Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
Навчально-науковий інститут комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету(відділення))
Кафедра інформаційних систем та комп'ютерного моделювання
(повна назва кафедри (предметної циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: "Розробка автоматизованого торгового бота для трейдингу криптовалюти"

Виконав студент 4 курсу, групи ІСТ-41
спеціальності
126 "Інформаційні системи та технології"
(шифр і назва напрямку підготовки спеціальності)

Мавдрик П.А.
(прізвище, ініціали)

Керівники: Проць А.М., Флуд Л.О.
(прізвище, ініціали)

Рецензент: Прохук Ю.С.
(прізвище, ініціали)

Львів-2024

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
ННІ комп'ютерних наук та інформаційних технологій
Кафедра інформаційних систем та комп'ютерного моделювання
Рівень вищої освіти перший(бакалаврський)
Спеціальність 126 " Інформаційні системи та технології "

ЗАТВЕРДЖУЮ:
Завідувач кафедри ІСКМ
_____ Сторожук О.Л.
„06" 02 2024.

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Мавдрик Павло Андрійович
(прізвище, ім'я, по батькові)

1.Тема роботи: "Розробка автоматизованого торгового бота для трейдингу криптовалюти"

керівники роботи: Проць А.М, Флуд Л.О.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "6" лютого 2024 року, №С-87

2.Термін подання студентом проекту(роботи) 10 червня 2024р

3. Вихідні дані до проекту (роботи) Торговий бот, що дозволить автоматизувати процеси купівлі та продажу криптовалют на платформі Binance.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Перелік скорочень та умовних позначень. Вступ.
Розділ 1. Стан проблемної області.
Розділ 2. Інформаційне та математичне забезпечення.
Розділ 3. Програмне та технічне забезпечення.
Висновки. Список використаних джерел. Додатки.

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

Слайди для доповіді.

6. Консультанти розділів проекту (роботи)

7. Дата видачі завдання 7 лютого 2024р.

КАЛЕНДАРНИЙ ПЛАН

№, з/п	Етапи бакалаврської роботи	Термін виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів.	07.02.24-20.02.24	Виконано
2.	Постановка задачі і її формалізація	21.02.24-05.03.24	Виконано
3.	Виконання вхідного етапу технології	06.03.24-25.03.24	Виконано
4.	Реалізація головних класів проекту	26.03.24-14.04.24	Виконано
5.	Виконання етапу відлагодження проєкту	15.04.24-10.05.24	Виконано
6.	Виконання етапу впровадження та випуску бета-версії.	13.05.24-02.06.24	Виконано
7.	Оформлення записки до дипломного проекту.	03.06.24-10.06.24	Виконано

Студент _____ Мавдрик П.А.
(підпис) (прізвище та ініціали)

Керівники роботи _____ Проць А.М.
(підпис) (прізвище та ініціали)

_____ Флуд Л.О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 46 сторінок пояснювальної записки, 38 рисунків, 1 додаток, 15 джерел.

Ця дипломна робота присвячена розробці та аналізу торгового бота, створеного за допомогою Python для автоматизації торгових операцій на криптовалютній біржі Binance. Метою проєкту є підвищення ефективності торгівлі шляхом автоматизації процесів купівлі та продажу криптовалют. Робота включає в себе детальний опис архітектури бота, його основних компонентів та алгоритмів, використаних у процесі розробки.

Ключові слова: *криптобіржа, автоматизована торгівля, Python, Binance API.*

ABSTRACT

The thesis contains 46 pages of explanatory notes, 38 figures, 1 appendix, and 15 references.

This thesis is dedicated to the development and analysis of a trading bot, created using Python to automate trading operations on the cryptocurrency exchange Binance. The project's goal is to enhance trading efficiency through the automation of cryptocurrency buying and selling processes. The work includes a detailed description of the bot's architecture, its key components, and the algorithms used in the development process.

Keywords: *cryptocurrency exchange, automated trading, Python, Binance API.*

ТЕХНІЧНЕ ЗАВДАННЯ

Реалізувати торгового бота, що дозволить автоматизувати процеси купівлі та продажу криптовалют на платформі Binance з використанням мови програмування Python. Бот повинен надавати можливість виконувати торговельні операції на основі попередньо встановлених параметрів користувачем, а також реагувати на зміни ринкових умов в реальному часі.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface (Інтерфейс прикладного програму-вання)

АТС – Автоматизована торгова система.

BTC – Біткоїн.

USDT – криптовалюта привязана до долара США.

ВСТУП

У сучасному світі, де темпи технологічного прогресу стрімко зростають, фінансові ринки постійно еволюціонують, а особливо активно розвивається сектор криптовалют. Завдяки своїй високій волатильності та ліквідності, криптовалютні ринки приваблюють як індивідуальних інвесторів, так і великі фінансові установи. Однак, для успішної торгівлі на таких швидкоплинних ринках, де ціни можуть змінюватися кожну секунду, потрібен швидкий аналіз великої кількості даних та оперативне прийняття рішень. У цьому контексті автоматизація торгових процесів за допомогою торгових ботів стає не просто перевагою, а необхідністю.

Об’єктом дослідження є використання торгового бота для автоматизації процесів купівлі та продажу криптовалют на біржі Binance.

Метою роботи є розроблення програмного забезпечення для торгового бота, який буде здійснювати автоматизовану торгівлю криптовалютами на біржі Binance, забезпечуючи ефективність і безпеку операцій.

Практичне значення роботи полягає у підвищенні ефективності торгівлі шляхом мінімізації людського фактора, оптимізації торговельних стратегій і забезпеченні стабільної роботи бота в умовах високої волатильності ринку.

Предметом дослідження є алгоритми та технічні рішення, що використовуються для реалізації автоматизованої торгівлі криптовалютами на платформі Binance.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Огляд проблемної області.

З розвитком криптовалют як нового цифрового фінансового інструменту традиційні фінансові ринки проходять через трансформацію. Криптовалютні біржі, наприклад Binance, зайняли ключову позицію у цій новій економічній моделі. Вони надають платформу (рис.1.1) для купівлі-продажу різноманітних віртуальних цифрових активів, даючи користувачам можливість інвестувати кошти та отримувати прибуток від коливань курсів криптовалют [9].

