішення для гнучкого, масштабованого розвитку свої системних продуктів.

Більшість сучасних фінансових бірж має мікросервісну архітектуру. Однією з головних переваг даної архітектури є її здатність легко масштабувати систему. Для бірж, які обробляють величезну кількість транзакцій і даних у реальному часі, це важливий фактор.

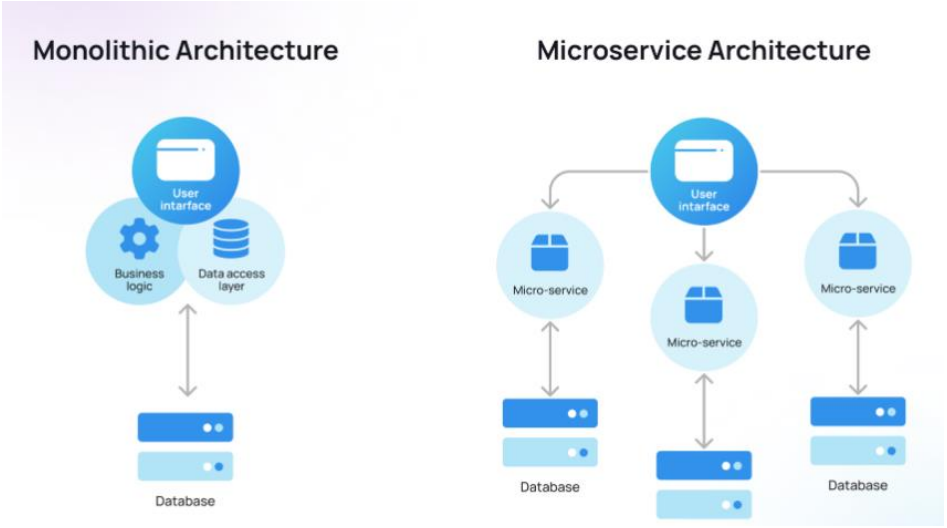


Рис. 2.1 Різниця між монолітною і мікросервісною архітектурою

Кожен мікросервіс можна масштабувати окремо, незалежно один від одного при потребі. Біржі часто потребують великих змін та оновлень без того, щоб порушити всю систему. Ця архітектура дозволяють окремо оновлювати чи замінювати компоненти без зупинки всієї системи, що дозволяє біржам постійно впроваджувати нові функції та підтримувати їх без значних перерв у роботі.

Для бірж критично важливо забезпечити мінімальні затримки та високу доступність. За допомогою мікросервісів можна розподіляти навантаження між різними сервісами, таким чином знижуючи ймовірність того, що одна помилка зупинить всю платформу.

Біржі зазвичай мають потребу в інтеграції з іншими численними системами, а мікросервісна архітектура дозволяють це робити швидко і ефективно, оскільки кожен сервіс може мати свій чітко визначений інтерфейс.

Кожен мікросервіс є окремим додатком, тож для різних частин системи можна вибирати різні технології реалізації. Це особливо корисно для бірж, де частини системи можуть мати різні вимоги до продуктивності або часу обробки.

Отже, мікросервісна архітектура надає біржам гнучкість, швидкість, масштабованість і надійність, що робить її ідеальним вибором для високонавантажених, постійно змінюваних торгових платформ.

У клієнт-серверній архітектурі важливу роль відіграють технології, що дозволяють взаємодіяти клієнтам та серверам між собою. Залежно від типу системи і вимог до швидкості, обсягу даних для передачі та рівня взаємодії, можуть застосовуватись різні технології та протоколи передачі даних.

HTTP (Hypertext Transfer Protocol) є стандартним протоколом для передачі даних через інтернет з’єднання. Використовується для взаємодії між веб-браузерами (клієнтами) і веб-серверами.

HTTPS (Hypertext Transfer Protocol Secure) є безпечним варіантом HTTP, використовує SSL/TLS для шифрування даних, що забезпечує безпечну передачу інформації між клієнтом і сервером.

Ці протоколи є основою для більшості веб-додатків і взаємодії через браузер, забезпечуючи запити (GET, POST, PUT, DELETE) і відповіді.

Для їх застосування використовують REST API (Representational State Transfer Application Programming Interface) — це архітектурний стиль і набір принципів для створення веб-сервісів. REST API дозволяє клієнтським додаткам взаємодіяти з сервером через стандартизований інтерфейс, що описує методи та структуру даних, як клієнт може спілкуватися з сервером. REST API дуже популярна в сучасних веб-застосунках завдяки своїй простоті та масштабованості.

Також є інші протоколи та інтерфейси для передачі даних.

GraphQL є альтернативою REST API і дозволяє клієнтам запитувати лише ті дані, які їм потрібні, що робить її більш гнучкою і економною у використанні ресурсів. У GraphQL можна виконувати запити для отримання даних з декількох джерел, об'єднуючи їх в один запит. Ця технологія була розроблена компанією Facebook у 2012 році. Офіційно проект був відкритий для спільноти як open-source у 2015 році. GraphQL широко застосовується для побудови сучасних веб-застосунків і мобільних додатків, де необхідна динамічна робота з даними.

В додатках, які потребують постійного з'єднання для швидкої передачі даних та мінімальної затримки активно використовують WebSocket. Це це протокол зв'язку, який забезпечує двосторонній, постійний канал між клієнтом і сервером через єдине TCP-з'єднання і дані можуть одночасно передаватися в обох напрямках між двома пристроями. Він дозволяє обмінюватися даними в реальному часі без необхідності постійно відправляти нові HTTP-запити. WebSocket був стандартизований у 2011 році як частина специфікації HTML5 і є широко застосовуваним для створення інтерактивних додатків, які вимагають обміну даними в реальному часі.

Веб-сокети активно застосовуються в системах моніторингу в реальному часі, так як для подібних систем критично важливим є постійна передача даних з мінімальною затримкою і яка при цьому не буде перенавантажувати ні клієнта ні сервера своїми запитами на з’єднання.

2.2 Технологія API

API (Application Programming Interface) — це набір визначень, протоколів і інструментів, що дозволяють одному програмному компоненту взаємодіяти з іншим. API виступає програмним інтерфейсом між різними системами або їх компонентами, надаючи стандартизований спосіб обміну даними.

Перші концепції API з'явилися ще у 1960-х роках, коли програмісти створювали бібліотеки функцій для обміну інформацією між різними частинами програм. У 1990-х роках із популяризацією інтернету та розвитком веб-застосунків і хмарних сервісів, цей протокол став невід'ємною частиною веб-технологій.

API працює за допомогою запитів і відповідей між клієнтом і сервером. Найпоширеніші типи запитів це GET, POST, PUT та DELETE.

У контексті системи моніторингу цін криптовалюти, різні типи запитів API виконують важливі функції, що забезпечують доступ до актуальних даних і взаємодію із сервером.

- Запит GET використовується для отримання інформації з серверу. Це можуть бути поточні ціни криптовалюти, історичні дані про коливання вартості, об’єму закупівель, баланс користувача, якщо була проведена аутентифікація, а також історія транзакцій, ордерів та інша інформація що доступна на біржі. Відповідь на такі запити зазвичай приходять в форматі JSON або XML в залежності від програмного інтерфейсу.
- Запит POST потрібен для створення нових ресурсів або передачі даних. У моніторинговій системі POST може використовуватися для надсилання користувацьких параметрів, внесення налаштувань на біржу, наприклад створення нових ордерів чи відкриття позицій, проведення транзакцій.
- Запит PUT використовується для зміни параметрів вже створених об’єктів. Це можу бути внесення змін до відкритих ордерів чи інших параметрів, які доступні користувачу для редагування. А також для редагування налаштувань API-ключів для отримання доступу.
- Запит DELETE призначений для видалення ресурсів з серверу(в даному випадку). Використовується для видалення сповіщень або налаштувань, які більше не потрібні, скасуванню підписки на торгові пари та інші задачі.

Різні типи запитів забезпечують повноцінну функціональність системи моніторингу цін криптовалюти. Завдяки цим механізмам система може ефективно працювати, задовольняючи потреби користувачів.

Дані по API зазвичай передаються в стандартизованому форматі, найчастіше це JSON (JavaScript Object Notation) - текстовий формат для зберігання і передачі структурованих даних між різними системами. JSON став стандартом для роботи з API завдяки простоті, легкості читання і сумісності з багатьма мовами програмування. Біржа Binance також пропонує у своєму програмному інтерфейсі саме цей формат даних. Формат JSON підтримується більшістю сучасних мов програмування, таких як Python, JavaScript, Java, C#, а також доступні бібліотеки

для роботи з даним форматом, що дозволяє розробнику працювати з ними на вищому рівні абстракції. Також він зручний для розуміння людиною, що полегшує налагодження і аналіз даних. Дані в JSON представлені у вигляді тексту, що дозволяє легко передавати їх через мережу.

Технологія API необхідна для інтеграції різних програм і систем, дозволяючи їм ефективно взаємодіяти між собою. Вона забезпечує автоматизацію, надаючи програмний доступ до функцій. Крім того, API сприяє масштабованості, дозволяючи створювати розподілені, модульні системи, які можна легко розширювати та вдосконалювати. Для створення мікросервісної архітектури. Використання готових API також значно економить час розробників, оскільки дає доступ до існуючих функцій і сервісів без необхідності їх створення з нуля.

Для забезпечення ефективної взаємодії між різними програмними системами та платформами, API дотримуються загальноприйнятих стандартів передачі даних. Найпоширенішим із них є REST (Representational State Transfer), який завдяки простоті та ефективності широко використовується для роботи через HTTP-протокол. SOAP (Simple Object Access Protocol) забезпечує більш формалізований і безпечний підхід, що підходить для корпоративних додатків. GraphQL, у свою чергу, надає гнучкість, дозволяючи клієнтам отримувати лише ті дані, які їм потрібні, зменшуючи обсяг непотрібної інформації в запитах. Для задач, які вимагають високої швидкості та масштабованості, використовується gRPC — сучасний бінарний протокол, що оптимізує передачу даних. Усі ці стандарти допомагають створювати універсальні, ефективні та масштабовані системи, що відповідають потребам сучасного програмного забезпечення.

Загалом використання API дає багато перевага, але існують і виклики. Одним із ключових аспектів є безпека: API повинні бути захищені від несанкціонованого доступу, що потребує належної авторизації та шифрування. Крім того, обмеження продуктивності, такі як квоти на кількість запитів, можуть впливати на ефективність роботи системи. Ще одним критичним фактором є якість документації: API повинні мати чітко структуровану та детальну документацію, щоб розробники могли легко інтегрувати її у свої проекти.

Отже, API є ключовою технологією у клієнт-серверній архітектурі та відіграє вирішальну роль у сучасних програмних системах. Вони дозволяють створювати розподілені додатки, інтегрувати зовнішні сервіси й автоматизувати взаємодію між різними програмними компонентами.

2.3 Технологія веб-сокетів

WebSocket — це сучасний мережевий протокол, який забезпечує двостороннє спілкування між клієнтом і сервером через єдине довготривале підключення. На відміну від традиційного HTTP-протоколу, де кожен запит клієнта вимагає відповіді від сервера, WebSocket дозволяє надсилати дані в обидва напрямки без необхідності повторного встановлення з'єднання. WebSockets активно використовуються там, де потрібна висока швидкість обміну даними в реальному часі.

Застосовується в фінансових системах, системах онлайн моніторингу, в онлайн чатах, де необхідна швидка реакція на повідомлення.

Використання веб-сокетів має низку переваг, якщо порівнювати з запитами API. Веб-сокети характеризуются своєю високою ефективнісю. Після встановлення з'єднання, передача даних проходить з мінімальними накладними витратами, оскільки немає необхідності повторного запиту. Це ідеально підходить для додатків, які потребують миттєвого оновлення даних. Також завдяки постійному з'єднанню зменшується використання серверних і мережевих ресурсів. Окрім того, є можливість передавати дані в обох напрямках між клієнтом і сервером у будь-який момент, не чекаючи на відповідь, що додає системам, які використовують цю технології гнучкості у реалізації свої функцій.

Але протокол WebSocket також має і свої недоліки. Так наприклад, він вимагає правильного управління з'єднаннями, особливо в масштабованих системах. Цей протокол є достатньо новим тому можуть викликати труднощі в застарілих системах. Також постійне з’єднання це не завжди добре, Постійне з'єднання може споживати більше ресурсів у системах із низькою кількістю

одночасних клієнтів. А також може бути вразливим до атак, наприклад, DDoS або втрати даних.

Перед тим як встановити двосторонній зв’язок між клієнтом та сервером, веб-сокет повинен встановити TCP-з'єднання за допомогою хендшейку (handshake) - це процес встановлення початкового з'єднання між клієнтом і сервером. Він проходить на програмному рівні через HTTP-протокол, а після завершення переходить до повнодуплексного з'єднання, тобто такого типу передачі даних, за якого інформація може одночасно передаватися в обох напрямках між двома пристроями. Щоб це зробити клієнт надсилає серверу спеціальний HTTP-запит зі статусом GET і заголовками, які сигналізують про бажання перейти на протокол WebSocket, після того як сервер опрацює цей запит, він згенерує і відправить клієнту відповідь, що готовий перейти на новий тип з’єднання. Коли підтвердження апгрейду відбудеться то зв’язок переходить у двосторонній режим. І далі передача даних відбувається за новими правилами WebSocket-протоколу.

Таблиця 2.1

Порівняння WebSockets та API

Параметр	WebSockets	API(HTTP/HTTPS)
З’єднання	Постійне (з’єднання відкривається раз і працює далі).	Кожен запит створює нове з’єднання.
Швидкість	Швидше, оскільки не створюється нове з’єднання.	Вища затримка через багаторазове створення HTTP-запитів.
Напрямок зв’язку	Двосторонній: сервер може ініціювати передачу даних клієнту.	Односторонній: сервер відповідає лише на запити клієнта.
Типи передачі	Підходить для систем, що працюють в режимі реального часу, дані передаються без затримок.	Підходить для періодичної передачі даних (REST API)
Об’єм даних	Підтримує потоки даних без повторення HTTP-заголовків.	Включає HTTP-заголовки у кожному запиті, що збільшує розмір передачі.

Хендшейк - це ключовий етап, який робить цей протокол одночасно гнучким і ефективним для застосунків.

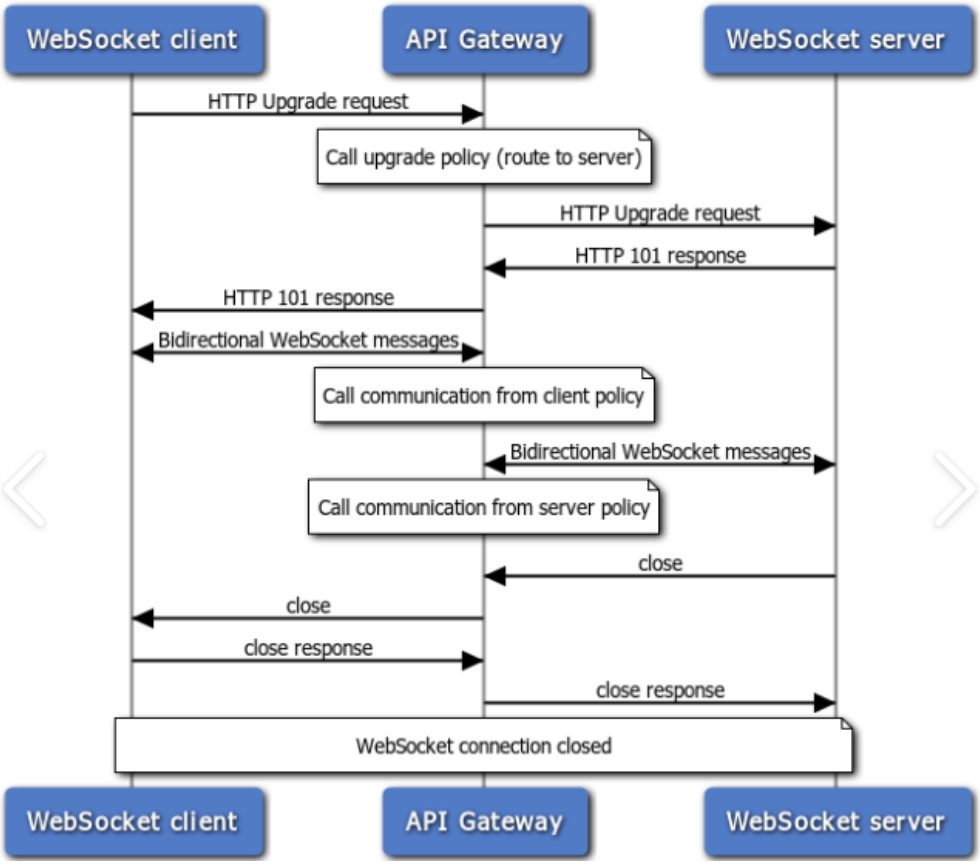


Рис. 2.2 Схема трьох-етапного хендшейку

Отже, WebSocket — це критично важлива технологія для сучасних додатків, які працюють у реальному часі. Її ефективність і швидкість роблять її незамінною в фінансових додатках, та системах моніторингу в онлайн режимі.

2.4 Технології для створення графічного інтерфейсу

Графічний інтерфейс користувача (GUI) відіграє важливу роль у системах моніторингу, зокрема на біржах криптовалют, завдяки своєму здатності наочно та інтуїтивно показувати дані в реальному часі. Це дозволяє користувачам швидко отримувати та аналізувати інформацію, що необхідна для прийняття рішень щодо торгівлі або інвестицій.

Кожна сучасна інформаційна моніторингова платформа має графічний інтерфейс, тому розробці даного компоненту потрібно приділити достатньо уваги.

User-friendly інтерфейс забезпечує зручність і легкість використання, дозволяючи користувачам швидко орієнтуватися в системі, що зменшує час навчання та підвищує ефективність виконання завдань. Це сприяє позитивному досвіду користувача, знижує ймовірність помилок та підвищує продуктивність роботи, особливо в складних системах, таких як біржові платформи чи системи онлайн моніторингу.

Вимоги до сучасних графічних інтерфейсів включають адаптивність до різних пристроїв, інтуїтивно зрозумілий дизайн, високу швидкість роботи. Добре створений інтерфейс сприяє якості взаємодії, підвищує залученість користувачів і допомагає бізнес проектам досягати своїх цілей.

Для створення інтерфейсу в веб-застосунках здебільшого використовують поєднання розміток HTML, стилів CSS та мови програмування JavaScript.

HTML (Hyper Text Markup Language) є основою для створення структури веб-сторінки. HTML відповідає за формування основних елементів інтерфейсу: таблиць, панелей, кнопок, графіків, текстових блоків, форм для введення даних та виведення. Він задає логічний порядок і взаємодію компонентів, забезпечуючи їх доступність для подальшого стилізування.

CSS (Cascading Style Sheets) доповнює HTML, відповідаючи за візуальне оформлення цих елементів. За допомогою CSS визначаються кольори, шрифти, розташування елементів, їхні розміри та анімації, що створює естетичний і зручний інтерфейс для користувача.

За допомогою мови JavaScript можна надати інтерфейсу динамічності та можливості взаємодії в режимі реального часу. У системах моніторингу JS дозволяє оновлювати таблиці даних і графіки без перезавантаження сторінки, обробляти події користувача (наприклад, кліки, введення даних, переміщення курсора і так далі) та динамічно змінювати вигляд елементів. JS інтегрується з API для отримання та виводу актуальних даних у реальному часі, що критично важливо для систем моніторингу цін криптовалют.

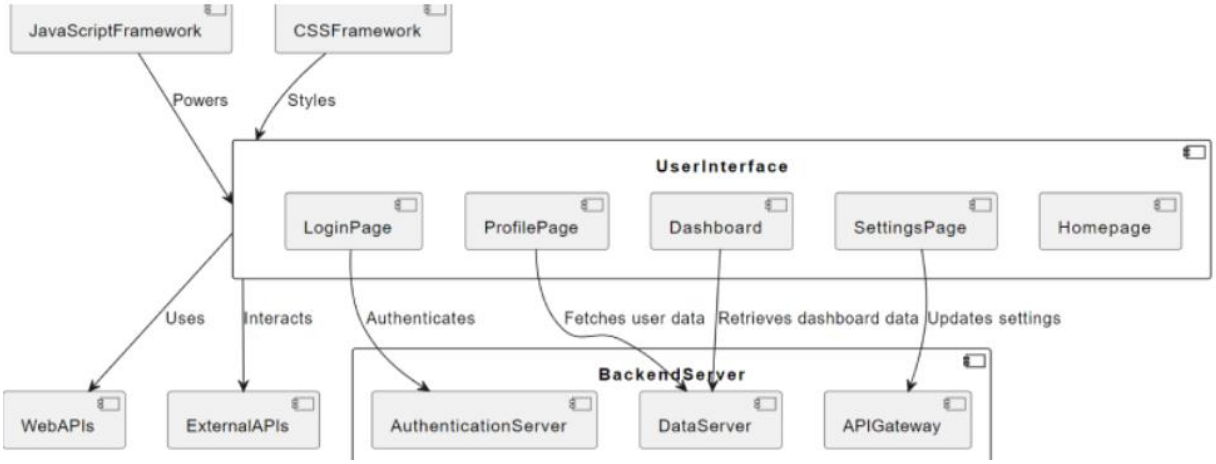


Рис. 2.3 Схема взаємодії з використанням GUI

На рис. 2.3 зображено схему взаємодії між графічним інтерфейсом користувача та серверною частиною системи. Вона демонструє, як різні сторінки інтерфейсу (Dashboard або SettingsPage) звертаються до бекенд-сервера для отримання або оновлення даних. Особливу роль відіграють JavaScript та CSS фреймворки, які забезпечують функціональність та візуальну складову інтерфейсу, а також API для інтеграції з зовнішніми сервісами.

Сучасні інтерфейси повинні залишатися зручними на різних пристроях, тому CSS використовується для створення адаптивного дизайну за допомогою медіа-запитів. Це дозволяє оптимізувати відображення графіків, таблиць та інших елементів залежно від розміру екрану. Наприклад, на мобільних пристроях складні таблиці можуть замінюватися компактними списками, а великі графіки - спрощеними.

Для спрощення роботи з HTML, CSS і JS у розробці інтерфейсів часто застосовують сучасні фреймворки та бібліотеки. Такі інструменти забезпечують готові стилі та компоненти, що дозволяють швидко створювати зручний і стильний інтерфейс. JavaScript-фреймворки, як-от React.js, допомагають будувати складні динамічні системи з можливістю повторно використовувати компоненти, що спрощує масштабування системи.

Таким чином, використання цих інструментів у створенні інтерфейсів забезпечує структурованість, стильність і динамічність, що є критично важливим для систем моніторингу й аналітики в режимі реального часу.

Окрім веб-застосунків, часто використовуються десктопні застосунки для ПК. Такі програми забезпечують більш високу продуктивність і автономність, ніж веб-застосунки, що робить їх зручними для роботи з великими обсягами даних у системах моніторингу. Інтерфейс таких програм часто створюється з використанням спеціалізованих фреймворків і бібліотек, наприклад, PyQt для Python чи JavaFX для Java. Ці інструменти надають широкі можливості для інтеграції складних графіків, інтерактивних таблиць та індикаторів.

Однією з головних переваг цих застосунків є використання системних ресурсів комп’ютера. Це дозволяє виконувати обчислення, візуалізацію даних і обробку складних графіків локально, без залежності від швидкості інтернет-з'єднання чи завантаженості серверу. У системах моніторингу це особливо важливо для користувачів, які працюють з фінансовими інструментами, такими як криптовалютні біржі, де необхідне миттєве оновлення даних.

Для побудови графічного інтерфейсу використовуються інструменти, які забезпечують високу гнучкість і кастомізацію (процес адаптації та налаштування для цільового користувача). PyQt і Qt надають можливість створювати складні вікна з інтерактивними кнопками, графіками та іншими елементами системи необхідними для ефективного моніторингу цін криптовалют. Саме тому я буду використовувати цей набір інструментів для створення власного програмного забеспечення.

Отже, десктопні застосунки є важливим інструментом у системах моніторингу завдяки своїй автономності, гнучкості та ефективності. А розробці графічних інтерфейсів потрібно приділити достатньо уваги, щоб програмний продукт був конкурентоспроможним.

2.5 Технології баз даних

Бази даних є основою для зберігання, організації та обробки даних у багатьох системах, і фінансових в тому числі. Вони забезпечують централізоване або децентралізоване в залежності від архітектури, зберігання великих обсягів інформації, забезпечують доступність і надійність цих даних для подальшої обробки. У системах моніторингу біржових цін криптовалют, бази даних необхідні для зберігання інформації про курси валют, історії транзакцій, ордерів та іншої важливої інформації, яка потребує швидкого доступу.

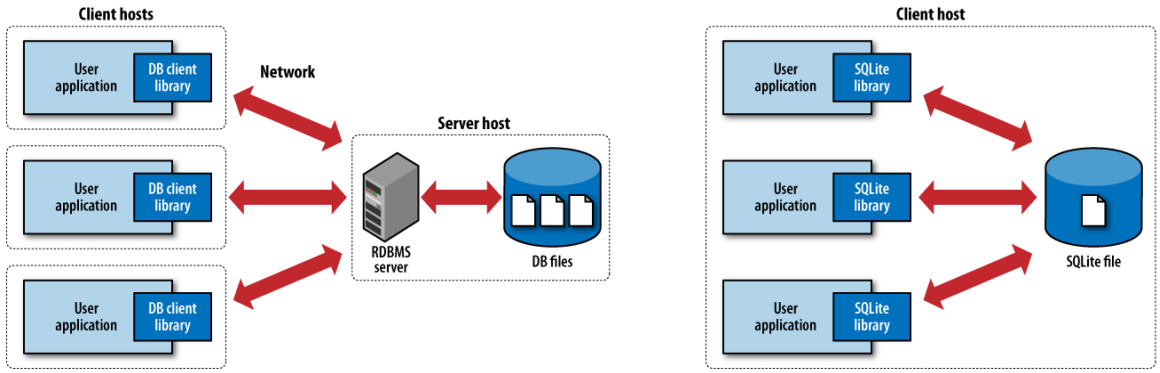
Існує кілька типів баз даних, які використовуються в залежності від потреб конкретної системи. Найпоширенішими є реляційні бази даних, такі як MySQL, PostgreSQL та SQLite, які використовують структуру таблиць для зберігання та організації даних. Крім того, для неструктурованих або великих даних використовуються нереляційні бази даних, такі як MongoDB, Cassandra та Redis. Вони дають більшу гнучкість і масштабованість для зберігання даних, які часто змінюються або мають складну структуру.

Дані можуть зберігатися в різних середовищах залежно від того, де вони потрібні. Для великих організацій і онлайн-платформ зазвичай використовуються хмарні сервіси. Ці сервіси надають масштабовані рішення для зберігання даних, а також для їх обробки та аналітики. Для менших або локальних додатків дані можуть зберігатися на фізичних серверах або в локальних базах даних.

Доступ до даних на біржі, в даному випадку Binance, забезпечується через API. Цей інтерфейс дозволяє користувачам отримувати інформацію про поточні ціни, історію транзакцій, ордери, а також здійснювати операції. Дані на біржі зазвичай зберігаються в розподілених базах даних для забезпечення швидкого доступу та високої доступності. Веб-сервіси API надають структурований доступ до даних, дозволяючи стороннім додаткам проводити інтеграція з функціями біржі.

Для систем моніторингу біржових цін криптовалют, доступ до актуальних даних є критично важливим. Для ефективного моніторингу необхідно оперативно отримувати і обробляти великі обсяги інформації. Важливо, щоб система була здатна обробляти запити в режимі реального часу, без затримок.

У реляційних базах даних дані зберігаються у структурованому вигляді, що дозволяє ефективно з ними взаємодія за допомогою SQL-запитів.



(a) Traditional client-server architecture (b) SQLite serverless architecture

Рис. 2.5 Різниця між серверною архітектурою і локальною для бази даних

Локальні бази даних, такі як SQLite, мають певні переваги в порівнянні з хмарними рішеннями. Вони не потребують інтернет-з'єднання для доступу до даних, що робить їх корисними для мобільних або десктопних (настільних) додатків. Локальні бази даних можуть бути швидшими в певних сценаріях, коли потрібно зберігати та обробляти дані без необхідності в зовнішніх запитах до серверів. Вони також зручні для використання в розробці і тестуванні програм, де підключення до великої бази даних може бути зайвим.

Для реалізації власної системи добре підходить локальна база даних SQLite.

Отже, технології баз даних є критично важливими для будь-якої системи моніторингу, оскільки вони забезпечують зберігання, доступ і обробку даних у реальному часі. Вибір типу бази даних залежить від вимог системи, обсягу даних та швидкості доступу до них.

SQLite є вбудованою реляційною базою даних, що використовує файл для зберігання всіх даних, замість сервера. Його архітектура побудована так, щоб

забезпечити ефективність і легкість використання в межах локальних або невеликих додатків, де не потрібен потужний сервер бази даних.

Так як вся база даних зберігається в одному файлі то це дозволяє легко переносити та створювати резервні копії. Кожен новий запис у базі даних записується в цей файл, що робить доступ до даних швидким і зручним для малих та середніх за обсягом додатків. SQLite не має традиційного клієнт-серверного підходу, тому відсутня необхідність в окремому сервері бази даних або складній інфраструктурі для обробки запитів. Додаток, що працює з SQLite, безпосередньо відкриває файл бази даних і виконує SQL (Structured Query Language) запити.

У базі даних дані зберігаються в бінарному форматі на диску, що забезпечує високу швидкість зчитування та запису. SQLite використовує механізм журналу транзакцій, який дозволяє здійснювати атомарні операції і гарантує цілісність даних. При цьому база даних залишається в консистентному стані навіть після неочікуваних збоїв або помилок. Також підтримуються індекси для швидкого доступу і є механізми для оптимізації запитів, щоб забезпечити швидку роботу навіть при великих обсягах даних.

Для роботи з SQLite в Python використовується бібліотека sqlite3, яка дозволяє виконувати SQL-запити, управляти базою даних і отримувати результати. Вона дає можливість створювати таблиці, вставляти дані, оновлювати та видаляти записи, а також використовувати транзакції для безпечної роботи з даними.

Запити до бази даних здійснюються за допомогою мови SQL (Structured Query Language). Для того щоб взаємодіяти з базою даних, спочатку потрібно встановити з'єднання між додатком та базою. Потім відправляються SQL-запити, які можуть виконувати різні операції.

- SELECT - отримання даних
- INSERT - вставка нових записів
- UPDATE - оновлення існуючих записів
- DELETE - видалення

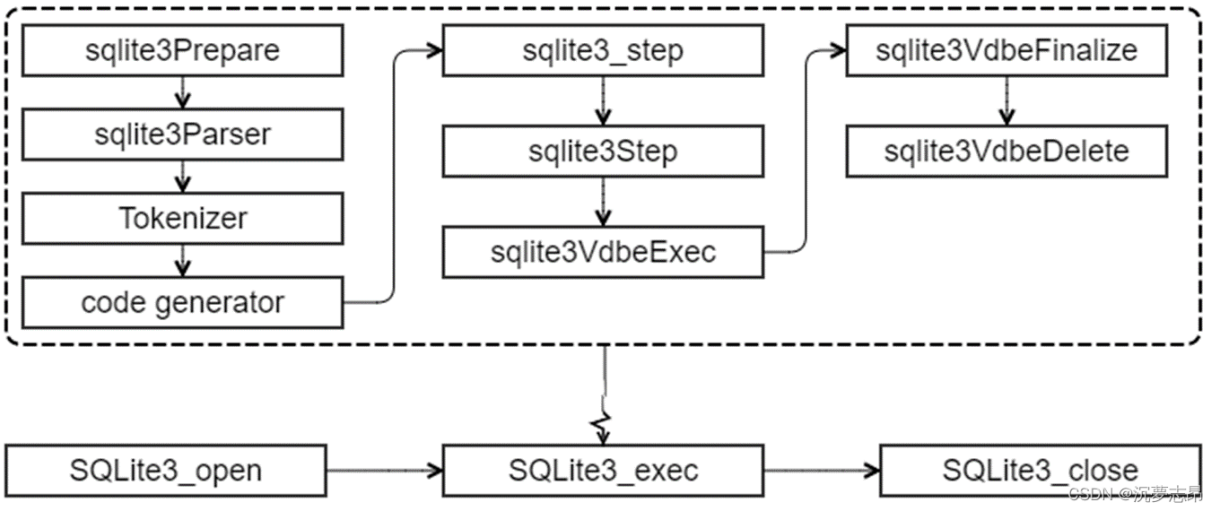


Рис. 2.6 Детальний опис етапів роботи SQLite

Ця схема на рис. 2.6 показує основні етапи роботи бази даних, починаючи з відкриття і закінчуючи її закриттям.

- SQLite3_open - це початковий етап, де створюється або відкривається файл бази даних. На цьому рівні встановлюється базове підключення між програмою і базою даних.
- SQLite3_exec - починається робота з даними, завантажуються команди на виконання
- sqlite3Prepare - SQL-запит, переданий у вигляді тексту, обробляється для створення готового до виконання байт-коду. На цьому етапі запит аналізується і оптимізується.
- sqlite3Parser і Tokenizer - запит поділяється на окремі частини (токени) для розуміння його синтаксису, щоб зрозуміти його структуру.
- code generator- на цьому етапі генерується низькорівневий байт-код, який готовий до виконання віртуальною машиною SQLite.
- sqlite3_step - запускається виконання запиту. Віртуальна машина бази даних обробляє байт-код і виконує операції, такі як вибірка, оновлення або видалення даних, в залежності від запиту.
- sqlite3VdbeExec -фактичне виконання байт-коду. На цьому етапі відбуваєтсья звернення до таблиць, зчитуються чи змінюються дані.

- sqlite3VdbeFinalize і sqlite3VdbeDelete -на фінальному етапі завершуються операції і звільняються ресурси, що використовувалися для виконання запиту.
- SQLite3 Close -завершальний етап роботи: база даних закривається, а всі відкриті з'єднання та процеси звільняються.

SQLite ідеально підходить для застосунків, де потрібен локальний, малий за розміром і швидкий доступ до даних без необхідності в централізованому сервері бази даних. А мова програмування Python надає, що дозволяє легко інтегрувати бази даних у будь-які програми, включаючи системи моніторингу криптовалют.

2.6 Вибір мови програмування

Правильний вибір мови програмування є ключовим рішенням під час розробки будь-якої інформаційної системи, зокрема в такій складній галузі, як фінансовий моніторинг. Від обраної мови залежить не лише продуктивність розробки, а й ефективність самої системи, її зручність для користувача, надійність і здатність адаптуватися до майбутніх змін. Мова програмування визначає доступ до необхідних бібліотек, інструментів для інтеграції з іншими системами, можливості роботи з різними платформами і швидкість виконання коду.

Для створення систем моніторингу криптовалют Python є ідеальним вибором. Насамперед, мова має величезну екосистему фреймворків і бібліотек для роботи з графікою (PyQt6, Matplotlib), фінансовими даними (Pandas, NumPy), графіками (Plotly, PyQtGraph) і мережевими протоколами (WebSockets, Requests). Це дозволяє швидко інтегрувати в систему функціональність для візуалізації даних, роботи з API бірж і моніторингу даних у реальному часі.

Більше того, платформа Binance офіційно надає SDK на Python, що забезпечує легкий доступ до її API, зокрема для отримання біржових котирувань, виконання транзакцій і роботи з історичними даними.