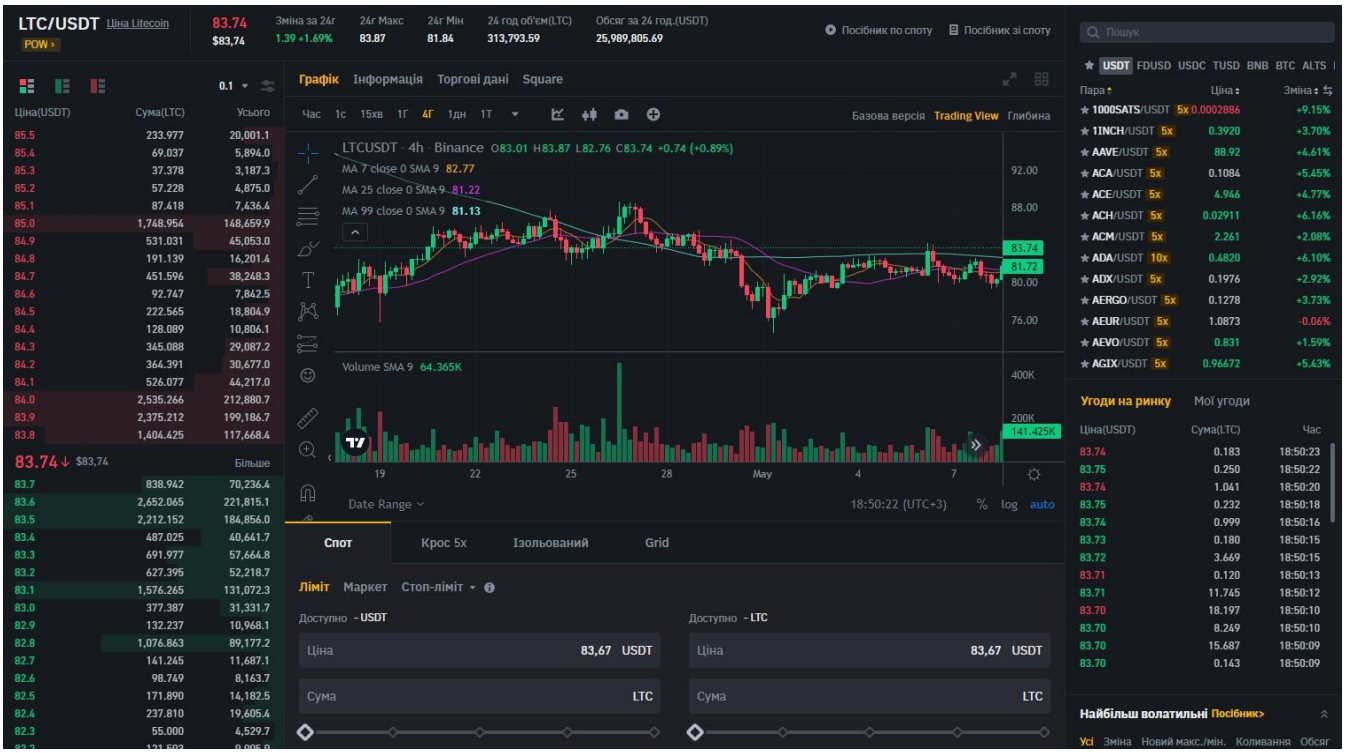

Рисунок 1.1– Криптовалютна біржа Binance

Трейдери на криптовалютних ринках стикаються з серйозними викликами через надзвичайно мінливу природу цих активів. Різкі коливання курсів, нестійкі ціни та стрімкі зміни на ринку створюють складні умови для прийняття виважених торгових рішень, адже ситуація може кардинально змінитися буквально за лічені секунди. Для успішної торгівлі необхідно постійно відстежувати ринкову кон'юнктуру, оперативно аналізувати величезні масиви даних та вчасно проводити угоди купівлі-продажу.

Одним з дієвих рішень для подолання складнощів на криптовалютних ринках є застосування торгових ботів. Торгові боти - це спеціальні комп'ютерні програми, здатні самостійно здійснювати торгові операції купівлі-продажу за заздалегідь запрограмованими алгоритмами [10, 12]. Їх можна налаштувати на виконання різноманітних торгових стратегій, як от скальпінг, арбітраж, трендовий трейдинг тощо. Ключовими перевагами використання торгових ботів є:

- **Автоматизація процесів**: Боти дозволяють автоматизувати рутинні торговельні завдання, що звільняє трейдерів від необхідності постійного моніторингу ринку.
- **Швидкість реакції**: Програмні алгоритми можуть реагувати на зміни ринку значно швидше, ніж це може зробити людина.
- **Об'єктивність**: Боти приймають рішення на основі заданих правил і алгоритмів, що виключає вплив людських емоцій на торговельні операції.
- **Підвищення ефективності**: За рахунок точного виконання стратегії боти можуть допомогти трейдерам оптимізувати свої прибутки та мінімізувати збитки.

Однак, розробка торгового бота також супроводжується певними проблемами та викликами. Одним з найважливіших аспектів є забезпечення належного рівня безпеки, адже при роботі з реальними грошовими коштами необхідно мати надійний захист від шахрайських дій та несанкціонованого доступу. Крім того, ефективність роботи бота значною мірою залежить від якості закладених в нього торгових алгоритмів та здатності пристосовуватися до мінливих ринкових обставин.

У цьому контексті, розробка торгового бота для платформи Binance з використанням мови Python є актуальною темою дослідження. Це дозволить не тільки зрозуміти механізми роботи торгових ботів, але й створити практичний інструмент для автоматизації торгівлі криптовалютами, що підвищить ефективність і безпеку операцій на ринку [7, 11].

1.2. Специфіка криптовалютних ринків

Однією з визначальних рис криптовалютних ринків є наявність підвищеної волатильності, що виражається у різких коливаннях цін на віртуальні активи навіть за відносно короткі проміжки часу. Ця особливість одночасно створює сприятливі умови для отримання високих прибутків, однак несе в собі й ризики значних фінансових втрат. Ключовими чинниками, що зумовлюють волатильність на досліджуваних ринках, є:

- **Новини та події**: Інформаційний простір має величезний вплив на ринок криптовалют. Наприклад, регуляторні зміни, технологічні оновлення або інциденти безпеки можуть призвести до миттєвих змін цін.
- **Ринкові спекуляції**: Велика кількість учасників ринку займаються спекулятивними операціями, намагаючись скористатися короткостроковими коливаннями цін. Це посилює волатильність і робить ринок менш передбачуваним.
- **Ліквідність**: Рівень ліквідності ринку може значно варіюватися залежно від криптовалюти та біржі. Недостатня ліквідність може призводити до значних цінових стрибків при виконанні великих ордерів.

1.3. Технічні аспекти торгівлі на криптовалютних ринках

Торгівля на криптовалютних біржах вимагає високого рівня технічних знань і вмінь. Сучасні трейдери використовують різноманітні інструменти для аналізу ринку та виконання торговельних операцій, включаючи:

- **Технічний аналіз**: Це метод прогнозування майбутніх рухів цін на основі історичних даних. Він включає в себе аналіз графіків цін, обсягу торгів, а також різних індикаторів і моделей.
- **Алгоритмічна торгівля**: Використання алгоритмів для автоматизації торговельних процесів. Алгоритмічні трейдери розробляють стратегії, що дозволяють автоматично купувати або продавати активи на основі певних умов або тригерів.

- **API для торгівлі**: Більшість криптовалютних бірж надають інтерфейси прикладного програмування (API), що дозволяють автоматизувати процеси торгівлі. Це включає в себе можливість отримання ринкових даних у реальному часі, розміщення ордерів і управління ризиками [1, 4].

1.4. Економічні фактори та ризики

Економічні фактори також відіграють важливу роль у торгівлі криптовалютами. Серед них можна виділити:

- **Регуляторні зміни:** Закони та правила, що регулюють криптовалютний ринок, постійно змінюються. Регуляторні нововведення можуть як позитивно, так і негативно впливати на ринок.
- **Інфраструктура ринку:** Розвиток інфраструктури, такої як криптовалютні біржі, гаманці та платіжні системи, впливає на зручність та безпеку торгівлі криптовалютами.
- **Економічна стабільність:** Глобальні економічні події, такі як фінансові кризи або зміни у валютній політиці, можуть впливати на ринок криптовалют. Наприклад, економічна нестабільність у деяких країнах може призводити до збільшення попиту на криптовалюти як на альтернативу традиційним валютам.

1.5. Роль автоматизованих торгових систем

Автоматизовані торгові системи (АТС), відомі також як торгові боти, є вагомим інструментарієм у практиці сучасного трейдингу. Дані програмно-апаратні комплекси дозволяють автоматизувати виконання рутинних операційних процедур, забезпечувати оперативне реагування на зміни ринкової кон'юнктури та реалізовувати складні торговельні стратегії. Основними перевагами використання торгових ботів у криптовалютній торгівлі є:

- **Швидкість:** Боти можуть виконувати операції набагато швидше, ніж це може зробити людина. Це особливо важливо на ринках з високою волатильністю, де затримка в декілька секунд може призвести до значних фінансових втрат.
- **Точність:** АТС виконують операції точно відповідно до заданих параметрів, виключаючи можливість людських помилок.
- **Цілодобова робота:** Боти можуть працювати 24/7 без перерв, забезпечуючи постійний моніторинг ринку та виконання торгових стратегій навіть тоді, коли трейдер спить або зайнятий іншими справами.
- **Об'єктивність:** АТС приймають рішення на основі заданих алгоритмів та ринкових даних, виключаючи емоційний фактор, який може негативно впливати на прийняття рішень.

1.6. Виклики та обмеження

Незважаючи на численні переваги, автоматизовані торгові системи також мають свої виклики та обмеження. Серед основних можна виділити:

- **Складність налаштування:** Створення та налаштування ефективного торгового бота вимагає глибоких знань у галузі програмування, математики та фінансів. Недостатньо підготовлені користувачі можуть створити бота, який буде працювати неефективно або навіть приносити збитки.
- **Ризики безпеки:** Зберігання API ключів та інших конфіденційних даних на сторонніх серверах або у відкритому доступі може призвести до несанкціонованого доступу та фінансових втрат.
- **Залежність від зовнішніх факторів:** Навіть найкращий торговий бот не застрахований від впливу зовнішніх факторів, таких як раптові зміни ринку, технічні збої на біржі або регуляторні зміни.
- **Висока конкуренція:** Криптовалютний ринок є дуже конкурентним середовищем, де безліч трейдерів використовують різноманітні стратегії та боти. Це може призводити до зниження ефективності окремих стратегій через перенасиченість ринку.