Таблиця 2.2

Порівняння популярних мов програмування

Критерій	Python	C++	Java	C#
Підтримка SDK Binance	Офіційна бібліотека Binance API Python	Доступно через сторонні бібліотеки	Доступно через сторонні бібліотеки	Доступно через сторонні бібліотеки
Фреймворки для GUI	PyQt, Kivy, Tkinter, PySide	Qt, WxWidgets, GTK	JavaFX, Swing	WPF, Windows Forms
Підтримка багатопоточності	Так (threading multiprocessing)	Так(Std::thread, OpenMP)	Так (Java.util.concurrent)	Так (Threading.Tasks)
Інтеграція з SQLite	Вбудована бібліотека Sqlite3	Через сторонні бібліотеки	JDBC	ADO.NET

Окрім того, Python має низку інших переваг. Це мова з простою і зрозумілою синтаксичною структурою, що робить її зручною навіть для новачків. Водночас вона є потужним інструментом для досвідчених розробників, адже підтримує принципи об’єктно-орієнтованого програмування (ООП). Це дозволяє створювати модульний код, який легше масштабувати, тестувати й підтримувати в майбутньому. Завдяки цьому підхід на основі об’єктів є особливо важливим у великих проектах, де потрібна чітка структуризація логіки додатка. Python підходить як для бекенд-розробки, так і для створення десктопних і веб-інтерфейсів, що забезпечує ефективну інтеграцію між різними компонентами системи. Завдяки цьому вся розробка може здійснюватися однією мовою, що спрощує координацію роботи і підтримку продукту.

Тому вибір Python як основної мови для проекту моніторингу цін криптовалют в режимі онлайн є виваженим і раціональним рішенням.

3 РОЗРОБКА СИСТЕМИ АВТОМАТИЗОВАНОГО ОНЛАЙН МОНІТОРИНГУ ЦІН КРИПТОВАЛЮТ

3.1 Постановка задачі

Система моніторингу криптовалютних бірж, зокрема Binance, має на меті задоволення потреби в локальних рішеннях для відслідковування і аналізу криптовалютних ринків без необхідності інтеграції в глобальні, ліцензовані інформаційні системи. Відповідно до цього, розробка орієнтована на програмне забеспечення, яке може бути використане для локального моніторингу та аналізу даних в режимі реального часу. Наявність такої системи дозволить зробити крок до створення масштабованих систем. Після інтеграції в більші системи, як приклад автоматизованої торгівлі, можливість інтеграції цієї системи у вже існуючі комплексні інфраструктури значно підвищить її ефективність та додасть функціональності і практичності моєму проекту.

Необхідно виконати інтеграцію з API біржі Binance. Оскільки моя система базується на отриманні даних з біржі, важливо, щоб вона могла працювати з API Binance. Це забезпечить постійний доступ до актуальних ринкових даних, що дозволить відображати ціну криптовалют, об’єми торгів та інші необхідні параметри в режимі реального часу.

Необхідно створити зручний інтерфейс користувача для відображення даних. Для ефективної роботи з фінансовими даними, система повинна мати інтуїтивно зрозумілий та простий графічний інтерфейс, що дозволить користувачам швидко отримувати та обробляти інформацію. Оскільки інформація на біржах змінюється швидко, важливо, щоб інтерфейс був максимально зручним та оптимізованим для швидкої реакції, а також підтримував побудову графіків для кращого аналізу ринку.

Необхідно забезпечити наявність функціоналу для побудови графіків і обробки фінансових даних. Графічні елементи, як-от діаграми і графіки, є основою для аналізу даних у фінансових системах. Вони дають можливість

користувачам бачити цінові тренди, обсяги торгів та інші фінансові показники. Для їх побудови потрібно реалізувати функціонал, який дозволяє відображати історичні дані та здійснювати оновлення графіків без затримок.

Потрібно налаштувати взаємодію з телеграм-ботом та можливість автоматичного оповіщення. Механізм оповіщення користувачів про зміну ціни криптовалют є важливим для своєчасного реагування на ринкові коливання. Інтеграція з телеграм-ботом дозволить оперативно сповіщати користувачів про досягнення заданих порогових значень ціни чи інші критичні зміни на ринку.

Для забезпечення ефективної роботи користувачів без затримок необхідно реалізувати багатопоточність у програмі. Це дасть можливість виконувати кілька завдань одночасно, наприклад, переглядати різні криптопари, слідкувати за кількома ринками або одночасно працювати з кількома графіками без впливу на загальну продуктивність системи.

Оновлення даних в режимі онлайн з прямим доступом до даних біржі. Оскільки ціни на криптовалюти змінюються дуже швидко, система повинна мати можливість постійно отримувати нові дані з біржі без необхідності вручну оновлювати вікно чи графік.

Необхідно забезпечити можливість використовувати локальну базу даних SQLite для збереження налаштувань. Зберігати налаштування користувачів, таблиці умов для спрацювання оповіщень, історії запитів і результатів обробки даних.

Десктопний застосунок повинен мати базові функціональні можливості для виконання свої профільних завдань.

Користувачі повинні мати можливість обирати, які саме криптовалюти вони хочуть відслідковувати. Це забезпечить персоналізацію системи під потреби конкретного користувача та дозволить ефективно відстежувати зміни в ціні обраної пари.

Користувач повинен мати можливість вибрати умови для спрацювання оповіщення та налаштування тексту самого повідомлення. Система має дозволяти користувачам встановлювати умови для оповіщень, наприклад, досягнення певної

ціни або зміна обсягу торгів. Також можна налаштовувати текст повідомлення, що дозволить отримати персоналізовані сповіщення за обраними умовами.

Потрібно забезпечити функцію побудови графіків з історичними даними криптовалют з оновленням в реальному часі. Система повинна підтримувати побудову графіків на основі історичних даних з біржі, а також постійно оновлювати їх відповідно до нових даних, що надходять. Це дозволить користувачам здійснювати глибший аналіз ринку і прогнозувати рухи цін на основі поточних трендів.

Додати збереження налаштувань до локальної БД. Всі налаштування користувача, включаючи вибір криптовалюти, умови оповіщень та історію переглядів, зберігаються в локальній базі даних. Це забезпечить їх збереження при перезапуску програми і дозволить користувачеві швидко відновити налаштування без необхідності їх повторного вводити в терміналі.

Користувач повинен мати можливість проводити менеджмент умов. Може редагувати, активувати, видаляти умови. Важливим функціоналом є також можливість активувати або вимикати певні умови, що дозволить більш гнучко використовувати наявний функціонал.

Система повинна мати модульну архітектуру та складатися з кількох основних модулів. Це інтерфейс користувача, модуль обробки даних, методи для інтеграція з API біржі, модуль бази даних та модуль для взаємодії з API телеграм бота. Всі ці модулі повинні працювати разом для забезпечення зручного та ефективного моніторингу цін криптовалют.

Архітектура системи передбачає багатопоточність, що дозволить виконувати кілька операцій одночасно без зниження продуктивності. А модульна архітектура дозволяє легко додавати нові функціональні можливості та покращення в систему без необхідності переписувати її цілком. Система буде сумісна з Windows 10 та не потребуватиме великих системних ресурсів для нормальної роботи.

3.2 Проектування системи

Для кожного модуля визначена модель даних, щоб вони могли взаємодіяти між собою. Запеспичена синхронізація при доступі до спільних даних.

Перед початком розробки системи необхідно врахувати вже існуючі принципи проектування, що допомагають забезпечити ефективність, масштабованість та зручність у майбутньому використанні й підтримці. Одним із ключових принципів є модульна архітектура.

Модульна архітектура передбачає поділ системи на незалежні компоненти (модулі), кожен з яких виконує конкретну функцію. Цей підхід дає змогу працювати над різними модулями паралельно, знижуючи складність системи загалом. У разі необхідності внесення змін або покращень вони можуть впливати лише на певний модуль, не чіпаючи інших частин.

Такий дизайн спрощує тестування та налагодження. Окремі модулі легше перевіряти на коректність, що підвищує якість кінцевого продукту. Також легко додавати нові функціональні модулі або розширювати існуючі, адаптуючи систему до нових вимог, якщо такі з'явилися.

Крім того, модулі можна використовувати повторно в інших проектах або інтегрувати у більші системи. Для сучасних систем, це звичайна практика.

Модульна архітектура — це основа для побудови гнучкої, надійної та довговічної системи, яка може відповідати потребам сучасного програмного забезпечення.

При розробці багатьох застосунків часто використовують модель MVC (Model-View-Controller) — це патерн архітектури для розробки масштабованих і модульних систем. Він передбачає розподіл відповідальносі між трьома основними компонентами: Model (Модель), View (Представлення) та Controller (Контролер). Такий підхід забезпечує чітку організацію коду, спрощує розширення та обслуговування системи.

Таким чином компоненти системи розподіляють по модулям за їх функціоналом.

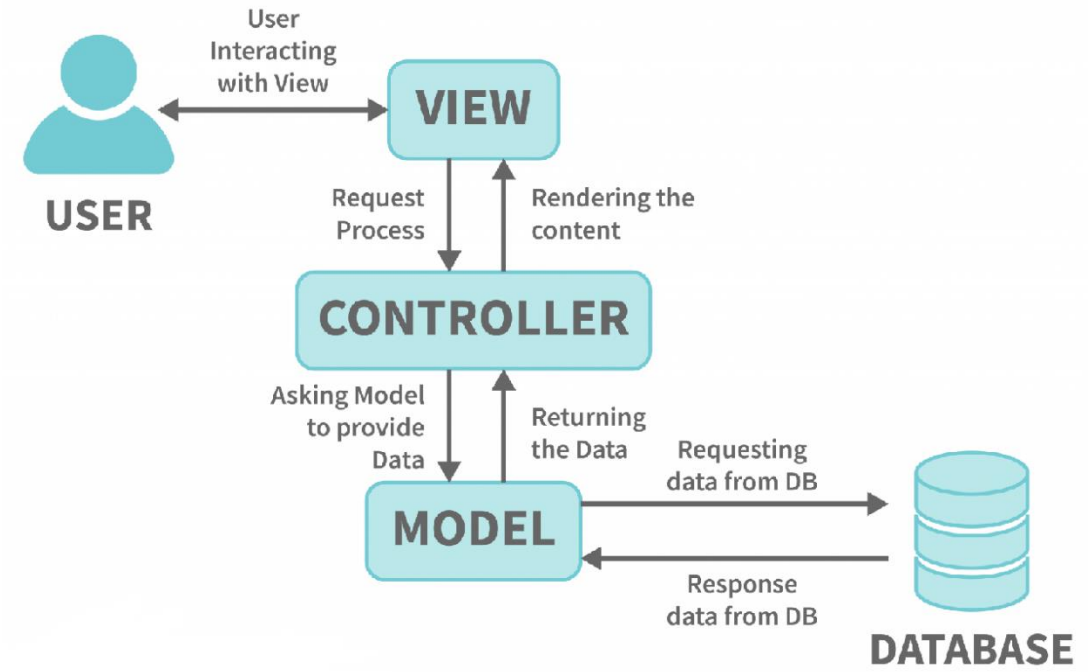


Рис. 3.1 Модель MVC

Суть компонентів (рис. 3.1) моделі MVC:

- Model є центральним компонентом, що відповідає за управління даними, логікою бізнес-процесів і взаємодію з базою даних. Вона зберігає та обробляє дані додатка.
- View відповідає за відображення інформації користувачу. Вона не має прямого доступу до логіки, а лише відображає дані, які передає їй модель через контролер.
- Controller виступає посередником між Model та View. Він обробляє запити від користувача, викликає відповідні методи моделі та передає дані представленню.

Активно використовується в моніторингових системах, так як має ряд перевагах, що забезпечують зручність розробки у складних системах. Також активно використовується мікросервісна архітектура, для забеспечення максимальної незалежності між компонентами системи і більшій масштабованості.

Для невеликих проектів добре підходить архітектура з менеджером завдань (Task Manager) і є досить популярною, особливо в системах, де потрібне багатозадачне виконання завдань, координація різних модулів і висока ефективність роботи.

Архітектура з менеджером завдань є ефективним підходом до організації роботи багатопотокових систем, що забезпечує координацію, планування й контроль виконання завдань у межах програми. Ця архітектура ідеально підходить для систем моніторингу, де потрібно виконувати багато різноманітних операцій паралельно, такі як, отримувати дані з API, оновлювати графічний інтерфейс та зберігати інформацію в базі даних.

Менеджер завдань виступає центральним елементом системи, який керує всіма потоками та ставить завдання на виконання. Вони спочатку реєструються у менеджері завдань, потім створюються окремі потоки для кожної операції. Менеджер забезпечує правильну послідовність виконання завдань і стежить за їх статусом, що особливо важливо для систем із високою інтенсивністю обміну даними.

У моніторингових системах, де необхідно виконувати операції паралельно, архітектура з менеджером завдань є найкращим вибором. Вона дозволяє гнучко розподіляти завдання між модулями, підтримувати високу продуктивність і забезпечувати безперебійну роботу системи в реальному часі.

При розробці власного програмного забеспечення, використовую головні ідеї даної архітектури, хоча є також частини і від MVC, так як модулі в системі поділяються за своєю функціональністю і мають чітко визначений інтерфейс для взаємодії між частинами компонентів.

Система складається з кількох важливих модулів, кожен з яких виконує специфічні функції, і разом вони забезпечують роботу всього застосунку.

- Модуль графічного інтерфейсу

Цей модуль забезпечує безпосередній зв'язок між користувачем та системою. Він складається з кількох компонентів, які відповідають за обробку виведення

таблиць, введення даних та побудову фінансових графіків. Таблиці використовуються для відображення актуальних даних, що постійно оновлюються, а графіки необхідні для візуалізації фінансової інформації, що дозволяє користувачу наглядно оцінювати тенденції на ринку. Компонент введення даних дозволяє користувачу налаштовувати умови моніторингу, а також обирати криптовалютні пари для відстеження. Всі ці компоненти сприяють забезпеченню зручним та інтуїтивно зрозумілим графічним інтерфейсом.

- Модуль обробки запитів користувача

Цей модуль забезпечує обробку запитів користувача в багатопоточному режимі. Завдяки цьому, система може одночасно обробляти кілька запитів без зупинки інших процесів. Це критично важливо для моніторингових систем, оскільки вони часто мають справу з великим обсягом запитів, які потребують швидкої обробки та повернення результатів без затримок. В цьому режимі зменшується навантаження на основний потік програми, що дозволяє підтримувати високу продуктивність системи в цілому і він не буде блокуватися, що може привести до зниженню інтерактивності графічного компоненту системи.

- Модуль для взаємодії з API Binance

Цей модуль відповідає за з'єднання з API біржі Binance. Він має функціональність для отримання даних про криптовалюти, а також їх передачу. Важливою особливістю цього модуля є постійне з'єднання через веб-сокет, що дозволяє отримувати дані в режимі реального часу. Завдяки цьому, система може оперативно реагувати на зміни цін на ринку, забезпечуючи точні та актуальні дані для подальшої обробки та відображення на графіках і таблицях. Модуль API взаємодіє з іншими компонентами системи, зокрема з модулем обробки даних та графічним інтерфейсом.

- Модуль взаємодії з телеграм-ботом

Цей модуль виконує певну інтеграцію з Telegram для налаштування автоматичних оповіщень. Він дозволяє відправляти повідомлення користувачам про зміни на ринку або про виконання визначених умов моніторингу. Взаємодія з

телеграм-ботом забезпечує зручний спосіб отримання інформації в реальному часі, навіть якщо користувач не відкрив додаток, а просто запустив в фоновому режимі. Це важливий аспект для будь-якої моніторингової системи, адже дає змогу оперативно реагувати на зміни без необхідності постійного перебування біля терміналу.

- Модуль взаємодії з базою даних

Модуль для роботи з базою даних займається збереженням та маніпулюванням даними, які стосуються налаштувань користувача, моніторингу криптовалют, а також історичних даних. Він дозволяє виконувати операції читання, редагування та видалення даних. Зберігання налаштувань у базі даних є важливим для того, щоб користувач міг зберігати свої налаштування при вимкненні системи. Цей модуль тісно взаємодіє з іншими, такими як модуль обробки даних і модуль графічного інтерфейсу, для відображення актуальних і точних результатів.

- Модуль обробки даних

Цей модуль відповідає за перевірку виконання умов моніторингу криптовалют. Він здійснює логіку, необхідну для прийняття рішень про активацію оповіщень чи зміну даних, що підлягають моніторингу. Модуль обробки даних аналізує інформацію, яку він отримує через API біржі, порівнює її з попередніми умовами користувача та виводить результати, що відповідають заданим критеріям. Завдяки цьому компоненту користувач має можливість налаштовувати умови, за яких система повідомлятиме про певні зміни.

- Модуль "Менеджер завдань"

Цей модуль є головним для забезпечення багатопоточності в системі. Він створює окремі потоки для кожного нового завдання, таким чином забезпечуючи паралельне виконання кількох процесів. Модуль менеджера завдань дозволяє значно знизити навантаження на основний потік програми, тим самим підвищуючи її продуктивність. Він є важливим елементом архітектури, що дозволяє ефективно організовувати паралельну обробку різних процесів.

3.3 Вибір фреймворків для розробки системи

У процесі розробки програмного забезпечення важливо правильно вибирати інструменти та фреймворки, які забезпечать ефективну реалізацію поставлених завдань. Вибір відповідних технологій значною мірою впливає на продуктивність системи, її масштабованість і зручність у подальшій підтримці. Для розробки цієї системи було вибрано кілька потужних фреймворків, серед яких важливу роль відіграє фреймворк для розробки графічних інтерфейсів PyQt6.

PyQt6 — це одна з найпопулярніших бібліотек для розробки графічних інтерфейсів на мові Python для операційних систем Windows, Linux, macOS. Вона є обгорткою для Qt. PyQt6 надає доступ до великого набору віджетів та інструментів, що дозволяють створювати як прості, так і складні графічні інтерфейси, включаючи підтримку мультимедіа, веб-оглядачів, а також інтеграцію з базами даних.

Однією з основних причин використання PyQt6 є його здатність забезпечувати високу продуктивність та ефективну роботу навіть з великими обсягами даних, що важливо для моніторингових та аналітичних систем, таких як ця. Важливою особливістю є також підтримка багатозадачності за допомогою потоків, що дозволяє створювати виконувати задачі в режимі багатопоточності без зниження продуктивності.

У поєднанні з іншими бібліотеками, такими як PyQtGraph, можна створювати потужні інструменти для візуалізації даних у вигляді інтерактивних графіків і діаграм, що є важливим для фінансових та аналітичних додатків.

Для того щоб використовувати фреймворк його потрібно імпортувати в простір імен.

PyQt6 дозволяє ефективно організовувати взаємодію між різними компонентами системи, таким як обробка запитів, зв'язок з API, базами даних та іншими частинами програмного забезпечення. Він забезпечує зручну реалізацію

подій, сигналів і слотів, що є основною моделлю для взаємодії між компонентами в реальному часі.

```
import sys
import telebot
from threading import Thread
import pyqtgraph as pg
import requests
import websocket
import json
import pandas as pd
from PyQt6 import QtCore, QtGui, QtWidgets
from time import time as now, sleep
from datetime import datetime, timezone
from lightweight_charts import Chart
from lightweight_charts.widgets import QtChart
from PyQt6.QtCore import QTimer
import sqlite3
```

Рис. 3.3 Приклад імпорту фреймворків

Якщо потрібен лише компонент, а не вся бібліотека то можна включити лише певну частину як показану на рис. 3.4.

```
from PyQt6.QtCore import QObject, QThread, pyqtSlot ,pyqtSignal
from PyQt6.QtWidgets import QGraphicsProxyWidget, QLabel
from PyQt6.QtWidgets import QVBoxLayout, QWidget
from PyQt6.QtWidgets import QApplication, QMainWindow
```

Рис. 3.4 Приклад імпорту конкретних модулів

Враховуючи потреби в реалізації функціоналу сповіщень та інтерактивних повідомлень для користувачів, в даній системі використовую бібліотеку telebot, яка надає простий інтерфейс для створення бота в популярному месенджері Telegram.

Telebot — це бібліотека Python, яка дозволяє легко створювати боти для Telegram, підтримуючи всі необхідні функції для інтерактивної взаємодії з користувачем. Вона забезпечує зручний доступ до API Telegram для обробки повідомлень, команд, передачі медіа-файлів та взаємодії з користувачами через чат. Бібліотека telebot є дуже популярною серед розробників завдяки простоті використання, широкому функціоналу та стабільності роботи.

Однією з ключових переваг використання telebot є його здатність працювати в асинхронному режимі, що дає змогу обробляти кілька запитів одночасно, не

блокуючи виконання інших операцій у системі. Це особливо корисно при необхідності обробки численних запитів від користувачів, без затримок і зависань системи. Завдяки асинхронним можливостям, бот може миттєво реагувати на команди користувачів, обробляти інформацію в реальному часі та забезпечувати актуальні сповіщення про зміни на ринку, нові угоди чи інші важливі події.

Повідомлення можуть бути надіслані у вигляді текстовій або графічній формі в чат.

Для візуалізації фінансових даних у системі важливо використовувати бібліотеки для побудови графіків, які забезпечують точність і швидкість відображення великих обсягів даних. Для цієї мети в системі були обрані дві популярні бібліотеки, а саме finplot та lightweight_charts.

Finplot — це бібліотека для Python, спеціалізована на побудові інтерактивних фінансових графіків. Вона була розроблена з урахуванням потреб фінансових аналітиків та трейдерів, тому має вбудовані функції для візуалізації різноманітних фінансових індикаторів, таких як свічки, лінії тренду, рівні підтримки та опору, а також інших показників, що використовуються в технічному аналізі.

Бібліотека дозволяє створювати графіки з мінімальними затримками, що критично важливо при роботі з реальними даними ринку. Вона також підтримує інтерактивність, що дає змогу користувачам взаємодіяти з графіками, наприклад, змінювати масштаби, додавати або прибирати індикатори.

Lightweight_charts — це бібліотека, схожа по функціоналу з попередньою, але має є спрощеною версією і зосереджена в першу чергу на простому відображенні фінансового графіку, при цьому її простота робить допомагає підібрати сумісні компоненти системи . Крім того, що вона оптимізована під невеликі проекти, Ця бібліотека надає інтуїтивно зрозумілі методи що легко інтегрувати в власну систему. Lightweight_charts має вбудовані можливості для налаштування кольорів, шрифтів, стилів графіків, що дає можливість кастомізувати графіки під конкретні потреби. Вона також добре інтегрується з різними веб-технологіями, що дозволяє безперешкодно використовувати її в системах, що мають веб-інтерфейси для відображення фінансової інформації.

Бібліотека json в Python використовується для роботи з форматом JSON (JavaScript Object Notation), який є популярним форматом для зберігання і передачі даних.

Ця бібліотека дозволяє легко перетворювати списки чи словники у формат JSON і навпаки. Вона є незамінною при роботі з API, коли потрібно обробляти дані, що надходять у форматі JSON. Так наприклад, біржа Binance відправляє дані у цьому форматі і потрібно з ними працювати, і при передачі даних до бази даних SQLite також потрібен цей формат.

Бібліотека requests є однією з найпопулярніших для виконання HTTP-запитів у Python. Вона дозволяє легко взаємодіяти з веб-сервісами, надсилати GET, POST, PUT, DELETE запити до серверів, а також обробляти відповіді (JSON, текст, файли). Requests значно спрощує роботу з API та відправку даних через HTTP-протокол.

Бібліотека binance.um_futures є частиною офіційного Python SDK для роботи з Binance API, а конкретно для роботи з ф'ючерсними контрактами на платформі. Використання UMFutures дозволяє здійснювати операції з ф'ючерсними контрактами на криптовалюти, такі як відкриття і закриття позицій, отримання даних про ринок, робота з ордерами і багато іншого. Ця бібліотека потрібна для взаємодії з біржою.

Модуль websocket надає можливість роботи з вебсокетами в Python. Цей протокол часто використовується для створення з’єднання між клієнтом та Binance і оновленням даних в режимі реального часу.

Бібліотека pandas є однією з найпопулярніших бібліотек для обробки та аналізу даних у Python. Вона надає зручні структури даних, такі як DataFrame та Series, які дозволяють ефективно працювати з великими обсягами даних, виконувати різноманітні операції з ними, маніпулювати інформацією і проводити аналіз.

Я використовую її для фільтрації, групування і агрегації, а також виконання різних складних обчислень і операцій з даними.

У поєднанні з lightweight_charts, фреймворк pandas дозволяє створювати графіки та візуалізувати дані.

Модуль sqlite3 створений для роботи базами даних SQLite в Python. Це вбудована реляційна база даних, яка може бути розміщена на локальній машині для зберігання постій даних, таких як налаштування користувача.

Завдяки своїй легкості і швидкості роботи з файлами, SQLite є гарним вибором для невеликих програм.

Модуль threading у Python дозволяє працювати з багатозадачністю, забезпечуючи виконання кількох потоків в одному процесі. Thread — це клас, який дозволяє створювати та запускати нові потоки в програмі.

Використання потоків дає змогу здійснювати паралельну обробку задач, що є корисним для реалізації моніторингової системи з можливістю працювати з багатьма криптовалютами одночасно. А також забезпечує безперебійну роботу графічного інтерфейсу, API запитів, прослуховування веб-сокетів та обробку даних без необхідності блокувати якесь із завдань для виконання іншого.

QThread — це клас, наданий бібліотекою PyQt (або Qt), який є більш специфічним для використання в додатках з графічним інтерфейсом. Він є частиною QtCore і забезпечує багатозадачність. QThread є більш гнучким і інтегрованим рішенням для роботи з потоками в графічних інтерфейсах, де потоки повинні взаємодіяти з елементами інтерфейсу, як-от кнопки, текстові поля та інше.

На відміну від стандартного Thread, QThread може безпосередньо взаємодіяти з елементами інтерфейсу без порушення основного потоку (який відповідає за оновлення GUI).

QThread активно використовує концепцію сигналів та слотів в Qt. Це дозволяє потокам передавати дані та події між собою або з основним потоком без необхідності використання м’ютексів (Mutex) або складних механізмів синхронізації.

QThread має вбудовані методи для коректної зупинки потоку та контролю його виконання.

При потребі, можна додавати нові фреймворки в програму, якщо це буде потрібно, при використанні модульної системи достатньо просто імпортувати потрібні файли до потрібного простору імен.

3.4 Дизайн графічного інтерфейсу

Проектування графічного інтерфейсу є важливим етапом створення системи, оскільки саме через нього користувач взаємодіє з функціоналом програми. У цьому проекті інтерфейс розроблений за допомого графічного редактора Qt Designer, а також функціоналу, що надає бібліотека PyQt6. Дизайн передбачає високу інформативність, інтуїтивність і можливість миттєвого доступу до основних функцій, таких як перегляд графіків, таблиць і налаштування сповіщень.

Однією з ключових вимог до інтерфейсу є забезпечення інформативності. Біржові термінали надають доступ до великої кількості даних, і при цьому гармонічно поєднуються елементи, щоб користувач міг швидко знайти необхідну інформацію. При розробці свого програмного забезпечення я користуюся тим самим принципом.

Qt Designer — інструмент, що дає змогу легко додавати кнопки, поля введення, графічні віджети та налаштовувати їхні властивості. Однією з головних переваг є можливість експорту створеного дизайну у вигляді файлу з розширенням .ui, який легко інтегрується з Python-кодом. Це значно прискорює процес розробки, знижуючи кількість ручної роботи.

Ще одним важливим аспектом дизайну є використання стилів CSS. Завдяки цій технології можна адаптивно змінювати вигляд елементів інтерфейсу, задаючи кольори, шрифти, розміри й поведінку елементів під час взаємодії з користувачем. Стилі CSS дозволяють створити професійний вигляд кнопок, таблиць і списків, а

також додати ефекти наведення або натискання. Це робить інтерфейс більш і привабливим для користувача.

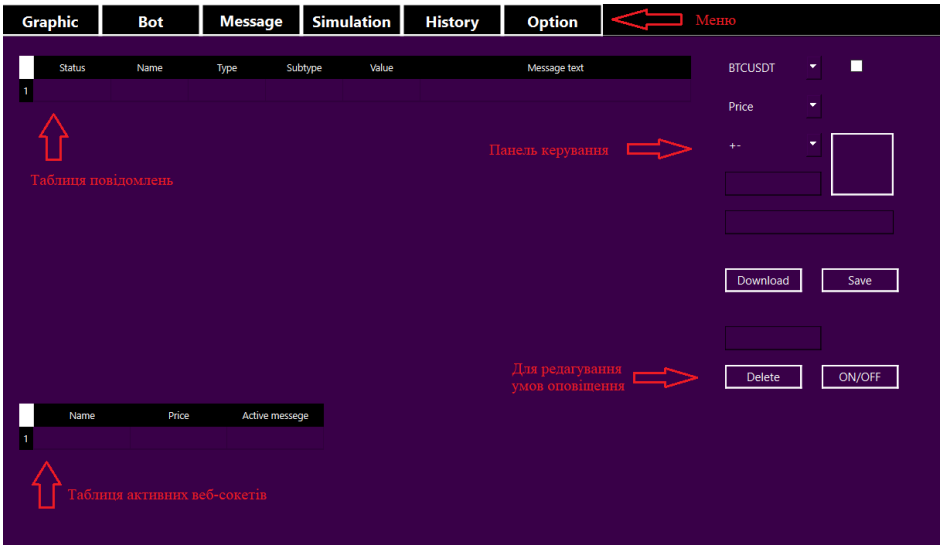


Рис. 3.5 Макет графічного інтерфейсу

Інтерактивність інтерфейсу досягається за рахунок програмування логіки взаємодії. Це включає оновлення таблиць, графіків та інших елементів у реальному часі.

Фінансовий графік є невід'ємною частиною сучасних моніторингових систем, тому я додаю можливість побудови такого в свою систему, для цього використовую методи lightweight_charts та історичні дані з біржі Binance.

Рис. 3.6 Вигляд фінансового графіку

Даний графік має шкалу ціни та часу. Пунктирна лінія показує поточну ціну. А також є можливість інтерактивної взаємодії. Користувач можу зсувати та зміщувати його за допомогою мишки.

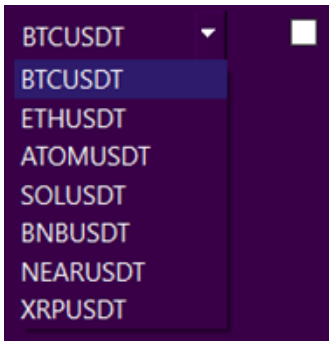


Рис. 3.7 Використання випадаючого списку

Використовую поля для вводу та для виводу, таблиці та чек-бокси. Користувач зможе вибрати криптовалюти зі списку як показано на рис. 3.7.

Або поставити галочку біля поля та ввести назву з клавіатури, що відкриє доступ до всіх наявних торгових пар на біржі.

Рис. 3.8 Ввід даних з клавіатури, якщо поставлена галочка в чек-бокс

Для забезпечення інтерактивності інтерфейсу використовуються слоти та сигнали, що надаються функціоналом PyQt6. Це дозволяє оперативно реагувати на події та виконувати потрібні функції. Така архітектура підвищує зручність і керованість розробки, дозволяючи легко додавати нові функції, змінювати логіку взаємодії т розширювати систему без значних змін до коду.

```python
@pyqtSlot()
def on_button1_clicked(self):
    print('Button ATOMUSDT (1)')
    # self.flag_signal_1.disconnect()
    if(self.IS_button_1_cliked):
        try:
            self.flag_signal_1.disconnect(self.Chart_update_candle_2_2)
            self.flag_signal_2.disconnect(self.Update_tiks_online)
        except:
            pass
        self.IS_button_1_cliked=False
        if(self.IS_web_socket_Use):
            self.client_web_socket.close_web_socket()
            self.IS_web_socket_Use=False
    else:
        self.IS_button_1_cliked=True
    self.Thread_RUN_UCC=True

    self.flag_signal_1.connect(self.Chart_update_candle_2_2)
    self.make_chart_update=Get_candlestick_from_Binance(self.flag_signal_1,
                                                        self.Thread_RUN_UCC,
                                                        'ATOMUSDT', '15m', 200)

    self.make_chart_update.start()
```

Рис. 3.9 Приклад використання слота

Для використання слота потрібно поставити перед методом модифікатор @pyqtSlot(), як це показано на рис. 3.9 та під'єднати кнопку чи інший інтерактивний об'єкт до нього, визначити тип взаємодії за допомогою спеціального методу.

```python
self.pushButton.clicked.connect(self.on_button1_clicked)
self.pushButton_2.clicked.connect(self.on_button2_clicked)
```

Рис. 3.10 Приклад встановлення з'єднання між кнопкою та слотом

Окрім такого варіанту, є можливість викликати слот і з коду, для цього можна використати метод emit() та сигнал, який можна приєнати до методу, що буде виконуватися при активації сигналу

```python
flag_changed = pyqtSignal()

self.flag_changed.connect(self.on_button4_2_4_clicked)
self.flag_changed.emit() # оновлення статусу в таблиці
```

Рис. 3.11 Використання сигналу та його активація

Умови оповіщення будуть виводитися у таблиці. Тут буде міститися головна інформація необхідна для їх ідентифікації та управління. Буде показово їх

поточний статус, назву, тип, саму умову та текст, що в разі активації буде вислано до телеграм чату.

```python
@pyqtSlot()
def Show_messeges_from_pd_df_AM(self):
    print("EMIT self.flag_signal_14")
    global Global_pd_df_of_all_messeges

    self.tableWidget_9.setRowCount(Global_pd_df_of_all_messeges.shape[0])
    self.tableWidget_9.setColumnCount(Global_pd_df_of_all_messeges.shape[1])
    self.tableWidget_9.setHorizontalHeaderLabels(Global_pd_df_of_all_messeges.columns.to_list())

    for i in range(Global_pd_df_of_all_messeges.shape[0]):
        for j in range(Global_pd_df_of_all_messeges.shape[1]):
            item = QtWidgets.QTableWidgetItem(str(Global_pd_df_of_all_messeges.iloc[i, j]))
            self.tableWidget_9.setItem(i, j, item)

    if Global_pd_df_of_all_messeges.empty:
        self.tableWidget_9.setRowCount(1)
```

Рис. 3.12 Метод для заповнення таблиці даними

Використовую фреймворк pandas для роботи з табличними даними в зручному форматі та заповнюю комірки таблиці відповідними даними. tableWidget – це екземпляр табличного віджету, що надає бібліотека PyQt6.

Теж саме про ціну криптовалюти. Власне її назва та поточна ціна, яка буде оновлюватися в режимі онлайн та кількість активних умов, що прив’язані до даної пари.

Якщо користувачу потрібно переглянути графік з історичними даними то можна перейти на вкладку «Graphic» та побудувати його за актуальними даними з можливістю оновлення.

```python
@pyqtSlot()
def Update_tiks_online(self):
    # метод для візуалізації тіків на графіку
    global Data_dict_ticks
    series_ticks = pd.Series(Data_dict_ticks)
    self.chart.update_from_tick(series_ticks)
```

Рис. 3.14 Фрагмент, що відповідає за оновлення графіку в режимі онлайн

Для оновлення даних на графіку використовую метод update_from_tick, та дані що надходять від веб-сокету. Кожного разу, коли з’являються нові дані цей слот активується, тому користувач завжди має доступ до актуальних даних, отриманих від біржі Binance.

```python
@pyqtSlot()
def Chart_update_candle(self):
    layout = QVBoxLayout()
    widget = QWidget()
    widget.setLayout(layout)
    self.chart = QtChart(widget)
    global P_dataframe_chart
    P_dataframe_chart['date'] = pd.to_datetime(P_dataframe_chart['date'], unit='ms')
    self.chart.set(P_dataframe_chart)
    layout.addWidget(self.chart.get_webview())
    layout.setContentsMargins(0, 0, 0, 0)

    proxy_widget = QGraphicsProxyWidget()
    proxy_widget.setWidget(widget)

    scenes = QtWidgets.QGraphicsScene()
    scenes.addItem(proxy_widget)

    widget.resize(self.graphicsView.size())
    self.graphicsView.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarPolicy.ScrollBarAlwaysOff)
    self.graphicsView.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarPolicy.ScrollBarAlwaysOff)
    self.graphicsView.setScene(scenes)

    if(self.IS_web_socket_Use):
        pass
    else:
        self.flag_signal_2.connect(self.Update_tiks_online) # метод для візуалізації тіків
        client = BinanceWebsocket(self.flag_signal_2)
        client.start()
        self.client_web_socket=client
        self.IS_web_socket_Use=True
```

Рис. 3.13 Побудова фінансового графіку за історичними даними

Для графічного зображення історії цін криптовалюти використовую бібліотеку Lightweight_charts, а для взаємодії з PyQt потрібен QGraphicsProxyWidget. Спочатку створюю сам віджет та налаштовую його взаємодію з методами, що відповідають за побудову графіку. Потім створюю проксі віджет та проксі сцену. Визначаю її як для веб-перегляду та передаю туди сам графік. Таким чином можна отримати фінансовий графік, вбудований в десктопний застосунок.