Криптовалютний ринок являє собою складну та високо динамічну систему, в якій автоматизовані торгові комплекси відіграють роль важливого інструментального засобу підвищення ефективності трейдингової діяльності. Впровадження торгових ботів дозволяє учасникам ринку оперативно реагувати на зміни кон'юнктури, мінімізувати ризики та оптимізувати застосовувані торговельні стратегії. Проте успішна експлуатація подібних автоматизованих систем вимагає від розробників та користувачів глибоких знань і комплексного розуміння як технічних, так і економічних аспектів функціонування криптовалютного ринку. Відтак, дослідження та розробка ефективних торгових ботів залишаються актуальними та важливими завданнями для численних трейдерів, фахівців з розробки програмного забезпечення та наукових працівників в цій галузі.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. API Binance та його методи

Інформаційне забезпечення дипломної роботи ґрунтується на використанні програмного інтерфейсу (API) однієї з провідних світових криптовалютних бірж Binance для отримання та обробки даних щодо обігу різноманітних фінансових активів у цифровому форматі [1, 4]. Біржова платформа Binance надає зручний набір інструментів для реалізації автоматизованих торговельних операцій через API. Ключовими аспектами інформаційного забезпечення дипломного дослідження є:

- **API ключі:** Для доступу до даних біржі та виконання торгових операцій використовуються API ключі (api_key та api_secret). Вони забезпечують безпеку та ідентифікацію користувача.
- **Отримання даних про ціну активу:** Функція ***get_price(symbol)*** використовує метод ***client.get_symbol_ticker(symbol=symbol)*** для отримання поточної ціни заданого активу.
- **Баланс користувача:** Функція ***get_balance(asset)*** викликає ***client.get_asset_balance(asset=asset)*** для отримання інформації про доступні кошти користувача.
- **Торгові операції:**
    a. ***sell_asset(symbol, quantity)*** виконує продаж активу за ринковою ціною.
    b. ***buy_asset_limit(symbol, quantity, price)*** розміщує лімітне замовлення на купівлю активу.
- **Інформація про лот:** Функція ***get_lot_size(symbol)*** отримує дані про крок кількості, мінімальну та максимальну кількість активу для торгів.
- **Регулювання кількості активу:** ***adjust_quantity(quantity, step_size)*** округлює кількість активу відповідно до кроку кількості.
- **Інтерактивні запити користувача:**
    - ***request_target_prices()*** запитує у користувача межі ціни для продажу.
    - ***request_purchase_info()*** збирає інформацію про символ активу, кількість і ціну покупки.

- ***request_sell_info()*** запитує інформацію для продажу активу.

2.2. Математичне забезпечення

Математичне забезпечення включає методи та алгоритми, які використовуються для обробки фінансових даних, розрахунків та прийняття рішень у торгівлі. Основні аспекти математичного забезпечення включають:

- **Округлення кількості активу:** Функція ***adjust_quantity(quantity, step_size)*** (рис.2.1) використовує операцію цілочисельного ділення з подальшим множенням, щоб округлити кількість активу до найближчого допустимого значення:

```
def adjust_quantity(quantity, step_size):
    return round((quantity // step_size) * step_size, 8)
```

Рисунок 2.1– Функція *adjust_quantity*()

- **Визначення цільових цін:** Вхідні дані користувача про межі ціни та кількість активу для торгів використовуються для побудови логіки продажу. Наприклад, функція ***request_target_prices()*** (рис.2.2) запитує у користувача нижню та верхню межі ціни для продажу:

```
def request_target_prices():
    lower_price = float(input("Введіть нижню межу ціни для продажу: "))
    upper_price = float(input("Введіть верхню межу ціни для продажу: "))
    return lower_price, upper_price
```

Рисунок 2.2– Функція *request_target_prices*().

- **Алгоритм торгівлі:** Основна торгова логіка, реалізована у функції ***trading_logic()*** (рис.2.3), включає перевірку поточної ціни активу та виконання торгових операцій на основі встановлених цільових цін та кількостей:

Математичні обчислення та логіка в цьому проєкті забезпечують точність і ефективність автоматизованої торгівлі на криптовалютній біржі, що дозволяє

```python
def trading_logic():
    """Основна торгова логіка."""
    while True:
        action = input("Введіть 'buy' для купівлі, 'sell' для продажу, або 'exit' для виходу: ").lower()
        if action == 'buy':
            try:
                symbol, amount, buy_price = request_purchase_info()
                lot_size_info = get_lot_size(symbol)
                quantity = adjust_quantity(amount / buy_price, lot_size_info['stepSize'])
                result = buy_asset_limit(symbol, quantity, buy_price)
                print("Результат купівлі:", result)
            except Exception as e:
                print(f"Сталася помилка при купівлі: {e}")
        elif action == 'sell':
            try:
                symbol, asset, lower_quantity, upper_quantity, lower_price, upper_price = request_sell_info()
                if symbol:
                    while True:
                        current_price = get_price(symbol)
                        print(f"Поточна ціна {symbol}: {current_price}")
                        if current_price ⩽ lower_price:
                            print(f"Виконується продаж {lower_quantity} {asset} за нижньою ціною {current_price}")
                            result = sell_asset(symbol, lower_quantity)
                            print("Результат продажу за нижньою межею:", result)
                            break
                        elif current_price ⩾ upper_price:
                            print(f"Виконується продаж {upper_quantity} {asset} за верхньою ціною {current_price}")
                            result = sell_asset(symbol, upper_quantity)
                            print("Результат продажу за верхньою межею:", result)
                            break
                        time.sleep(10)
            except Exception as e:
                print(f"Сталася помилка при продажу: {e}")
        elif action == 'exit':
            print("Вихід з програми.")
            break
        else:
            print("Введено неправильну команду. Будь ласка, введіть 'buy', 'sell' або 'exit'.")

        print("Операція завершена. Що далі?")
```

Рисунок 2.3– Функція *trading_logic*().

користувачам приймати обґрунтовані рішення на основі реальних даних та встановлених параметрів [13].

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Загальний огляд програмного забезпечення.

Програмне забезпечення для торгового бота складається з кількох основних компонентів, кожен з яких виконує специфічні функції для забезпечення ефективної роботи системи:

- **Мова програмування:**
  - **Python:** Вибір на користь Python обумовлений його простотою, зручністю та широкою підтримкою бібліотек для роботи з API та обробки даних.
- **Бібліотеки та модулі:**
  - **binance:** Офіційна бібліотека для взаємодії з Binance API, яка дозволяє виконувати запити для отримання інформації та здійснення торгових операцій.
  - **time:** Вбудована бібліотека Python для управління затримками в обробці запитів, що особливо важливо при реалізації циклів, які очікують зміни ринкових умов.
  - **sys:** Вбудована бібліотека Python для роботи з системними функціями, такими як завершення програми та обробка командного рядка.
- **Функції:**
  - **Функції для отримання інформації:** Ці функції забезпечують отримання поточних даних про ринок, таких як ціна активу та баланс користувача.
  - **Функції для виконання торгових операцій:** Ці функції реалізують купівлю та продаж активів через Binance API.
  - **Функції для взаємодії з користувачем:** Ці функції забезпечують введення даних користувачем, таких як цільові ціни та кількості для торгових операцій.

3.2. Архітектура програмного забезпечення.

Програмне рішення створене з використанням модульного підходу, завдяки якому забезпечується можливість легкого масштабування та вдосконалення системи. Ключові компоненти охоплюють:

- **Модуль доступу до API:**
  - **Ініціалізація клієнта Binance API:** Забезпечує встановлення з'єднання з API за допомогою наданих користувачем ключів.
  - **Виконання запитів до API:** Включає функції для отримання даних про ринок та виконання торгових операцій.
- **Модуль обробки даних:**
  - **Обробка та аналіз даних:** Отримує дані з API, аналізує їх та готує до подальшого використання в логіці торгівлі.
  - **Регулювання кількості активів:** Округлення кількості активів відповідно до правил біржі.
- **Модуль взаємодії з користувачем:**
  - **Збір інформації від користувача:** Забезпечує введення користувачем даних про цільові ціни, кількості для купівлі/продажу та інші параметри.
  - **Вивід результатів операцій та статусу:** Інформує користувача про результати виконаних операцій та поточний стан ринку.
- **Модуль логіки торгівлі:**
  - **Прийняття рішень на основі аналізу ринкових даних:** Включає алгоритми прийняття рішень на основі поточних ринкових умов.
  - **Виконання торгівлі відповідно до встановлених користувачем умов:** Реалізує торгові операції згідно з встановленими користувачем умовами.

3.3. Інтеграція з Binance API

Для інтеграції з Binance API використовується офіційна бібліотека ***python-binance***. Основні кроки включають:

- **Ініціалізація клієнта** (рис.3.1).

```
from binance.client import Client

api_key = 'your_api_key'
api_secret = 'your_api_secret'
client = Client(api_key, api_secret)
```

Рисунок 3.1– Створення клієнта для роботи з Binance API.

Цей рядок імпортує клас ***Client*** (рис.3.2) з бібліотеки ***binance.client***. Бібліотека ***python-binance*** надає офіційний інтерфейс для роботи з API біржі Binance, дозволяючи отримувати інформацію про ринок, виконувати торгові операції та управляти акаунтом [13, 4, 1].

```
from binance.client import Client
```

Рисунок 3.2– Імпорт бібліотеки.

Ці рядки визначають змінні ***api_key*** та ***api_secret*** (рис.3.3), які містять API ключ та секретний ключ, відповідно. Ці ключі використовуються для аутентифікації запитів до Binance API.

```
api_key = 'your_api_key'
api_secret = 'your_api_secret'
```

Рисунок 3.3– Оголошення змінних для API ключів.

Цей рядок створює екземпляр класу ***Client***, передаючи йому API ключ та секретний ключ (рис.3.4). Цей екземпляр (***client***) використовується для здійснення всіх наступних запитів до Binance API.

```
client = Client(api_key, api_secret)
```

Рисунок 3.4– Налаштування клієнта для подальшої взаємодії з Binance API

Таким чином, цей код налаштовує клієнта для подальшої взаємодії з Binance API. Після виконання цього коду, об'єкт client використовується для виконання різних операцій, таких як отримання інформації про ринок, баланс акаунту, здійснення купівлі або продажу криптовалют, та інших дій, підтримуваних API Binance.

- **Отримання даних про ціну активу:**

Ця функція (рис.3.5) призначена для отримання поточної ціни заданого торгового символу (наприклад, BTCUSDT) з біржі Binance за допомогою API Binance.

```
def get_price(symbol):
    ticker = client.get_symbol_ticker(symbol=symbol)
    return float(ticker['price'])
```

Рисунок 3.5– Отримання даних про ціну активу

Функція ***get_price*** (рис.3.6) приймає один аргумент ***symbol***, який є рядком і представляє торговий символ (наприклад, "BTCUSDT").

```
def get_price(symbol):
```

Рисунок 3.6– Оголошення функції *get_price()*

Виклик методу ***get_symbol_ticker*** об'єкта ***client***, який встановлює з'єднання з Binance API. Цей метод приймає параметр ***symbol***, що визначає торговий символ. Результатом виклику є словник ***ticker***, який містить інформацію про поточний стан ринку (рис.3.7) для заданого символу. Зокрема, цей словник включає поточну ціну активу.

```
    ticker = client.get_symbol_ticker(symbol=symbol)
```

Рисунок 3.7– Отримання поточного *ticker* для символу

Зі словника ***ticker*** витягується значення ключа ***'price'***, яке є рядком, що представляє поточну ціну активу. Це значення конвертується у число з плаваючою комою (тип float) і повертається як результат виконання функції.

Ця функція дозволяє легко отримувати поточну ціну активу для заданого символу, що є критично важливим для прийняття торгових рішень в реальному часі.

- **Отримання балансу активу:**

Функція ***get_balance(asset)*** (рис.3.8) призначена для отримання балансу конкретного активу на акаунті Binance. Вона використовує Binance API для отримання цієї інформації.

```
def get_balance(asset):
    balance = client.get_asset_balance(asset=asset)
    return float(balance['free'])
```

Рисунок 3.8– Функція для отримання балансу акутиву

Функція приймає один аргумент ***asset***, який є рядком і представляє символ активу (наприклад, "BTC" для біткоїна або "USDT" для Tether).

```
def get_balance(asset):
```

Рисунок 3.9– Оголошення функції *get_balance()*

Викликається метод ***get_asset_balance*** об'єкта ***client***, який встановлює з'єднання з Binance API. Цей метод приймає параметр ***asset*** (рис.3.10), що визначає символ активу. Результатом виклику є словник ***balance***, який містить інформацію про баланс активу на вашому акаунті.

```
    balance = client.get_asset_balance(asset=asset)
```

Рисунок 3.10– Отримання балансу активу

Зі словника ***balance*** витягується значення ключа ***'free'***, яке представляє доступний для торгівлі баланс активу. Це значення конвертується у число з плаваючою комою (тип float) і повертається як результат виконання функції.

Ця функція дозволяє отримувати доступний для торгівлі баланс конкретного активу на акаунті.

- **Виконання торгових операцій:**
    - **Продаж активу:**

Ця функція призначена для виконання операції продажу заданої кількості активу на біржі Binance за поточною ринковою ціною. Вона використовує Binance API для здійснення цієї операції і обробляє можливі помилки.

- Функція ***sell_asset*** приймає два аргументи: ***symbol*** і ***quantity***.
    - ***symbol*** (типу str): торговий символ активу (наприклад, "BTCUSDT").
    - ***quantity*** (типу float): кількість активу, яку необхідно продати.

У блоці ***try*** викликається метод ***order_market_sell*** об'єкта ***client***, який встановлює з'єднання з Binance API. Цей метод зображено на рисунку 3.11:

```
try:
    return client.order_market_sell(symbol=symbol, quantity=quantity)
```

Рисунок 3.11– Спроба виконання ринкового ордера на продаж

- Приймає два параметри: ***symbol*** і ***quantity***.
- Виконує операцію продажу зазначеної кількості активу за поточною ринковою ціною.
- Якщо операція успішна, метод повертає відповідь API, яка містить інформацію про виконаний ордер.

Якщо під час виконання ринкового ордера виникає помилка, вона обробляється у блоці ***except*** (рис.3.12).

```
except Exception as e:
    return str(e)
```

Рисунок 3.12– Обробка винятків

- Помилка зберігається у змінній ***e***.
- Рядкове представлення помилки повертається як результат функції.
    - **Купівля активу:**

Ця функція призначена для розміщення лімітного замовлення на купівлю активу на біржі Binance за допомогою API Binance. Лімітне замовлення виконується лише за вказаною або нижчою ціною (рис.3.13).

```
def buy_asset_limit(symbol, quantity, price):
    return client.order_limit_buy(symbol=symbol, quantity=quantity, price=f'{price:.8f}')
```

Рисунок 3.13– Функція для розміщення лімітного замовлення на купівлю

Функція ***buy_asset_limit*** оголошується з трьома параметрами:

- ***symbol***: рядок, що представляє торговий символ.
- ***quantity***: кількість активу, яку необхідно купити.
- ***price***: ціна, за якою потрібно розмістити лімітне замовлення.

Викликається метод ***order_limit_buy*** об'єкта ***client***, який встановлює з'єднання з Binance API. Цей метод розміщує лімітне замовлення на купівлю активу за вказаними параметрами:

- ***symbol***: торговий символ.
- ***quantity***: кількість активу для купівлі.
- ***price***: ціна, за якою має бути виконане замовлення. Використовується форматування рядка ***f'{price:.8f}'*** для забезпечення точності ціни до 8 знаків після коми.