3.6 Використання бази даних SQLite

Використання бази даних необхідне для даної системи. SQLite буде використовуватися для збереження налаштувань користувача, що дозволяє створити комфортне середовище роботи з програмою. Користувач зможе зберегти

обрані умови моніторингу та налаштування оповіщень, щоб після перезапуску програми завантажити їх знову, не переналаштовуючи все заново.

Однією з важливих функцій бази даних є можливість зберігати історичні дані про ціну криптовалют. Це необхідно для проведення постаналізу та побудови графіків, які можуть допомогти користувачу оцінити тренди та приймати обґрунтовані рішення. Крім того, це дозволяє працювати із даними навіть без доступу до інтернету, використовуючи збережену історію.

```
def run(self):
    # Підключення до бази даних
    conn = sqlite3.connect('Test_sqlite3.db')
    cursor = conn.cursor()
```

Рис. 3.15 Підключення до бази даних

Спочатку потрібно підключитися, вказавши при цьому назву бази даних та поставити курсор.

```
    create_table_query = f"""
    CREATE TABLE IF NOT EXISTS {table_name} (
        Status TEXT,
        Name TEXT,
        Type TEXT,
        Subtype TEXT,
        Value REAL,
        Message_text TEXT
    )
    """
    cursor.execute(create_table_query)
```

Рис. 3.16 Запит мовою SQL на створення таблиці

Також база даних може зберігати історію використання або певних операцій, якщо це буде необхідно. Крім того вже готові методи взаємодії з базою даних будуть корисними для майбутньої інтеграції з іншими системами, такими як автоматизована торгівля.

```
    # Перевірка, чи є записи в таблиці
    check_query = f"SELECT * FROM {table_name}"
    cursor.execute(check_query)
    records = cursor.fetchall()
```

Рис. 3.17 Запит мовою SQL на отримання даних

Після того як з’єднання встановлене можна працювати з даними і командами управління. Якщо такої таблиці, яка містить налаштування ще немає, або вона була видалена то буде створена нова з визначеними полями та типами.

Якщо таблиця вже існує то цей запит візьме всі доступні дані і запише їх в змінну records.

```
if records:
    # Якщо записи є, створюємо DataFrame
    # columns = ["Status","Name", "Type", "Subtype", "Value", "Message text" ]

    Global_pd_df_of_all_messeges = pd.DataFrame(records, columns=Global_pd_df_of_all_messeges.columns)
```

Рис. 3.18 Запис даних до Data Frame

Якщо дані існують то їх записуємо в спеціальну таблицю, що надає фреймворк pandas. Це зручний формат даних для зберігання передавання та роботи з даними, що добре оптимізований для подібних задач та має вбудовані методи взаємодії.

```
# Збереження змін
conn.commit()
# Закриття з'єднання з базою даних
conn.close()

self.flag_changed.emit()
```

Рис. 3.19 Закриття з’єднання з SqLite

Для коректної роботи необхідно правильно зберегти зміни та закрити базу даних. Після цього імітується сигнал, який показує системі те, що робота з БД завершена і можна переходити до наступного пункту.

Отже, використання SQLite у цій системі дозволяє забезпечити зручність і функціональність роботи з даними, залишаючись простим і ефективним рішенням для локального збереження інформації.

3.5 Багатопоточність як основа для подібних систем

Багатопоточність є однією з ключових вимог до систем, які працюють у режимі реального часу та обробляють значні обсяги даних. Це дозволяє ефективно розподіляти ресурси системи та забезпечує паралельну обробку завдань, що суттєво підвищує продуктивність програми.

Для реалізації багатопоточності в Python я використовую два основні підходи: стандартні потоки (Thread) та потоки на основі QThread, які надаються фреймворком PyQt6. Кожен із цих підходів має свої переваги й підходить для різних задач у межах системи.

```
class Messeges_manager(QThread):
    def __init__(self, index_messege_in_df, flag_signal_14,flag_signal_15,comand_to_do):
        QThread.__init__(self)
        self.flag_changed = flag_signal_14
        self.flag_signal_15=flag_signal_15
        self.comand_to_do=comand_to_do
        self.index_messege_in_df=index_messege_in_df

    def run(self):
        # перевірка статусу повідомленя
        # головна логіка управління повідомленнями
```

Рис. 3.20 Приклад використання QThread

QThread дозволяє легко передавати дані між потоками та основним графічним інтерфейсом через сигнали. Я використовую наслідування для того щоб додати для даного класу (рис. 3.20), потрібний функціонал. Ініціалізація за допомогою команди QThread.__init__(self). Головний код менеджера повідомлень буде прописаний в run(self) – саме цей метод і буде запущений в окремому потоці.

```
self.Start_messege_meneger=Messeges_manager(parsed_value,self.flag_signal_14,self.flag_signal_15, comand_to_do)
self.Start_messege_meneger.start()
```

Рис. 3.21 Ініціалізація менеджера повідомлень та запуск

Для запуску потоку потрібно виконати команду .start() – це запустить окремий потік і головний зможе продовжити свою роботу паралельно.

```
self.wsa = websocket.WebSocketApp(self.url_2, on_message=self.on_message,
                                  on_error=self.on_error,on_close=self.on_close)
```

```
def start(self):
    self.thread_io = Thread(target=self.wsa.run_forever)
    self.thread_io.daemon = True
    self.thread_io.start()
```

Рис. 3.22 Використання Thread бібліотеки threading

Коли робота з потоками не пов'язана з графічним інтерфейсом і не вимагає складної інтеграції з сигналами та слотами можна використовувати звичайний Thread, наприклад для запуску веб-сокету в окремому потоці, цей потік буде взаємодіяти з менеджером веб-сокетів і не буде отримувати доступ до графічного інтерфейсу напряму тому інтеграція слотів і сигналів тут не потрібно і є просто зайвим функціоналом, який може перенавантажувати систему, якщо їх буде відкрито занадто велика кількість

Загалом, обидва підходи можуть комбінуватися для оптимізації роботи системи. Використання багатопоточності в системі забезпечує масштабованість і стабільність її роботи. Завдяки потокам користувач отримує можливість працювати з додатком без затримок і зависань, навіть коли обробляється значна кількість паралельних завдань. Що актуально для моніторингу даних і взаємодії в системі реального часу.

Для синхронізації доступу до спільних даних було використано як інтегровані засоби QMutex так і інші можливості.

3.6 Створення телеграм бота для сповіщення користувача

Створення телеграм-бота для сповіщення користувача є важливим елементом системи, що забезпечує своєчасну передачу інформації в зручному форматі. Така функціональність дозволяє автоматизувати інформування користувача про

важливі події або зміни, наприклад, досягнення заданого порогу ціни криптовалюти чи інші визначені умови.

Для реалізації телеграм-бота в цій системі використовується бібліотека telebot, яка є частиною пакету pyTelegramBotAPI. Ця бібліотека надає прості й ефективні методи для інтеграції з API Telegram, дозволяючи створювати та налаштовувати бота із мінімальними витратами часу.

```python
class TelegramBot(QThread):

    def __init__(self,message_text, apikey_bot, chat_id):
        QThread.__init__(self)
        # Ініціалізуємо бота
        self.bot = telebot.TeleBot(apikey_bot)
        self.chat_id=chat_id
        self.message_text=message_text


        @self.bot.message_handler(func=lambda message: True)
        def echo_all(message):
            print(message.chat.id)
            print(message)
            self.bot.reply_to(message, "chat_id: " + str(message.chat.id))

    def send_messege(self):

        alert_emoji = "\U000026A0"  # значок лампочки
        currency_pair = "ATOMUSDT"  # монета
        price = "7.97"  # ціна
        dollar_emoji = "\U0001F4B2"  # знак долара
        message_text = "This is an alert for price change!"  # текст повідомлення

        self.message_text = f"{alert_emoji}{currency_pair} \n{dollar_emoji}{price}\n{message_text}"
        self.bot.send_message(self.chat_id, self.message_text )

    def run(self):
        self.send_messege()
        self.bot.polling(none_stop=True)
```

Рис. 3.23 Клас, що відповідає за роботу Telegram-бота

Основною задачею бота є отримання сигналів від системи та передача їх користувачеві через Telegram.

Створення бота починається з його реєстрація у месенджері. Для цього використовується BotFather, який видає токен доступу до API.

Після того як налаштували API ключ та визначити його ID чату, ці дані передаються до конструктору класу, де проводиться ініціалізація. Так як клас наслідує від QThread то даний код виконується в окремому потоці і не заважає основній частині системи займатися своїми задачами. В методі run(self) виконується безкінечне прослуховування чату- якщо користувач введе якийсь

текст то його можна буде опрацювати. Для цього є метод з спеціальним декоратором @self.bot.message_handler().

Оповіщення про виконання умови надходить користувачу в цей чат, повідомлення містить назву валюти, його поточну ціну та текстову частину, цей текст визначає сам користувач.

Telegram є популярною платформою для обміну повідомленнями, яка має широку підтримку на різних пристроях. Це робить інтеграцію з Telegram логічним і ефективним вибором для подібних систем.

Отже , додавання телеграм-бота до системи розширює її функціональність і значно підвищує зручність користування.

3.7 Проблеми сумісності

Проблеми сумісності можуть бути серйозною проблемою під час розробки програмного забезпечення на Python. Незважаючи на широкий вибір бібліотек і фреймворків, кожна з них має свої обмеження, які можуть створювати конфлікти між компонентами системи. У даній системі такі труднощі виникли під час використання інструментів для побудови графічного інтерфейсу та візуалізації даних.

Однією з основних проблем стала сумісність між QT Designer та версіями PyQt. QT Designer, інструмент для візуального проектування інтерфейсу, офіційно підтримує лише PyQt5. Однак для цього проекту була обрана PyQt6, оскільки вона пропонує сучасний функціонал, поліпшену інтеграцію з Python та підтримку нових стандартів.

Ще одна проблема сумісності виникла під час інтеграції бібліотеки Lightweight Charts для побудови графіків. Ця бібліотека частково не підтримує функціонал PyQt6, особливо в компонентах, які відповідають за інтерактивну роботу з графіками.

```
widget = QWidget()
widget.setLayout(layout)
self.chart = QtChart(widget)
proxy_widget = QGraphicsProxyWidget()
proxy_widget.setWidget(widget)
scenes = QtWidgets.QGraphicsScene()
scenes.addItem(proxy_widget)
self.graphicsView_5.setScene(scenes)
```

Рис. 3.24 Вирішення проблеми сумісності бібліотек

Вирішення проблем сумісності було реалізовано через використання декількох стратегій. Однією з них стало застосування одночасно двох версій PyQt — PyQt5 і PyQt6. Це дозволило зберегти функціонал QT Designer для побудови інтерфейсу та скористатися перевагами PyQt6 для роботи з сучасними компонентами. Для інтеграції графіку, який не підтримувався у PyQt6, було використано проксі-віджет (QGraphicsProxyWidget). Такий підхід дав змогу обійти обмеження, забезпечити сумісність компонентів і уникнути переписування значної частини коду.

Мова Python є кросплатформленною. Код може виконуватись на різних операційних системах, таких як Windows, macOS і Linux, практично без змін. Він запускається за допомогою інтерпретатора Python, який встановлюється на кожній цільовій платформі. Цей підхід усуває залежність від конкретної операційної системи, адже інтерпретатор обробляє код однаково на будь-якій платформі.

3.8 Розробка головних компонентів системи

Модуль взаємодії з API біржі Binance я реалізую за допомогою Python_SDK, що надає ця біржа для розробників.

Цей модуль забезпечує зв’язок між системою та біржею Binance. Використання Python SDK Binance дозволяє спростити процеси отримання та передачі даних завдяки готовим методам для роботи з API. Модуль відповідає за виконання запитів для отримання актуальних цін, історичних даних, а також для

підключення до веб-сокетів, які забезпечують постійний потік інформації в режимі реального часу.

```python
def get_current_price_USDT():
    try:
        ticker_p= um_futures_client.ticker_price(symbol=SYMBOL)
        price = float(ticker_p['price'])
        return price

    except Exception as err:
        print(err)
```

```python
def get_Balance_all():
    balances = um_futures_client.balance()
    print("Balances:")
    for asset in balances:
        print(f"asset: {asset['asset']},Balance: {asset['balance']}")
```

Рис. 3.25 Виконання запитів до біржі

В даному прикладі виконуються запити з використанням SDK. Це зменшує ризик виникнення помилок у порівнянні з ручним написанням HTTP-запитів, а також полегшує їхній контроль та обробку. Можна отримати доступ до аккаунту користувача та методів управління, якщо налаштувати доступ та знайти арі-ключі.

```python
class Messeges_Web_socket():
    def __init__(self,series,flag_signal_15,flag_signal_14):
        self.series = series.copy()
        self.flag_signal_15=flag_signal_15
        self.flag_signal_14=flag_signal_14

        self.Name_cripta = (self.series["Name"]).lower()  # Переведення в нижній регістр
        stream_name = f'{self.Name_cripta}@markPrice'
        self.url='wss://fstream.binance.com/ws/%s' % stream_name

        self.wsa = websocket.WebSocketApp(self.url, on_message=self.on_message,on_error=self.on_error,on_close=self.on_close)
```

Рис. 3.26 Веб-сокет з’єднання через публічний URL біржі Binance

А також можна використовувати публічний url, якщо авторизація не потрібна, або на даний момент неможлива.

Модуль додавання даних від користувача та запису в DataFrame Pandas відповідає за введення користувачем власних даних, які можуть бути використані для аналізу або налаштування умов. Дані, отримані з інтерфейсу користувача, обробляються й записуються в структуру DataFrame бібліотеки Pandas. Вибір

Pandas обумовлений її ефективністю при роботі з табличними даними та багатими можливостями для подальшої обробки, аналізу й візуалізації.

```
# Створюємо серії для кожного значення
data_series = pd.Series({
    "Status": status,
    "Name": selected_Name,
    "Type": selected_Type,
    "Subtype": selected_Subtype,
    "Value": selected_Value,
    "Message text": selected_Message_text
})

# Створюємо DataFrame і додаємо нову серію
global Global_pd_df_of_all_messeges

# Додаємо новий рядок в DataFrame
Global_pd_df_of_all_messeges.loc[len(Global_pd_df_of_all_messeges)]=data_series
print(Global_pd_df_of_all_messeges)

self.flag_signal_14.emit()
```

Рис. 3.27 Заповнення датафрейму інформацією від користувача

Спочатку беру з форм, розміщених в графічному інтерфейсі, інформацію по кожному пункту, створюю серію та записую туди ці дані, якщо вони пройшли перевірку на коректність, а потім додаю цю серії до таблиці у вигляді нового рядку.

Окремий модуль відповідає за створення потоку, в якому працює веб-сокет. Це забезпечує безперервне прослуховування даних у режимі реального часу без блокування основного потоку програми. Менеджер веб-сокетів ініціалізую за допомогою QThread бо він взаємодіє з GUI, а самі потоки веб-сокетів для кожної окремої криптовалюти вже через Thread, що дозволяє оптимально розподілити ресурси системи. Завдяки цьому модулю система може ефективно працювати з динамічними даними біржі.

Модуль перевірки виконання умови відповідає за аналіз даних у реальному часі та перевірку виконання активованих умов. Він отримує дані від веб-сокетів та користувацьких налаштувань і виконує необхідні обчислення для виявлення подій, які потребують реакції системи.

```
if self.List_of_active_messeges.loc[index_el, "Subtype"] == '>=':
    if data_dict['price']>= self.List_of_active_messeges.loc[index_el, "Value"]:
        # умова виконалася
        self.condition_complate(index_el)
        # якщо список повідомлень пустий то закрити веб-сокет та завершити потік
        if len(self.List_of_active_messeges)==0:
            self.close_web_socket()
        else:
            continue

elif self.List_of_active_messeges.loc[index_el, "Subtype"] == '<=':
    if data_dict['price']<= self.List_of_active_messeges.loc[index_el, "Value"]:
        # умова виконалася
        self.condition_complate(index_el)
        # якщо список повідомлень пустий то закрити веб-сокет та завершити потік
        if len(self.List_of_active_messeges)==0:
            self.close_web_socket()
        else:
            continue
```

Рис. 3.28 Перевірка виконання умов користувача

Коли система виявляє виконання заданих умов, цей модуль активується для виконання відповідного завдання.

```
def condition_complate(self, index_el):
    # відправити сигнал на виконання умови, видалити повідомлення :

    self.Start_condition_handler=Condition_handler(
        self.List_of_active_messeges.loc[index_el],self.cur_price)
    self.Start_condition_handler.start()
```

Рис. 3.29 Перевірка виконання умов користувача

При спрацюванні умови викликається обробник, який далі виконує дії, готує реакцію. В даному випадку це відправлення оповіщення до телеграм-бота з відповідним текстом повідомлення, але можу бути і щось інше, це можна використовувати як тригер до активації певного алгоритму.