Форматований рядок ***f'{price:.8f}'*** використовується, щоб представити ціну у форматі з 8 знаками після коми. Це гарантує, що ціна буде передана в правильному форматі, який підтримується Binance API, що є важливим для точності фінансових операцій.

- **Отримання інформації про лот:**

Ця функція призначена для отримання інформації про лот (крок кількості, мінімальну та максимальну кількість активу для торгів) для заданого торгового символу з біржі Binance (рис.3.14) за допомогою API Binance.

```
def get_lot_size(symbol):
    info = client.get_symbol_info(symbol)
    if info is not None:
        for filt in info['filters']:
            if filt['filterType'] == 'LOT_SIZE':
                return {
                    'stepSize': float(filt['stepSize']),
                    'minQty': float(filt['minQty']),
                    'maxQty': float(filt['maxQty'])
                }
    return None
```

Рисунок 3.14– Функція для отримання інформації про лот

Функція ***get_lot_size*** приймає один аргумент ***symbol***, який є рядком і представляє торговий символ.

Викликається метод ***get_symbol_info*** об'єкта ***client***, який встановлює з'єднання з Binance API. Цей метод приймає параметр ***symbol***, що визначає торговий символ. Результатом виклику є словник ***info***, який містить детальну інформацію про заданий символ. Якщо символ не знайдено або виникла помилка, ***info*** буде дорівнювати ***None***.

Далі перевіряється, чи результат виклику ***get_symbol_info*** не є ***None***. Якщо ***info*** дорівнює ***None***, це означає, що інформацію про символ не вдалося отримати, і функція перейде до завершення з поверненням ***None***.

Якщо інформація про символ отримана успішно, виконується перебір списку фільтрів, що міститься в словнику ***info*** під ключем ***'filters'***. Кожен фільтр представляє собою словник з параметрами торгівлі для цього символу.

Для кожного фільтра перевіряється значення ключа ***'filterType'***. Якщо цей тип дорівнює ***'LOT_SIZE'***, значить, знайдено потрібний фільтр, який містить інформацію про лот для цього символу.

Якщо знайдено фільтр типу ***'LOT_SIZE'***, створюється і повертається словник, що містить три ключі: ***'stepSize'*** (крок кількості), ***'minQty'*** (мінімальна кількість) та ***'maxQty'*** (максимальна кількість). Значення цих ключів витягуються з відповідних полів фільтра і конвертуються у числа з плаваючою комою (тип float).

Якщо серед фільтрів символу не знайдено фільтра типу ***'LOT_SIZE'***, функція повертає ***None***, що свідчить про неможливість отримання інформації про лот для цього символу.

- **Регулювання кількості активу:**

```
def adjust_quantity(quantity, step_size):
    return round((quantity // step_size) * step_size, 8)
```

Рисунок 3.15– Функція для регулювання кількості активу

Функція ***adjust_quantity(quantity, step_size)*** (рис.3.15) призначена для округлення кількості активу до найближчого допустимого значення, яке відповідає кроку торгів (step size) на біржі. Це необхідно для забезпечення відповідності торгових заявок правилам біржі щодо мінімальної і максимальної кількості активів, які можна торгувати.

Функція ***adjust_quantity*** приймає два аргументи:

- ***quantity:*** кількість активу, яку потрібно скоригувати.
- ***step_size:*** крок торгів, який визначає мінімальну зміну кількості активу.

Далі відбувається обчислення нової кількості, яка відповідає допустимому значенню, за такими підкроками:

1. Виконується ділення ***quantity*** на ***step_size*** з відкиданням дробової частини (цілочисельне ділення). Це дозволяє отримати найбільше ціле число разів, яке ***step_size*** вміщується у ***quantity***.
2. Цей результат множиться на ***step_size***, щоб отримати найближче допустиме значення кількості, кратне ***step_size***.
3. Отримане значення округлюється до 8 знаків після коми. Це необхідно для забезпечення точності і відповідності формату кількості активу, який приймає біржа.

Скориговане значення кількості активу повертається як результат виконання функції.

Приклад роботи функції:

Припустимо, що ***quantity*** дорівнює 0.123456789 і ***step_size*** дорівнює 0.01. Використовуючи функцію ***adjust_quantity***, отримуємо:

1. Цілочисельне ділення: 0.123456789 // 0.01 = 12.0
2. Множення: 12.0 * 0.01 = 0.12
3. Округлення: round(0.12, 8) = 0.12

Отже, функція поверне значення 0.12.

Ця функція гарантує, що кількість активу, яка використовується у торгових заявках, завжди відповідає вимогам біржі, забезпечуючи тим самим коректність і успішне виконання торгових операцій.

3.4. Програмні інтерфейси користувача

Програмні інтерфейси користувача реалізовані у вигляді консольних запитів, які забезпечують взаємодію з користувачем під час виконання програми. Основні функції включають:

1) **Запит цільових цін для продажу:**

```
def request_target_prices():
    lower_price = float(input("Введіть нижню межу ціни для продажу: "))
    upper_price = float(input("Введіть верхню межу ціни для продажу: "))
    return lower_price, upper_price
```

Рисунок 3.16– Функція для запиту у користувача межі цін.

Функція ***request_target_prices()*** (рис.3.16) призначена для запиту у користувача меж цін, за якими він хоче здійснити продаж активів. Вона отримує нижню та верхню межі цін, конвертує їх у формат числа з плаваючою комою ***(float)*** і повертає їх у вигляді кортежу. Розглянемо кожен крок цієї функції детально:

Оголошення функції ***request_target_prices***, яка не приймає жодних аргументів.

Запит нижньої межі ціни у користувача:

- Викликається функція ***input()*** з текстом запиту "Введіть нижню межу ціни для продажу: ".
- Функція ***input()*** зупиняє виконання програми і чекає, поки користувач введе значення і натисне клавішу Enter.
- Введене користувачем значення, яке є рядком, конвертується у тип ***float*** за допомогою функції ***float()***. Це потрібно для того, щоб працювати з числовими значеннями, а не рядками.
- Конвертоване значення присвоюється змінній ***lower_price***.

Запит верхньої межі ціни у користувача:

- Аналогічно, викликається функція ***input()*** з текстом запиту "Введіть верхню межу ціни для продажу: ".
- Програма чекає введення значення від користувача і після натискання Enter конвертує введений рядок у тип ***float***.
- Конвертоване значення присвоюється змінній ***upper_price***.

Повернення значень:

- Функція повертає кортеж, який містить дві змінні: ***lower_price*** та ***upper_price***.
- Це дозволяє використовувати обидва значення в інших частинах програми, зокрема для прийняття рішень щодо продажу активів на основі заданих користувачем меж(рис.3.17) .

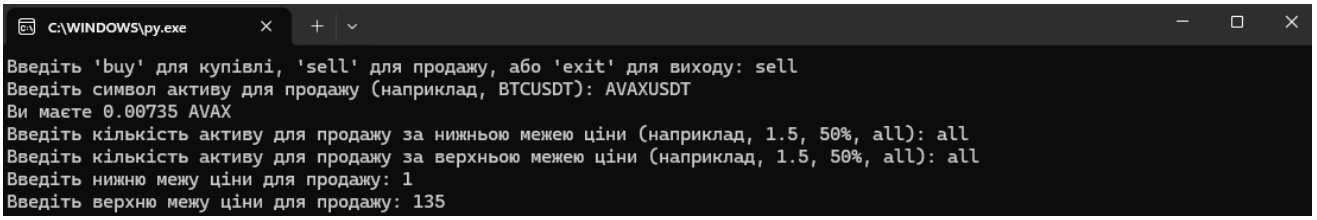

Рисунок 3.17– Приклад користування ботом.