Система складається з низки модулів, кожен із яких виконує окрему функцію, що сприяє її ефективності та гнучкості. Завдяки модульному підходу компоненти легко вдосконалювати або замінювати, адаптуючи систему до нових завдань.

3.9 Тестування системи

На етапі тестування систем перевіряється її готовність до реальної експлуатації. Тестування дозволяє оцінити, як система реагує на реальні умови, виявити слабкі місця та переконатися, що всі її функції інтегровані й працюють узгоджено.

Система була запущена в умовах, наближених до реальних. Перевірені всі ключові моменти, всі функції яким повинна відповідати програма

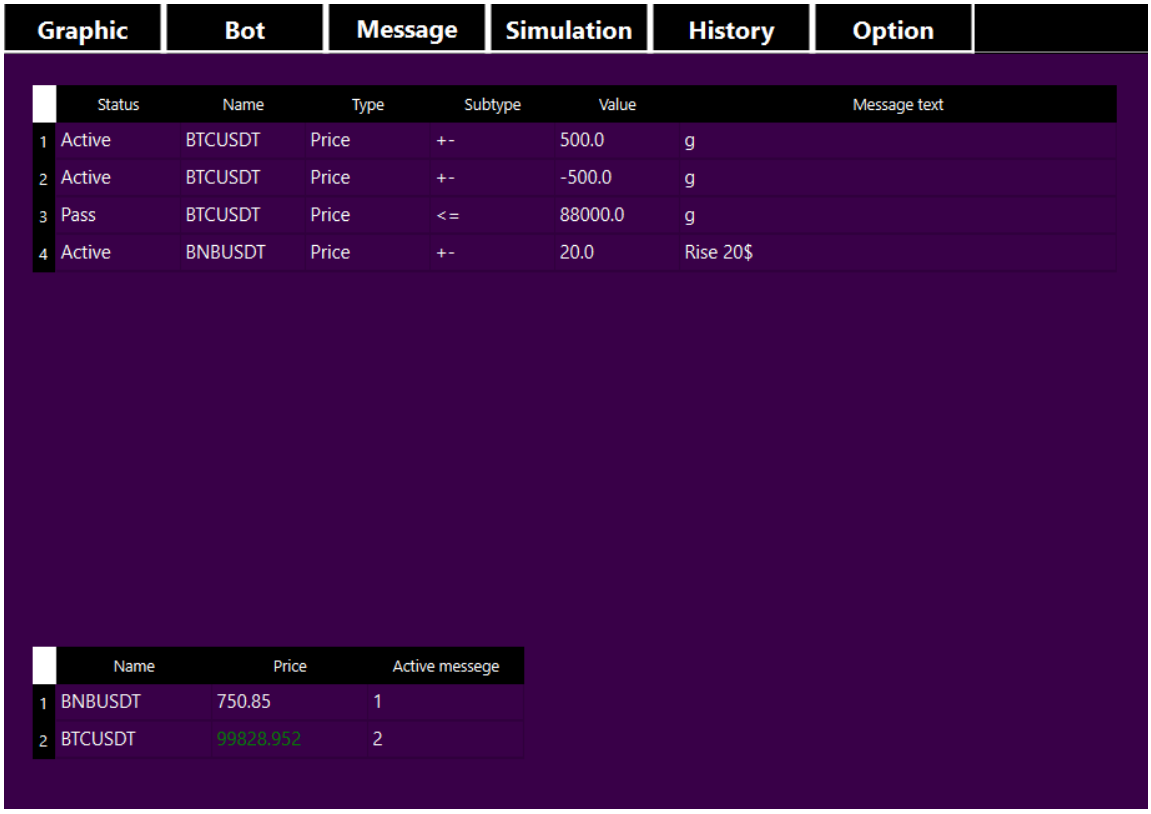


Рис. 3.30 Приклад використання програми

Таблиці були заповнені даними за допомогою спеціальної форми. Вони успішно додалися в таблицю. Також було вибрано декілька умов і активовано. Тож вони змінили свій статус на «Active» та почали перевірятися ці умови в режимі реального часу. Було створено два веб-сокет з’єднання для BNBUSDT і BTCUSDT.

У таблиці нижче відображається їх поточна ціна та кількість умов, що прив’язані до кожної криптовалюти. Якщо умова виконається, або буде зупинена чи видалена зі списку і умов більше не залишиться то потік завершить свою роботу.

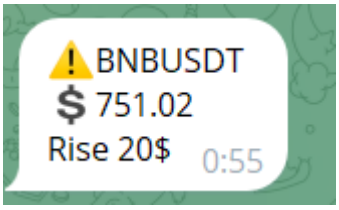


Рис. 3.31 Повідомлення, що прийшло до чату телеграм

Через деякий час можна очікувати оповіщення про виконання умови.

Якщо потрібно просто переглянути поточну ціну та історичні дані по можна зайти на вкладку «Graphic» та вибрати потрібну криптовалюту або ввести свою назву, якщо в випадаючому списку немає потрібної і система побудує графік. Та для оновлення налаштує відповідне з’єднання з біржою Binance.

Рис. 3.32 Фінансовий графік

Дані надходять кожні 3 секунди, можна налаштвувати і так, щоб була затримка лише в одну секунду. Інформація від біржі надходить у форматі Json, я

перетворюю дані в потрібний формат. Як видно на рис. 3.33 це часова мітка та поточна актуальна ціна.

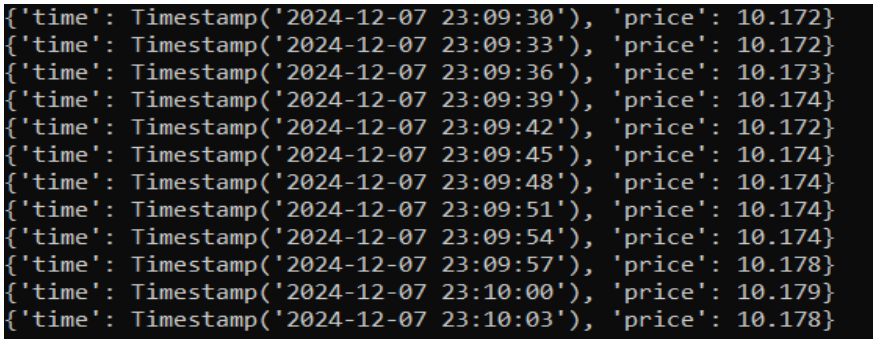


Рис. 3.33 Часова мітка та ціна оновлюється в режимі онлайн

Система продемонструвала свою здатність обробляти потоки даних у реальному часі, надавати користувачеві актуальну інформацію та вчасно реагувати на виконання заданих умов. Оповіщення через Telegram-бота було відправлено миттєво, підтвердивши ефективність інтеграції з зовнішніми сервісами.

Тестування показало, що система готова до використання. Вона демонструє високу функціональність і зручність для користувача, виконуючи всі поставлені завдання.

3.10 Можливості застосування та покращення системи

Система моніторингу біржових цін криптовалют, розроблена в межах цієї роботи, вже готова до використання за своїм прямим призначенням. Вона забезпечує автоматизоване отримання даних, моніторинг цін криптовалюти та сповіщення користувача.

Однією з ключових перспектив є інтеграція системи в автоматизовані системи торгівлі. Вона може стати частиною ширшої екосистеми, де виконуватиме роль аналітичного модуля для ухвалення торгових рішень.

Масштабування системи є ще однією перспективою. Вона може бути адаптована для роботи з іншими біржами, додатковими фінансовими

інструментами. Завдяки модульній архітектурі можна легко додати нові компоненти. Це зробить систему універсальною та ще більш корисною для користувачів.

Окремо слід зазначити можливість розміщення програмного коду на сервері. Такий підхід дозволить зробити систему доступною з будь-якого пристрою, забезпечити постійну роботу в цілодобовому режимі та знизить залежність від продуктивності локального комп’ютера. До того ж, серверне рішення спрощує масштабування, дозволяючи обробляти великі обсяги даних і підтримувати одночасно кількох користувачів.

Одним із напрямків для покращення є посилення системи безпеки. Використання сучасних стандартів шифрування даних та захисту з'єднань з API та Telegram-ботом гарантує конфіденційність і захист від потенційних загроз. Крім того, можна додати функції аутентифікації користувачів для захисту від несанкціонованого доступу.

Удосконалення графічного інтерфейсу також залишається актуальним завданням. Використання JavaScript та сучасних фреймворків, таких як React, може значно покращити дизайн і зробити інтерфейс більш привабливим.

Система демонструє високий рівень функціональності вже на поточному етапі, але має великий потенціал для подальшого розвитку.

ВИСНОВКИ

У рамках магістерської роботи було проведено всебічне дослідження, спрямоване на розробку автоматизованої системи моніторингу біржових цін криптовалют на основі мови Python. Дослідження підтвердило актуальність та доцільність створення таких систем, оскільки динамічний характер криптовалютного ринку вимагає від користувачів постійного моніторингу, аналізу даних та оперативного ухвалення рішень.

На основі аналізу сучасних технологій і фреймворків було розроблено програмне забезпечення з модульною архітектурою, яке дозволяє виконувати основні функції та задачі, що було поставлено перед системою. Було створено модулі для інтеграції з API біржі Binance, взаємодії з Telegram-ботом для сповіщення користувача, обробки запитів і виконання умов. Важливим компонентом є графічний інтерфейс, який забезпечує зручний доступ до даних, їх візуалізацію та введення нових параметрів.

Також були вирішені проблеми сумісності бібліотек. Завдяки використанню різних версій PyQt та застосуванню проксі-методів вдалося забезпечити стабільну роботу графічного інтерфейсу та інтеграцію сучасних бібліотек для побудови фінансових графіків. Це стало можливим завдяки застосуванню об'єктно-орієнтованого підходу.

Проведені дослідження підтвердили ефективність використання багатопоточності та асинхронного підходу для забезпечення стабільної роботи системи в реальному часі. Кожен ключовий процес, такий як, робота з веб-сокетами для отримання онлайн-даних, виконується у власному потоці. Це дозволило уникнути затримок і забезпечити максимальну швидкодію системи навіть за умови великого обсягу даних.

Тестування показало високу стабільність та функціональність системи. Було досягнуто повної відповідності поставленим у роботі завданням: оперативне отримання даних з біржі, перевірка виконання умов, сповіщення користувача та надання йому інструментів для аналізу даних.

Результати дослідження свідчать про широкі перспективи практичного використання розробленої системи. Вона може бути інтегрована в автоматизовані системи торгівлі, адаптована для роботи з іншими фінансовими інструментами або ринками. Система має високий потенціал для масштабування, що дозволяє її використовувати в ширшому контексті фінансового аналізу.

Подальший розвиток системи може включати впровадження серверного рішення для забезпечення її роботи в цілодобовому режимі, покращення безпеки даних, розширення функціональності Telegram-бота, а також удосконалення графічного інтерфейсу за допомогою сучасних фреймворків на основі JavaScript.

Таким чином, розроблена система відповідає сучасним вимогам та запитам користувачів, демонструючи високу гнучкість, надійність і адаптивність. Вона може слугувати основою для подальших досліджень і розробок у сфері автоматизації процесів аналізу та моніторингу фінансових ринків.

ПЕРЕЛІК ПОСИЛАНЬ

1. "PyQt6 Documentation" [Електронний ресурс]. – Режим доступу: https://www.riverbankcomputing.com/static/Docs/PyQt6/ (дата звернення 07.11.2024).
2. "Python Requests Documentation" [Електронний ресурс]. – Режим доступу: https://requests.readthedocs.io/en/master/ (дата звернення 07.11.2024).
3. "Lightcharts Documentation" [Електронний ресурс]. – Режим доступу: https://lightcharts.com/docs/ (дата звернення 07.11.2024).
4. "Pandas Documentation" [Електронний ресурс]. – Режим доступу: https://pandas.pydata.org/pandas-docs/stable/ (дата звернення 09.11.2024).
5. "WebSockets Documentation" [Електронний ресурс]. – Режим доступу: https://websockets.readthedocs.io/en/stable/ (дата звернення 09.11.2024).
6. "Binance API Documentation" [Електронний ресурс]. – Режим доступу: https://binance-docs.github.io/apidocs/spot/en/ (дата звернення 10.11.2024).
7. "Binance Futures API Documentation" [Електронний ресурс]. – Режим доступу: https://binance-docs.github.io/apidocs/futures/en/ (дата звернення 11.11.2024).
8. "Aiogram Documentation" [Електронний ресурс]. – Режим доступу: https://docs.aiogram.dev/en/latest/ (дата звернення 14.11.2024).
9. "SQLite Documentation" [Електронний ресурс]. – Режим доступу: https://www.sqlite.org/docs.html (дата звернення 18.11.2024).
10. "PyQtGraph Documentation" [Електронний ресурс]. – Режим доступу: http://www.pyqtgraph.org/documentation/ (дата звернення 18.11.2024).
11. "Matplotlib Documentation" [Електронний ресурс]. – Режим доступу: https://matplotlib.org/stable/contents.html (дата звернення 18.11.2024).
12. "yfinance Documentation" [Електронний ресурс]. – Режим доступу: https://pypi.org/project/yfinance/ (дата звернення 18.11.2024).
13. "Binance SDK" [Електронний ресурс]. – Режим доступу: https://github.com/binance/binance-connector-python (дата звернення 21.11.2024).

14. "Telegram Bot API Documentation" [Електронний ресурс]. – Режим доступу: https://core.telegram.org/bots/api (дата звернення 21.11.2024).

15. "Python for Data Science Handbook" [Книга] / Jake VanderPlas. – O'Reilly Media, 2019. – 549 с.

16. "Web Development with Python and Flask" [Книга] / Antonio Mele. – Packt Publishing, 2019. – 370 с.

17. "Requests: HTTP for Humans" [Електронний ресурс]. – Режим доступу: https://docs.python-requests.org/en/master/ (дата звернення 23.11.2024).

18. "WebSockets Documentation" [Електронний ресурс]. – Режим доступу: https://websockets.readthedocs.io/en/stable/ (дата звернення 25.11.2024).

19. "Python Documentation" [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/ (дата звернення 26.11.2024).

20. "Python Packaging User Guide" [Електронний ресурс]. – Режим доступу: https://packaging.python.org/ (дата звернення 27.11.2024).

21. "Real-time Data Monitoring with WebSockets" [Електронний ресурс]. – Режим доступу: https://www.toptal.com/python/real-time-data-monitoring-websockets (дата звернення 27.11.2024).

22. "CoinGecko API Documentation" [Електронний ресурс]. – Режим доступу: https://www.coingecko.com/en/api (дата звернення 29.11.2024).

23. "CryptoCompare API Documentation" [Електронний ресурс]. – Режим доступу: https://min-api.cryptocompare.com/documentation (дата звернення 30.11.2024).

24. "Cryptocurrency Data Analysis with Python" [Книга] / S. D. S. P. A. Bhattacharya. – Springer, 2021. – 312 с.

25. "Mastering Python for Data Science" [Книга] / Samir Madhavan. – Packt Publishing, 2020. – 400 с.

26. "Blockchain and Cryptocurrency Technologies" [Книга] / Arshdeep Bahga, Vijay Madisetti. – VPT, 2020. – 516 с.

27. WunderGraph. "5 Best Practices for Backend-for-Frontends" [Електронний ресурс]. – Режим доступу: https://wundergraph.com/blog/5-best-practices-for-backend-for-frontends (дата звернення 08.12.2024).

28. "Web Application Architecture" [Електронний ресурс]. – Режим доступу: https://peiko.space/blog/article/web-application-architecture (дата звернення 08.12.2024).

29. "Backend-for-Frontend (BFF) Architecture Guide" [Електронний ресурс]. – Режим доступу: https://codalien.com/blog/backend-for-frontend-bff-architecture-guide/ (дата звернення 08.12.2024).

30. "Front-End Architecture: In-Depth Analysis" [Електронний ресурс]. – Режим доступу: https://elitex.systems/blog/front-end-architecture-in-depth-analysis/ (дата звернення 08.12.2024).

ДОДАТОК А

```
class MyMainWindow(QtWidgets.QMainWindow, Ui_MainWindow,QObject):

    def __init__(self):
        super().__init__()
        self.setupUi(self)
        self.pushButton.setText("START")
        self.pushButton_2.setText("STOP")
        self.scene = QtWidgets.QGraphicsScene()
        self.web_view = QWebEngineView()

        # ____________________ GRAPHIC (tab_1) _________________________
        self.pushButton.clicked.connect(self.on_button1_clicked)
        self.pushButton_2.clicked.connect(self.on_button2_clicked)
        self.crypto_name="ATOMUSDT"
        self.timeframe="15m"
        items = ['BTCUSDT', 'ETHUSDT', 'ATOMUSDT', 'SOLUSDT',
                 'BNBUSDT', 'NEARUSDT', 'XRPUSDT']
        self.combo_box_1.addItems(items)
        items = ['4h','1h', '15m', '5m', '3m', '1m' ]
        self.combo_box_8.addItems(items)

        # ____________________  MESSEGE (tab_3) ____________________
        self.tabWidget.setCurrentWidget(self.tab_3)
        self.checkBox_3.stateChanged.connect(self.on_checkBox_3_stateChanged)
        items = ['BTCUSDT', 'ETHUSDT', 'ATOMUSDT', 'SOLUSDT',
                 'BNBUSDT', 'NEARUSDT', 'XRPUSDT']
        self.combo_box_5.addItems(items)
        items = ['Price', 'Position', 'Deposit']
        self.combo_box_6.addItems(items)
        items = ['+-','>=', '<=', '%']
        self.combo_box_7.addItems(items)
        self.combo_box_6.currentIndexChanged.connect(self.on_combo_box_6_changed)
        self.pushButton_22.clicked.connect(self.clicked_on_pushButton_22)
        self.pushButton_23.clicked.connect(self.clicked_on_pushButton_23)
        self.flag_signal_14.connect(self.Show_messeges_from_pd_df_AM)
        self.pushButton_24.clicked.connect(self.clicked_on_pushButton_24)
        self.pushButton_25.clicked.connect(self.clicked_on_pushButton_25)
        self.pushButton_26.clicked.connect(self.clicked_on_pushButton_26)
```

```
    @pyqtSlot(str, float, int)
    def Show_list_of_act_websockets_if_widget(self, Name, Price, Act_mes):
        self.mutex_2.lock()
        print("signal_15 is activated:")
        print(f"{Name}___{Price}___{Act_mes}")
        global list_sockets_of_messeges
        Index_of_web_sokets_in_table=-1
        self.tableWidget_10.setRowCount(len(list_sockets_of_messeges))
        index_el=0
        while index_el<len(list_sockets_of_messeges):
            if list_sockets_of_messeges[index_el].get_name()==Name:
                Index_of_web_sokets_in_table=index_el
                break
            index_el+=1

        self.tableWidget_10.setItem(Index_of_web_sokets_in_table, 0, QtWidgets.QTableWidgetItem(Name))
        self.tableWidget_10.setItem(Index_of_web_sokets_in_table, 2, QtWidgets.QTableWidgetItem(str(Act_mes)))
        item = self.tableWidget_10.item(Index_of_web_sokets_in_table, 1)
        if item is not None:
            try:
                current_value = float(item.text())
                new_price = float(Price)
                if new_price > current_value:
                    new_item = QtWidgets.QTableWidgetItem(str(new_price))
                    new_item.setForeground(QtGui.QBrush(QtGui.QColor("green")))
                    print("green")
                elif new_price < current_value:
                    new_item = QtWidgets.QTableWidgetItem(str(new_price))
                    new_item.setForeground(QtGui.QBrush(QtGui.QColor("red")))
                    print("red")
                else:
                    new_item = QtWidgets.QTableWidgetItem(str(new_price))
                    new_item.setForeground(QtGui.QBrush(QtGui.QColor("white")))
                    print("white")
                self.tableWidget_10.setItem(Index_of_web_sokets_in_table, 1, new_item)

            except ValueError:
                print("current_value not convert into float")
```

```
        else:
            print("==new_value_white==")
            new_item = QtWidgets.QTableWidgetItem(str(Price))
            new_item.setForeground(QtGui.QBrush(QtGui.QColor("white")))
            self.tableWidget_10.setItem(Index_of_web_sokets_in_table, 1, new_item)
        if len(list_sockets_of_messeges)==0:
            self.tableWidget_10.setRowCount(0)
            self.tableWidget_10.setRowCount(1)
        self.mutex_2.unlock()

    @pyqtSlot()
    def Show_messeges_from_pd_df_AM(self):
        print("EMIT self.flag_signal_14")
        global Global_pd_df_of_all_messeges

        self.tableWidget_9.setRowCount(Global_pd_df_of_all_messeges.shape[0])
        self.tableWidget_9.setColumnCount(Global_pd_df_of_all_messeges.shape[1])
        self.tableWidget_9.setHorizontalHeaderLabels(Global_pd_df_of_all_messeges.columns.to_list())

        for i in range(Global_pd_df_of_all_messeges.shape[0]):
            for j in range(Global_pd_df_of_all_messeges.shape[1]):
                item = QtWidgets.QTableWidgetItem(str(Global_pd_df_of_all_messeges.iloc[i, j]))
                self.tableWidget_9.setItem(i, j, item)

        if Global_pd_df_of_all_messeges.empty:
            self.tableWidget_9.setRowCount(1)

    @pyqtSlot()
    def clicked_on_pushButton_26(self):
        global Global_pd_df_of_all_messeges

        value = self.line_Edit_11.text()

        try:
            parsed_value = int(value)
            parsed_value=parsed_value-1
            if parsed_value>=0:
                if not self.IS_ConnectedTo_flag_signal_15:
                    self.flag_signal_15.connect(self.Show_list_of_act_websockets_if_widget)
```

```
                    self.IS_ConnectedTo_flag_signal_15=True
                comand_to_do=0
                self.Start_messege_meneger=Messeges_manager(parsed_value,self.flag_signal_14,self.flag_signal_15, comand_t
                self.Start_messege_meneger.start()

        except Exception as e:
            print(str(e))
            self.line_Edit_11.setText("1")

    @pyqtSlot()
    def clicked_on_pushButton_25(self):
        global Global_pd_df_of_all_messeges

        value = self.line_Edit_11.text()

        try:
            parsed_value = int(value)
            parsed_value=parsed_value-1
            if parsed_value>=0:
                if Global_pd_df_of_all_messeges.loc[parsed_value,"Status" ]=="Active":
                    comand_to_do=1
                    self.Start_messege_meneger=Messeges_manager(parsed_value,self.flag_signal_14,self.flag_signal_15, coma
                    self.Start_messege_meneger.start()
                else:
                    Global_pd_df_of_all_messeges=Global_pd_df_of_all_messeges.drop(parsed_value).reset_index(drop=True)
                    self.flag_signal_14.emit()
        except:
            print("error index")
            self.line_Edit_11.setText("1")


    @pyqtSlot()
    def clicked_on_pushButton_24(self):
        self.Save_mes=Find_db_and_Save_all_messeges()
        self.Save_mes.start()

    @pyqtSlot()
    def clicked_on_pushButton_23(self):
        self.Find_mes=Find_db_and_return_all_messeges(self.flag_signal_14)
```

```
    @pyqtSlot()
    def clicked_on_pushButton_22(self):
        fail_flag=0
        is_checked = self.checkBox_3.isChecked()
        selected_Name =''
        if is_checked:
            selected_Name = self.line_Edit_8.text()
            if re.fullmatch(r"[A-Z]+", selected_Name):
                pass
            else:
                fail_flag+=1
        else:
            selected_Name = self.combo_box_5.currentText()

        selected_Type = self.combo_box_6.currentText()
        selected_Subtype = self.combo_box_7.currentText()
        if selected_Subtype =='':
            fail_flag+=1

        selected_Value = self.line_Edit_9.text()
        selected_Value = selected_Value.replace(",", ".")
        try:
            selected_Value = float(selected_Value)
        except ValueError:
            fail_flag+=1
        selected_Message_text= self.line_Edit_10.text()

        status='Pass'
        if fail_flag>0:
            pass
        else:
            data_series = pd.Series({
                "Status": status,
                "Name": selected_Name,
                "Type": selected_Type,
                "Subtype": selected_Subtype,
                "Value": selected_Value,
                "Message text": selected_Message_text
```

```
            global Global_pd_df_of_all_messeges

            Global_pd_df_of_all_messeges.loc[len(Global_pd_df_of_all_messeges)]=data_series
            print(Global_pd_df_of_all_messeges)

            self.flag_signal_14.emit()
    @pyqtSlot()
    def on_combo_box_6_changed(self):
        selected_text = self.combo_box_6.currentText()
        if selected_text =='Price':
            items = ['+-','>=', '<=', '%']
            self.combo_box_7.clear()
            self.combo_box_7.addItems(items)
            self.combo_box_7.setCurrentIndex(0)
        if selected_text =='Position':
            self.combo_box_7.clear()
        if selected_text =='Deposit':
            self.combo_box_7.clear()

    @pyqtSlot()
    def on_checkBox_3_stateChanged(self):
        is_checked = self.checkBox_3.isChecked()
        if is_checked:
            self.combo_box_5.setHidden(True)
            self.line_Edit_8.setHidden(False)
        else:
            self.line_Edit_8.setHidden(True)
            self.combo_box_5.setHidden(False)

class Messeges_manager(QThread):
    def __init__(self, index_messege_in_df, flag_signal_14,flag_signal_15,comand_to_do):
        QThread.__init__(self)
        self.flag_changed = flag_signal_14
        self.flag_signal_15=flag_signal_15
        self.comand_to_do=comand_to_do
        self.index_messege_in_df=index_messege_in_df
```

```
    def run(self):
        global Global_pd_df_of_all_messeges
        global list_sockets_of_messeges

        if Global_pd_df_of_all_messeges.loc[self.index_messege_in_df, "Status"] in ["Pass", "Done", "Error"]:
            if Global_pd_df_of_all_messeges.loc[self.index_messege_in_df, "Type"]=="Price":
                Is_Use_websocket_already=False
                index_el=0
                while index_el<len(list_sockets_of_messeges):
                    if list_sockets_of_messeges[index_el].get_name()==Global_pd_df_of_all_messeges.loc[
                        self.index_messege_in_df, "Name"]:
                        list_sockets_of_messeges[index_el].add_messeges_to_list(Global_pd_df_of_all_messeges.loc[
                            self.index_messege_in_df])
                        Is_Use_websocket_already=True
                        break
                    index_el+=1

                if not Is_Use_websocket_already:
                    series = pd.Series()
                    series = Global_pd_df_of_all_messeges.loc[self.index_messege_in_df]
                    client = Messeges_Web_socket(series,self.flag_signal_15, self.flag_changed)
                    client.start()

                    list_sockets_of_messeges.append(client)
                Global_pd_df_of_all_messeges.loc[self.index_messege_in_df, "Status"]= "Active"
                self.flag_changed.emit()

        elif Global_pd_df_of_all_messeges.loc[self.index_messege_in_df, "Status"] == "Active":

            if Global_pd_df_of_all_messeges.loc[self.index_messege_in_df, "Type"]=="Price":
                Is_Use_websocket_already=False
                index_el=0
                while index_el<len(list_sockets_of_messeges):
                    if list_sockets_of_messeges[index_el].get_name()==Global_pd_df_of_all_messeges.loc[
                        self.index_messege_in_df, "Name"]:
                        list_sockets_of_messeges[index_el].turn_off_messege(Global_pd_df_of_all_messeges.loc[
                            self.index_messege_in_df])
                        Is_Use_websocket_already=True
                        break
```

```
                    index_el+=1
                if self.comand_to_do==0:
                    if Is_Use_websocket_already:
                        Global_pd_df_of_all_messeges.loc[self.index_messege_in_df, "Status"]= "Pass"
                        self.flag_changed.emit()
                    else:
                        Global_pd_df_of_all_messeges.loc[self.index_messege_in_df, "Status"]= "Error"
                        self.flag_changed.emit()
                elif self.comand_to_do==1:
                    if Is_Use_websocket_already:
                        Global_pd_df_of_all_messeges=Global_pd_df_of_all_messeges.drop(self.index_messege_in_df).reset_inde
                        self.flag_changed.emit()
                    else:
                        Global_pd_df_of_all_messeges.loc[self.index_messege_in_df, "Status"]= "Error"
                        self.flag_changed.emit()

class Messeges_Web_socket():
    def __init__(self,series,flag_signal_15,flag_signal_14):
        self.series = series.copy()
        self.flag_signal_15=flag_signal_15
        self.flag_signal_14=flag_signal_14
        self.Name_cripta = (self.series["Name"]).lower()
        stream_name = f'{self.Name_cripta}@markPrice'
        self.url='wss://fstream.binance.com/ws/%s' % stream_name
        self.wsa = websocket.WebSocketApp(self.url, on_message=self.on_message,on_error=self.on_error,on_close=self.on_clos
        columns = ["Status","Name", "Type", "Subtype", "Value", "Message text" ]
        self.series["Status"]="Active"
        self.List_of_active_messeges=pd.DataFrame(columns=columns)
        self.List_of_active_messeges.loc[len(self.List_of_active_messeges)]=self.series
        self.List_of_active_messeges_reserv=pd.DataFrame(columns=columns)
        self.List_of_active_messeges_reserv.loc[len(self.List_of_active_messeges_reserv)]=self.series
        self.cur_price=0

    def on_message(self, ws, message):
        data = json.loads(message)
        data_dict = {'time': data['E'], 'price': data['p']}
        data_dict['time'] = pd.to_datetime(data_dict['time'], unit='ms')
        data_dict['price'] = round(float(data_dict['price']), 3)
        self.cur_price=data_dict['price']
```

```
        index_el = 0
        while index_el < len(self.List_of_active_messeges):
            if self.List_of_active_messeges.loc[index_el, "Subtype"] == '>=':
                if data_dict['price']>= self.List_of_active_messeges.loc[index_el, "Value"]:
                    self.condition_complate(index_el)
                    if len(self.List_of_active_messeges)==0:
                        self.close_web_socket()
                    else:
                        continue

            elif self.List_of_active_messeges.loc[index_el, "Subtype"] == '<=':
                if data_dict['price']<= self.List_of_active_messeges.loc[index_el, "Value"]:
                    self.condition_complate(index_el)
                    if len(self.List_of_active_messeges)==0:
                        self.close_web_socket()
                    else:
                        continue

            elif self.List_of_active_messeges.loc[index_el, "Subtype"] == '+-':
                if self.List_of_active_messeges.loc[index_el, "Value"] > 0:
                    self.List_of_active_messeges.loc[index_el, "Subtype"] = '>='
                else:
                    self.List_of_active_messeges.loc[index_el, "Subtype"] = '<='
                self.List_of_active_messeges.loc[index_el, "Value"] += data_dict['price']

            elif self.List_of_active_messeges.loc[index_el, "Subtype"] == '%':
                if self.List_of_active_messeges.loc[index_el, "Value"] > 0:
                    self.List_of_active_messeges.loc[index_el, "Subtype"] = '>='
                    self.List_of_active_messeges.loc[index_el, "Value"] = round(
                        data_dict['price'] + data_dict['price']*0.01 * self.List_of_active_messeges.loc[index_el, "Value"]
                else:
                    self.List_of_active_messeges.loc[index_el, "Subtype"] = '<='
                    self.List_of_active_messeges.loc[index_el, "Value"] = round(
                        data_dict['price'] + data_dict['price']*0.01 * self.List_of_active_messeges.loc[index_el, "Value"]

            index_el += 1
        self.flag_signal_15.emit(self.series["Name"], data_dict['price'], len(self.List_of_active_messeges))
        print(data_dict['price'])
```

```
        if len(self.List_of_active_messeges)==0:
            self.close_web_socket()

    def add_messeges_to_list(self, new_series):
        self.new_series = new_series.copy()
        self.new_series["Status"]="Active"
        self.List_of_active_messeges.loc[len(self.List_of_active_messeges)]=self.new_series
        self.List_of_active_messeges_reserv.loc[len(self.List_of_active_messeges_reserv)]=self.new_series

    def turn_off_messege(self, new_weries):

        index_to_remove=-1
        index_el =0
        while index_el<len(self.List_of_active_messeges):
            if (self.List_of_active_messeges_reserv.loc[index_el, "Name"]==new_weries["Name"] and
            self.List_of_active_messeges_reserv.loc[index_el, "Type"]==new_weries["Type"] and
            self.List_of_active_messeges_reserv.loc[index_el, "Subtype"]==new_weries["Subtype"] and
            self.List_of_active_messeges_reserv.loc[index_el, "Value"]==new_weries["Value"]):
                index_to_remove=index_el
                break
            index_el+=1
        if index_to_remove>=0:
            self.List_of_active_messeges=self.List_of_active_messeges.drop(index_to_remove).reset_index(drop=True)
            self.List_of_active_messeges_reserv=self.List_of_active_messeges_reserv.drop(index_to_remove).reset_index(drop=True)
        else:
            pass

    def get_name(self):
        return self.series["Name"]

    def condition_complate(self, index_el):

        self.Start_condition_handler=Condition_handler(self.List_of_active_messeges.loc[index_el],self.cur_price)
        self.Start_condition_handler.start()

        global Global_pd_df_of_all_messeges

        index_el_pd_df=0
        index_to_remove=-1
```

```
        while index_el_pd_df< len(Global_pd_df_of_all_messeges):

            if (Global_pd_df_of_all_messeges.loc[index_el_pd_df,"Status" ]==self.List_of_active_messeges_reserv.loc[index_el, "Status"]
            Global_pd_df_of_all_messeges.loc[index_el_pd_df,"Name" ]==self.List_of_active_messeges_reserv.loc[index_el, "Name"] and
            Global_pd_df_of_all_messeges.loc[index_el_pd_df,"Type" ]==self.List_of_active_messeges_reserv.loc[index_el, "Type"] and
            Global_pd_df_of_all_messeges.loc[index_el_pd_df,"Subtype" ]==self.List_of_active_messeges_reserv.loc[index_el, "Subtype"] a
            Global_pd_df_of_all_messeges.loc[index_el_pd_df,"Value" ]==self.List_of_active_messeges_reserv.loc[index_el, "Value"]):
                index_to_remove=index_el_pd_df
                break
            index_el_pd_df+=1
        if index_to_remove>=0:
            Global_pd_df_of_all_messeges.loc[index_to_remove, "Status"] = "Done"
            self.flag_signal_14.emit()

        else:

            pass

        self.List_of_active_messeges=self.List_of_active_messeges.drop(index_el).reset_index(drop=True)
        self.List_of_active_messeges_reserv=self.List_of_active_messeges_reserv.drop(index_el).reset_index(drop=True)



    def on_error(self, ws, error):
        print(error)
    def on_close(self, ws,close_status_code, close_msg ):
        print("Close")
    def on_open(self, ws):
        print("on_open")
    def close_web_socket(self):

        global list_sockets_of_messeges
        for socket in list_sockets_of_messeges:
            if socket.get_name() == self.series["Name"]:
                list_sockets_of_messeges.remove(socket)
                break
        self.wsa.close()
        print("close_web_socket()")
```

```
    def start(self):
        self.thread_io = Thread(target=self.wsa.run_forever)
        self.thread_io.start()

class BinanceWebsocket():
    def __init__(self, flag_signal,criptoname,criptotimeframe):
        self.url = 'wss://stream.binance.com/stream'
        self.criptoname=criptoname.lower()
        self.criptotimeframe=criptotimeframe
        stream_name = f'{self.criptoname}@markPrice'
        stream_name_2='<atomusdt@kline_15m'
        self.url_2='wss://fstream.binance.com/ws/%s' % stream_name
        self.flag_signal=flag_signal
        self.wsa = websocket.WebSocketApp(self.url_2, on_message=self.on_message,on_error=self.on_error,on_close=self.on_close)

    def on_message(self, ws, message):
        data = json.loads(message)
        data_dict = {'time': data['E'], 'price': data['p']}
        data_dict['time'] = pd.to_datetime(data_dict['time'], unit='ms')
        data_dict['price'] = round(float(data_dict['price']), 3)
        print(data_dict)
        global Data_dict_ticks
        Data_dict_ticks = data_dict
        self.flag_signal.emit()

    def on_error(self, ws, error):
        print(error)
    def on_close(self, ws,close_status_code, close_msg ):
        print("Close")
    def on_open(self, ws):
        print("on_open")
    def close_web_socket(self):
        self.wsa.close()
        print("close_web_socket()")
    def start(self):
        self.thread_io = Thread(target=self.wsa.run_forever)
        self.thread_io.daemon = True
        self.thread_io.start()
```

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА
на тему:
“АВТОМАТИЗОВАНА СИСТЕМА ОНЛАЙН МОНІТОРИНГУ БІРЖОВИХ ЦІН КРИПТОВАЛЮТ НА ОСНОВІ МОВИ ПРОГРАМУВАННЯ PYTHON”

Виконав: студент групи ІСДМ-61
Олексій МАЩЕНКО

Керівник: к.т.н., доцент кафедри ІСТ
Оксана ТКАЛЕНКО

Київ - 2024

Об'єкт дослідження - процеси моніторингу, аналізу та візуалізації ринкових даних у системах обробки інформації.