2) Запит інформації для покупки активу:

Функція ***request_purchase_info()*** (рис.3.18) призначена для отримання інформації від користувача щодо символу активу, суми для покупки та ціни покупки [12, 1, 11, 6]. Ця інформація використовується для розміщення ордеру на купівлю активу на біржі Binance. Отже розберемо детально код:

```python
def request_purchase_info():
    while True:
        asset_symbol = input("Введіть символ активу для купівлі (наприклад, BTCUSDT): ")
        if client.get_symbol_info(asset_symbol) is not None:
            break
        print("Актив не знайдено на Binance. Будь ласка, спробуйте інший символ.")

    usdt_balance = get_balance('USDT')
    print(f"Ваш доступний баланс: {usdt_balance} USDT")
    while True:
        input_type = input("Введіть суму в USDT, відсоток від балансу або 'all' для використання всього балансу (наприклад, 100, 50%, all): ").strip().lower()
        if input_type.endswith('%'):
            try:
                percentage = float(input_type[:-1]) / 100
                amount_usdt = usdt_balance * percentage
                if amount_usdt <= usdt_balance:
                    break
                print("Введений відсоток перевищує доступний баланс.")
            except ValueError:
                print("Неправильний відсоток. Будь ласка, спробуйте знову.")
        elif input_type == 'all':
            amount_usdt = usdt_balance
            break
        else:
            try:
                amount_usdt = float(input_type)
                if amount_usdt <= usdt_balance:
                    break
                else:
                    print("Недостатньо коштів на вашому гаманці. Будь ласка, введіть меншу суму.")
            except ValueError:
                print("Неправильний формат суми. Будь ласка, введіть число, відсоток або 'all'.")

    buy_price = float(input("Введіть ціну, за якою бажаєте купити актив: "))
    return asset_symbol, amount_usdt, buy_price
```

Рисунок 3.18– Функція *request_purchase_info()*.

Функція не приймає жодних аргументів і повертає три значення: символ активу, суму в USDT та ціну покупки.

Введення символу активу:

1. **Запуск циклу:**
   - Функція починає з безкінечного циклу ***while True:***, який продовжуватиметься, поки не буде введено дійсний символ активу.
2. **Введення символу активу:**
   - Користувач вводить символ активу, за допомогою функції ***input()***.
3. **Перевірка символу активу:**
   - Функція перевіряє, чи існує актив на Binance за допомогою методу ***client.get_symbol_info(asset_symbol)***. *Я*кщо актив існує, метод повертає інформацію про актив, і цикл завершується за допомогою ***break***.
4. **Повідомлення про помилку:**
   - Якщо актив не знайдено, виводиться повідомлення "Актив не знайдено на Binance. Будь ласка, спробуйте інший символ." і користувачеві пропонується ввести символ знову.

Отримання балансу в USDT:

1. **Отримання балансу:**
    - Викликається функція ***get_balance('USDT')***, яка повертає баланс USDT на рахунку користувача.
2. **Вивід балансу:**
    - Баланс виводиться на екран за допомогою ***print()***, щоб користувач знав свій доступний баланс (рис.3.19).

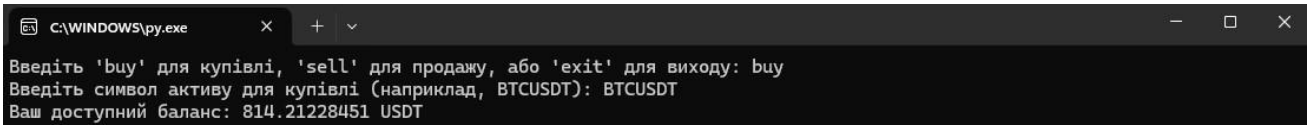

Рисунок 3.19– Приклад виводу балансу.

Введення суми для покупки:

1. **Запуск циклу:**
    - Починається новий безкінечний цикл для введення суми покупки.
2. **Введення суми:**
    - Користувач вводить суму в USDT, відсоток від балансу або 'all' для використання всього балансу.

3. **Обробка відсотку:**
    - Якщо введене значення закінчується на *'%'*, програма намагається перетворити введене значення у відсоток і обчислити відповідну суму в USDT.
    - Якщо введений відсоток перевищує доступний баланс, виводиться повідомлення про помилку.
4. **Обробка значення 'all':**
    - Якщо введено '**all**', вся доступна сума USDT використовується для покупки.
5. **Обробка числового значення:**
    - Якщо введене значення є числом, воно перетворюється на тип ***float*** і перевіряється, чи не перевищує воно доступний баланс. У випадку перевищення виводиться повідомлення про помилку.
6. **Повідомлення про помилку формату:**
    - Якщо введене значення не є числом, відсотком або 'all', виводиться повідомлення про неправильний формат суми (рис.3.20).

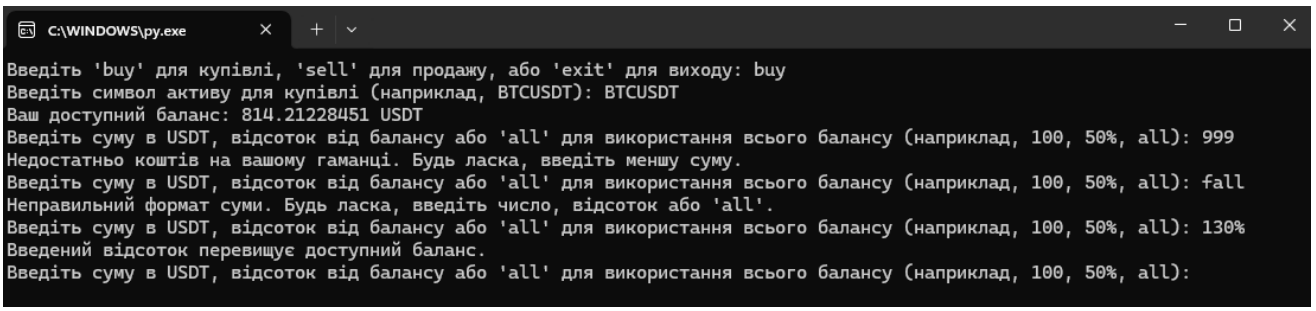

Рисунок 3.20– Відображення помилок при некоректному введенні.

Введення ціни покупки

1. **Введення ціни:**
    - Користувач вводить ціну покупки активу, яка перетворюється на тип ***float***.

Введення ціни покупки

1. **Повернення значень:**
    - Функція повертає три значення: символ активу, суму в USDT та ціну покупки, які будуть використані для розміщення ордеру на купівлю активу.

Отже ця функція дозволяє користувачу вводити необхідні дані для купівлі активу, перевіряючи правильність введених даних та доступність активу на біржі.

3) Запит інформації для покупки активу:

Функція ***request_sell_info()*** запитує у користувача інформацію, необхідну для продажу активу на біржі Binance, і повертає цю інформацію у вигляді декількох значень. Розглянемо цей код покроково:

a) Запит символу активу для продажу:

```
symbol = input("Введіть символ активу для продажу (наприклад, BTCUSDT): ")
```

Рисунок 3.21– Код для введення символу активу для продажу.

- Функція починається із запиту у користувача символу активу (рис.3.21), який він хоче продати. Наприклад, символ "BTCUSDT" представляє торгівлю між Bitcoin (BTC) та Tether (USDT).

b) Витяг назви активу з символу:

```
asset = symbol.replace('USDT', '')
```

Рисунок 3.22– Код для отримання назви активу шляхом видалення суфікса 'USDT' з символу активу.

- З введеного символу видаляється рядок "USDT" (рис.3.22), щоб отримати назву активу (наприклад, "BTC").

c) Отримання балансу активу:

```
asset_balance = get_balance(asset)
print(f"Ви маєте {asset_balance} {asset}")
```

Рисунок 3.23– Код для отримання балансу активу та виведення його на екран.

- Викликається функція ***get_balance***, яка повертає баланс користувача для даного активу (рис.3.23). Отримане значення виводиться на екран для інформування користувача про наявний баланс.

d) Визначення вкладеної функції *get_quantity*:

```
def get_quantity(prompt):
    while True:
        input_type = input(prompt).strip().lower()
        if input_type.endswith('%'):
            percentage = float(input_type[:-1]) / 100
            quantity = asset_balance * percentage
        elif input_type == 'all':
            quantity = asset_balance
        else:
            try:
                quantity = float(input_type)
            except ValueError:
                print("Введіть правильну кількість.")
                continue
        if quantity > asset_balance:
            print("Введена кількість перевищує ваш баланс.")
            continue
        return quantity
```

Рисунок 3.24– Функція для отримання кількості активу з урахуванням відсотків, значення 'all' або конкретного числа, з перевіркою на перевищення балансу

- Оголошується внутрішня функція ***get_quantity***, яка запитує у користувача кількість активу для продажу (рис.3.24). Вона може приймати кількість у вигляді абсолютного значення, відсотка від балансу або всього балансу (команда ***"all"***).

e) Запит кількості активу для продажу за нижньою межею ціни:

```
lower_quantity = get_quantity("Введіть кількість активу для продажу за нижньою межею ціни (наприклад, 1.5, 50%, all): ")
```

Рисунок 3.25. Виклик функції для отримання кількості активу для продажу за нижньою межею ціни.