Предмет дослідження - технології та інструменти моніторингу, включаючи API, веб-сокети, бази даних, засоби візуалізації та алгоритми автоматизації аналізу даних.

Мета роботи – розробка автоматизованої системи моніторингу біржових цін криптовалют із використанням сучасних технологій збору, обробки та візуалізації даних

Завдання:

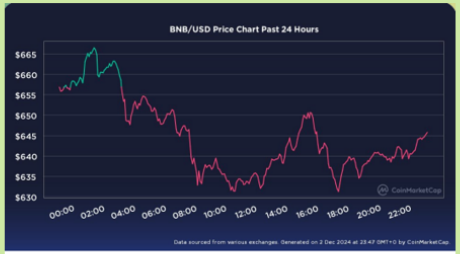
- Дослідити технології збору та обробки даних, включаючи API та web-socket для підключення до зовнішніх сервісів і отримання ринкових даних у реальному часі.
- Обрати базу даних для локального зберігання даних.
- Розробити графічний інтерфейс користувача за допомогою PyQt6 і CSS для зручної візуалізації даних і налаштування параметрів системи.
- Створити систему сповіщень за допомогою Telegram-бота з бібліотекою aiogram для оперативного інформування користувачів про зміни цін.
- Провести тестування і оцінити ефективність розробленої системи на реальних даних біржі та скласти звіт.

Система моніторингу біржових цін – це програмно-апаратний комплекс, що забезпечує збір, обробку, аналіз та візуалізацію інформації про зміни вартості активів на біржових ринках у реальному часі. Така система дозволяє відстежувати коливання цін, аналізувати ринкові тренди, зберігати історичні дані та оперативно інформувати користувачів про важливі зміни для прийняття рішень.

Особливості:

- Можуть одночасно працювати з великим обсягом даних
- Обробляти дані, відсіювати непотрібне, проводити агрегацію
- Візуалізувати дані
- Перевіряти виконання чи невиконання умови
- При потребі відправляти повідомлення чи автоматично виконувати певну дію

ОГЛЯД ІСНУЮЧИХ СИСТЕМ ОНЛАЙН МОНІТОРИНГУ

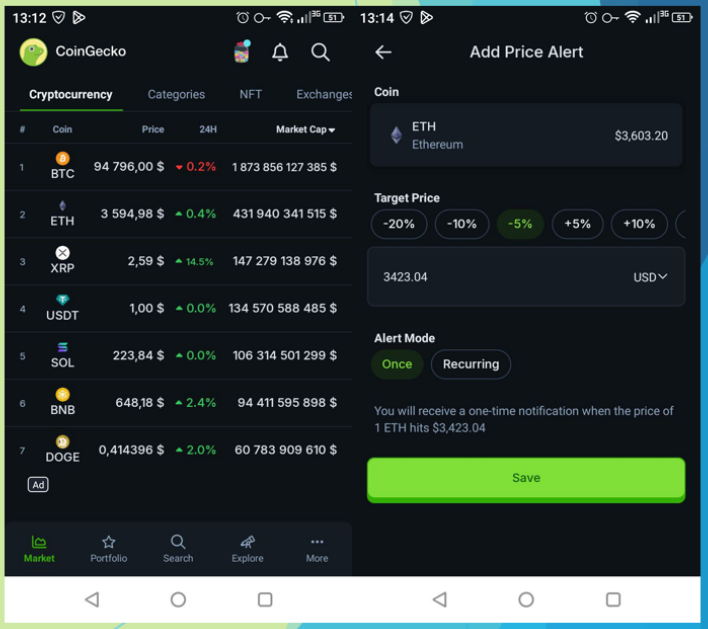


Приклад відповіді від телеграм-бота CoinMarketCap

Ключові особливості:

- Містять актуальну інформацію про назву, ціну, об’єм, зміни за останній період часу
- Більшість систем мають функцію візуалізації історичних даних за допомогою графіків
- Деякі системи надають можливість встановити оповіщення через месенджер або push-повідомлення від мобільного застосунку

Приклад роботи мобільного застосунку CoinGecko



Технологія API

API (Application Programming Interface) — це набір визначень, протоколів і інструментів, що дозволяють одному програмному компоненту взаємодіяти з іншим. API виступає програмним інтерфейсом між різними системами або їх компонентами, надаючи стандартизований спосіб обміну даними.

Методи HTTP

№ п/п	Назва методу	Призначення
1	GET	Отримати наявний ресурс
2	POST	Створити новий ресурс
3	PUT	Оновити існуючий ресурс
4	PATCH	Часткове оновлення існуючого ресурсу
5	DELETE	Видалити ресурс

API працює за допомогою запитів і відповідей між клієнтом і сервером. Найпоширеніші типи запитів це GET, POST, PUT та DELETE.

Технологія веб-сокетів

WebSocket — це сучасний мережевий протокол, який забезпечує двостороннє спілкування між клієнтом і сервером через єдине довготривале підключення.

- Дозволяє надсилати дані в обидва напрямки без необхідності повторного встановлення з'єднання
- активно використовуються там, де потрібна висока швидкість обміну даними в реальному часі

Застосовується в фінансових системах, системах онлайн моніторингу, в онлайн чатах, де необхідна швидка реакція на повідомлення.

Порівняння WebSocket та API

Параметр	WebSocket	API(HTTP/HTTPS)
З'єднання	Постійне (з'єднання відкривається раз і працює далі).	Кожен запит створює нове з'єднання.
Швидкість	Швидше, оскільки не створюється нове з'єднання.	Вища затримка через багаторазове створення HTTP-запитів.
Напрямок зв'язку	Двосторонній: сервер може ініціювати передачу даних клієнту.	Односторонній: сервер відповідає лише на запити клієнта.
Типи передачі	Підходить для систем, що працюють в режимі реального часу, дані передаються без затримок.	Підходить для періодичної передачі даних (REST API)
Об'єм даних	Підтримує потоки даних без повторення HTTP-заголовків.	Включає HTTP-заголовки у кожному запиті, що збільшує розмір передачі.

Графічний інтерфейс

(GUI - Graphical User Interface) — це сукупність засобів взаємодії користувача з програмою через графічні елементи, такі як кнопки, вікна, меню, поля введення та інші компоненти, які забезпечують інтуїтивно зрозуміле та зручне управління функціоналом програми.

Вимоги до графічного інтерфейсу:

1.Зрозумілий – без потреби у додаткових інструкціях.

2.Неперевантажений – дизайн мінімалістичним, без зайвих елементів, які відволікають увагу.

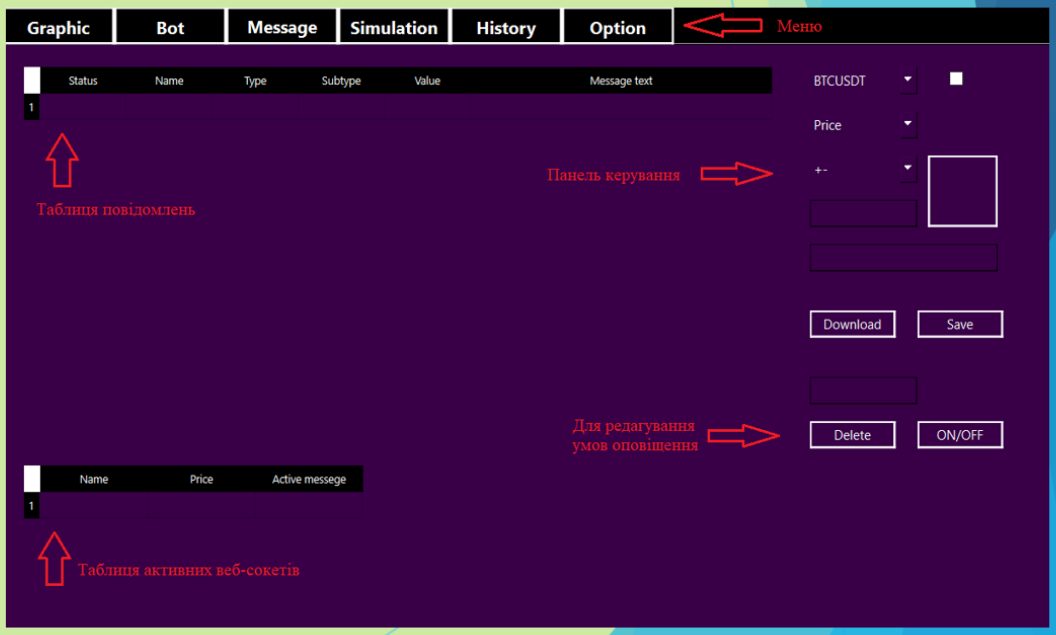
3.Інформативний – надавати користувачу необхідну інформацію в зручному для сприйняття вигляді.

4.Функціональний – забезпечувати легкий доступ до основних функцій системи.

5.Динамічний – дані повинні оновлюватись у реальному часі.

6.Ефективний – забезпечувати швидке виконання користувацьких операцій.

Макет графічного інтерфейсу



Побудова фінансового графіку за історичними даними з біржі

Використовую бібліотеку lightweight-charts та PyQt6

```python
@pyqtSlot()
def Chart_update_candle(self):
    layout = QVBoxLayout()
    widget = QWidget()
    widget.setLayout(layout)
    self.chart = QtChart(widget)
    global P_dataframe_chart
    P_dataframe_chart['date'] = pd.to_datetime(P_dataframe_chart['date'], unit='ms')
    self.chart.set(P_dataframe_chart)
    layout.addWidget(self.chart.get_webview())
    layout.setContentsMargins(0, 0, 0, 0)

    proxy_widget = QGraphicsProxyWidget()
    proxy_widget.setWidget(widget)

    scenes = QtWidgets.QGraphicsScene()
    scenes.addItem(proxy_widget)

    widget.resize(self.graphicsView.size())
    self.graphicsView.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarPolicy.ScrollBarAlwaysOff)
    self.graphicsView.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarPolicy.ScrollBarAlwaysOff)
    self.graphicsView.setScene(scenes)

    if(self.IS_web_socket_Use):
        pass
    else:
        self.flag_signal_2.connect(self.Update_tiks_online) # метод для візуалізації тіків
        client = BinanceWebsocket(self.flag_signal_2)
        client.start()
        self.client_web_socket=client
        self.IS_web_socket_Use=True
```

Приклад фінансового графіку

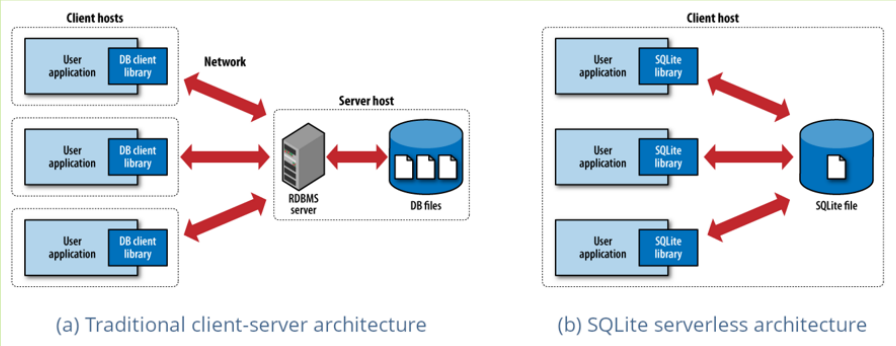
Для побудови графіку потрібно:

- Отримати історичні дані від біржі в форматі OCHL
- Отримувати дані про поточну ціну в режимі онлайн

База даних SQLite

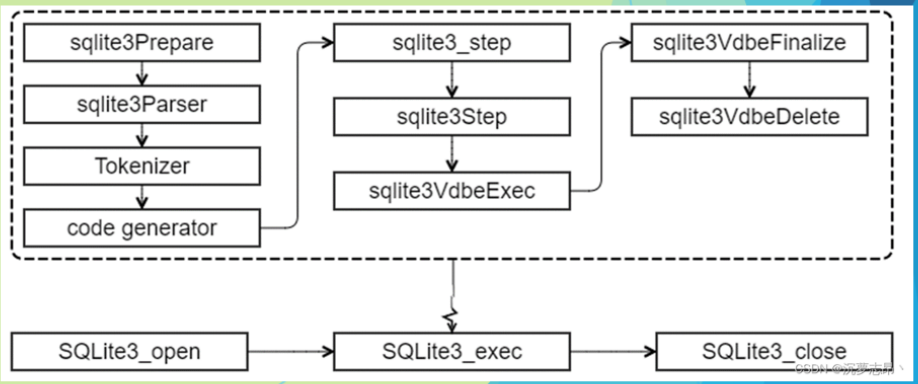
Це вбудована реляційна база даних, яка не потребує окремого серверного процесу та налаштування.

Активно використовується для локального зберігання даних в автономних системах та додатках, де немає необхідності в централізованому сервері.



Різниця між серверною архітектурою і локальною для бази даних

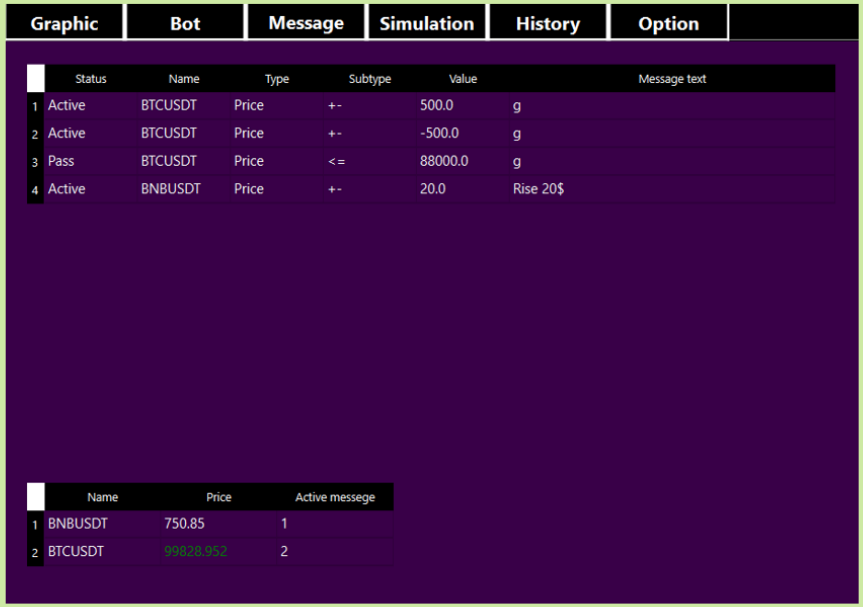
Детальний опис етапів роботи SQLite



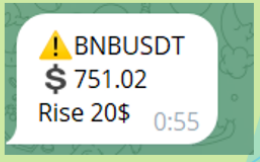
Приклад використання системи моніторингу

Функції які надає програма:

- Додавання, видалення умов, їх активація та деактивації
- Збереження та завантаження з бази даних умов
- Перевірка виконання умов на актуальних даних, отриманих з біржі Binance, в режимі онлайн
- Перегляд фінансового графіку, що будується на історичних даних та оновлюється в режимі онлайн
- Можливість налаштувати оповіщення, що прийде до телеграм-боту



Повідомлення, що прийшло до чату телеграм-бота:



ВИСНОВКИ

Було проведено всебічне дослідження, спрямоване на розробку автоматизованої системи моніторингу біржових цін криптовалют на основі мови Python.

Дослідження підтвердило актуальність та доцільність створення таких систем, оскільки динамічний характер криптовалютного ринку вимагає від користувачів постійного моніторингу, аналізу даних та оперативного ухвалення рішень.

На основі аналізу сучасних технологій і фреймворків було розроблено програмне забезпечення з модульною архітектурою, яке дозволяє виконувати основні функції та задачі, що було поставлено перед системою. Було створено модулі для інтеграції з API біржі Binance, взаємодії з Telegram-ботом для сповіщення користувача, модулі для обробки запитів і виконання умов.

Важливим компонентом є графічний інтерфейс, який забезпечує зручний доступ до даних, їх візуалізацію та введення нових параметрів.

Дана система моніторингу може бути інтегрована в автоматизовані системи торгівлі, адаптована для роботи з іншими фінансовими інструментами або ринками.

Система має високий потенціал для масштабування, що дозволяє її використовувати в ширшому контексті фінансового аналізу.

Апробація результатів магістерської роботи:

Мащенко О.Ю - «Система моніторингу біржових цін криптовалют». Тези доповіді на II Всеукраїнській науково-технічній конференції «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу». – Київ, 18 листопада 2024 року.

Мащенко О.Ю - «Веб-технології в системах моніторингу фінансових даних». Тези доповіді на VII Всеукраїнській науково-технічній конференції «Комп'ютерні технології: інновації, проблеми, рішення». – Київ, 2-3 грудня 2024 року.

ДЯКУЮ ЗА УВАГУ!