- Викликається функція ***get_quantity*** для отримання кількості активу (рис.3.25), який користувач бажає продати за нижньою межею ціни.

f) Запит кількості активу для продажу за верхньою межею ціни:

```
upper_quantity = get_quantity("Введіть кількість активу для продажу за верхньою межею ціни (наприклад, 1.5, 50%, all): ")
```

Рисунок 3.26– Виклик функції для отримання кількості активу для продажу за верхньою межею ціни.

- Викликається функція ***get_quantity*** для отримання кількості активу (рис.3.26), який користувач бажає продати за верхньою межею ціни.

g) Запит цінових меж для продажу:

```
lower_price, upper_price = request_target_prices()
```

Рисунок 3.27– Виклик функції для отримання цільових цін нижньої та верхньої меж

- Викликається функція ***request_target_prices***, яка запитує у користувача нижню та верхню межі ціни для продажу активу (рис.3.27).

h) Отримання інформації про розмір лоту:

```
lot_size_info = get_lot_size(symbol)
if lot_size_info is None:
    print("Не вдалося отримати інформацію про лот для даного символу.")
    return None, None, None, None, None
```

Рисунок 3.28– Отримання інформації про лот і перевірка її наявності для заданого символу

- Викликається функція ***get_lot_size*** для отримання інформації про крок кількості, мінімальну та максимальну кількість активу для торгів (рис.3.28). Якщо інформацію не вдалося отримати, виводиться повідомлення про помилку і функція повертає значення ***None***.

i) Округлення кількості активу до дозволеного розміру лоту:

```
lower_quantity = adjust_quantity(lower_quantity, lot_size_info['stepSize'])
upper_quantity = adjust_quantity(upper_quantity, lot_size_info['stepSize'])
```

Рисунок 3.29– Коригування кількості активу відповідно до розміру кроку лота для нижньої та верхньої межі ціни

- Викликається функція ***adjust_quantity*** для округлення кількості активу до найближчого дозволеного розміру лоту згідно з інформацією, отриманою у попередньому кроці (рис.3.29).

j) Округлення кількості активу до дозволеного розміру лоту:

```
return symbol, asset, lower_quantity, upper_quantity, lower_price, upper_price
```

Рисунок 3.30– Повернення символу активу, назви активу, кількості для нижньої та верхньої межі, а також цільових цін

- Функція повертає зібрану інформацію: символ активу, назву активу, кількість активу для продажу за нижньою та верхньою межею ціни, а також самі ціни (рис.3.30).

```
def request_sell_info():
    symbol = input("Введіть символ активу для продажу (наприклад, BTCUSDT): ")
    asset = symbol.replace('USDT', '')
    asset_balance = get_balance(asset)
    print(f"Ви маєте {asset_balance} {asset}")

    def get_quantity(prompt):
        while True:
            input_type = input(prompt).strip().lower()
            if input_type.endswith('%'):
                percentage = float(input_type[:-1]) / 100
                quantity = asset_balance * percentage
            elif input_type == 'all':
                quantity = asset_balance
            else:
                try:
                    quantity = float(input_type)
                except ValueError:
                    print("Введіть правильну кількість.")
                    continue
            if quantity > asset_balance:
                print("Введена кількість перевищує ваш баланс.")
                continue
            return quantity

    lower_quantity = get_quantity("Введіть кількість активу для продажу за нижньою межею ціни (наприклад, 1.5, 50%, all): ")
    upper_quantity = get_quantity("Введіть кількість активу для продажу за верхньою межею ціни (наприклад, 1.5, 50%, all): ")

    lower_price, upper_price = request_target_prices()

    lot_size_info = get_lot_size(symbol)
    if lot_size_info is None:
        print("Не вдалося отримати інформацію про лот для даного символу.")
        return None, None, None, None, None

    lower_quantity = adjust_quantity(lower_quantity, lot_size_info['stepSize'])
    upper_quantity = adjust_quantity(upper_quantity, lot_size_info['stepSize'])

    return symbol, asset, lower_quantity, upper_quantity, lower_price, upper_price
```

Рисунок 3.31– Функція request_sell_info()

Ця функція (рис.3.31) дозволяє користувачу вводити всі необхідні параметри для виконання продажу активу, включаючи символ активу, кількість для продажу, а також цінові межі, і повертає ці дані для подальшої обробки у програмі [7, 8, 5].

4) Основна торгова логіка:

Функція ***trading_logic*** реалізує основну торгову логіку, що дозволяє користувачеві взаємодіяти з торговим ботом через консоль, виконуючи операції купівлі та продажу активів на основі поточних ринкових умов [10, 12]. Розглянемо кожен крок цієї функції детальніше.

a) Оголошення функції:

Функція ***trading_logic*** не приймає жодних параметрів і є основним циклом взаємодії з користувачем.

b) Вічний цикл:

Функція використовує безкінечний цикл ***while True***, щоб постійно приймати вхідні команди від користувача, поки не буде введена команда виходу.

c) Прийом команди користувача:

Функція чекає на введення команди користувачем і зберігає її у змінну ***action***. Введене значення перетворюється на нижній регістр для полегшення обробки (рис.3.32).

```
action = input("Введіть 'buy' для купівлі, 'sell' для продажу, або 'exit' для виходу: ").lower()
```

Рисунок 3.32– Код для введення дії користувача: купівля, продаж або вихід з програми

d) Обробка команди 'buy':

```
if action == 'buy':
    try:
        symbol, amount, buy_price = request_purchase_info()
        lot_size_info = get_lot_size(symbol)
        quantity = adjust_quantity(amount / buy_price, lot_size_info['stepSize'])
        result = buy_asset_limit(symbol, quantity, buy_price)
        print("Результат купівлі:", result)
    except Exception as e:
        print(f"Сталася помилка при купівлі: {e}")
```

Рисунок 3.33– Код для виконання купівлі активу за лімітною ціною з обробкою виключень

Цей крок виконує операцію купівлі фінансового активу. Операція починається при виборі дії ***'buy'*** і включає кілька основних кроків: отримання інформації про купівлю, коригування кількості активу відповідно до розміру лота, спробу купівлі активу за лімітною ціною та обробку можливих помилок (рис.3.33).

4.1 Перевірка дії

Код виконується тільки тоді, коли змінна ***action*** має значення ***'buy'***. Це означає, що операція купівлі починається лише при відповідному значенні цієї змінної.

4.2 Використання блоку try-except для обробки винятків

Використання блоку ***try*** дозволяє обробляти помилки, які можуть виникнути під час виконання операції купівлі. Якщо помилка виникає в будь-якій частині блоку ***try***, вона перехоплюється блоком ***except***.

4.3 Отримання інформації про купівлю

Виклик функції ***request_purchase_info()*** для отримання інформації про купівлю, яка повертає три значення: ***symbol*** (символ фінансового активу), ***amount*** (сума грошей для купівлі) та ***buy_price*** (ціна, за якою планується купівля).

4.4 Отримання інформації про розмір лота

Виклик функції ***get_lot_size(symbol)***, яка отримує інформацію про розмір лота для вказаного символу активу. Ця інформація включає ***stepSize*** — мінімальний крок зміни кількості активу, який можна купити.

4.5 Коригування кількості активу

Розрахунок кількості активу, яку можна купити за вказану суму грошей за ціною купівлі ***(amount / buy_price)***, з подальшим коригуванням цієї кількості відповідно до мінімального кроку зміни кількості ***(lot_size_info['stepSize'])***. Функція ***adjust_quantity*** гарантує, що кількість активу буде відповідати вимогам Binance API.

4.6 Спроба купівлі активу за лімітною ціною

Виклик функції ***buy_asset_limit(symbol, quantity, buy_price)***, яка виконує лімітний ордер на купівлю вказаного символу активу ***(symbol)*** в зазначеній кількості ***(quantity)*** за вказаною ціною ***(buy_price)***. Результат цієї операції зберігається в змінній ***result***.

4.7 Вивід результату купівлі

Виведення результату операції купівлі на екран. Це дозволяє користувачеві бачити, що купівля була успішною або дізнатися додаткову інформацію про результат операції.

4.8 Обробка можливих помилок

Блок ***except*** перехоплює будь-які виключення, що можуть виникнути в блоці ***try***. Якщо виникає помилка, її повідомлення виводиться на екран за допомогою функції ***print***, що дозволяє діагностувати проблеми, які виникли під час спроби купівлі.

Цей фрагмент коду демонструє операцію купівлі активів з використанням лімітного ордера, включаючи перевірку та обробку можливих помилок. Кожен крок є важливим для забезпечення коректного виконання операції на платформі торгівлі, забезпечуючи точність та відповідність вимогам Binance API.

е) Обробка команди 'sell':

Цей код виконує операцію продажу фінансового активу. Операція починається при виборі дії ***'sell'*** і включає кілька основних кроків: отримання інформації про продаж, перевірка поточної ціни активу, виконання продажу при досягненні певних цінових рівнів та обробка можливих помилок(рис.3.34) .

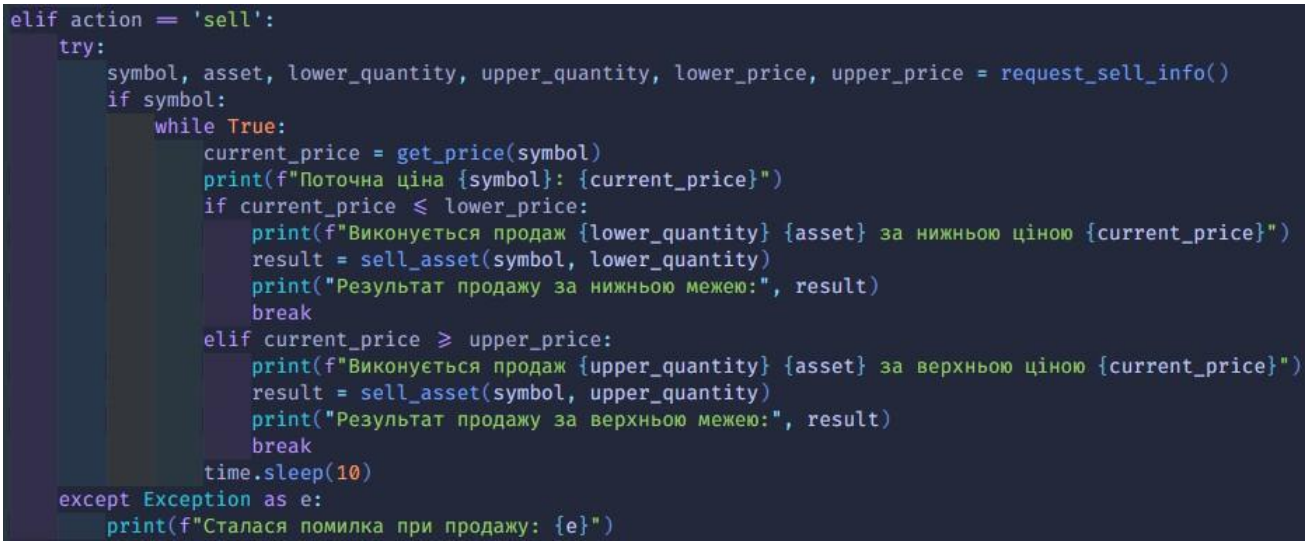

Рисунок 3.34– Код для виконання продажу активу при досягненні цільових цін з обробкою виключень

e.1. Перевірка дії

Код виконується тільки тоді, коли змінна ***action*** має значення ***'sell'***. Це означає, що операція продажу починається лише при відповідному значенні цієї змінної.

e.2. Використання блоку try-except для обробки винятків

Використання блоку ***try*** дозволяє обробляти помилки, які можуть виникнути під час виконання операції продажу. Якщо помилка виникає в будь-якій частині блоку ***try***, вона перехоплюється блоком ***except***.

e.3. Отримання інформації про продаж

Виклик функції ***request_sell_info()*** для отримання інформації про продаж, яка повертає кілька значень: ***symbol*** (символ фінансового активу), ***asset*** (назва активу), ***lower_quantity*** (кількість активу для продажу за нижньою ціною), ***upper_quantity*** (кількість активу для продажу за верхньою ціною), ***lower_price*** (нижня межа ціни) та ***upper_price*** (верхня межа ціни).

e.4. Перевірка наявності символу

Перевірка, чи символ активу не є порожнім. Якщо символ активу вказаний, продовжується виконання операції.

e.5. Цикл перевірки поточної ціни та виконання продажу

Безкінечний цикл, який продовжує виконуватися до тих пір, поки не буде виконано умови для продажу.

е.6. Отримання поточної ціни активу

Виклик функції ***get_price(symbol)***, яка отримує поточну ціну вказаного символу активу. Ця ціна виводиться на екран для інформування користувача.

е.7. Перевірка умови для продажу за нижньою ціною

Якщо поточна ціна менша або дорівнює нижній межі ціни ***(lower_price)***, виконується продаж активу в кількості ***lower_quantity***. Результат продажу виводиться на екран, і цикл переривається командою ***break***.

е.8. Перевірка умови для продажу за верхньою ціною

Якщо поточна ціна більша або дорівнює верхній межі ціни ***(upper_price)***, виконується продаж активу в кількості ***upper_quantity***. Результат продажу виводиться на екран, і цикл переривається командою ***break***.

е.9. Очікування перед наступною перевіркою ціни

Виклик функції ***time.sleep(10)***, яка затримує виконання наступної ітерації циклу на 10 секунд. Це дозволяє уникнути надмірного навантаження на систему шляхом постійних перевірок ціни.

е.10. Обробка можливих помилок

Блок ***except*** перехоплює будь-які виключення, що можуть виникнути в блоці ***try***. Якщо виникає помилка, її повідомлення виводиться на екран за допомогою функції ***print***, що дозволяє діагностувати проблеми, які виникли під час спроби продажу.

Код демонструє операцію продажу активів з використанням визначених умов цінових рівнів, включаючи перевірку та обробку можливих помилок. Кожен крок є важливим для забезпечення коректного виконання операції на платформі торгівлі, забезпечуючи точність та своєчасність виконання умов продажу.

3.5. Алгоритми та логіка торгівлі

Алгоритми та логіка торгівлі забезпечують автоматичне виконання операцій відповідно до встановлених користувачем параметрів [9, 12]. Основні аспекти включають:

1) **Моніторинг ринкових цін:**
   - 1.1. Регулярне отримання поточних цін активів за допомогою функції ***get_price***.
   - 1.2. Вивід поточних цін на екран для інформування користувача.
2) **Виконання умовних торгових операцій:**
   - 2.1. Виконання продажу активу при досягненні нижньої або верхньої межі ціни.
   - 2.2. Виконання покупки активу при введенні відповідної команди користувачем.
3) **Регулювання кількості активів:**

3.1. Використання функції ***adjust_quantity*** для округлення кількості активів відповідно до правил біржі.

4) **Обробка помилок та виключень:**

4.1. Використання конструкцій ***try...except*** для обробки помилок, що виникають під час виконання торгових операцій.

4.2. Вивід повідомлень про помилки для інформування користувача та можливості виправлення введених даних.

3.6. Тестування та налагодження

Тестування та налагодження програмного забезпечення включають кілька етапів, кожен з яких спрямований на забезпечення коректної та надійної роботи бота.

1. **Тестування функцій:**

1.1. **Отримання даних про ціну та баланс:** Перевірка правильності отримання даних за допомогою функцій ***get_price*** та ***get_balance***.

1.2. **Виконання торгових операцій:** Перевірка коректності виконання функцій ***sell_asset*** та ***buy_asset_limit***.

2. **Симуляція торгівлі:**

2.1. **Запуск у тестовому середовищі Binance:** Виконання тестових операцій у середовищі Binance Testnet для перевірки роботи без ризику втрати реальних коштів.

2.2. **Реакція на зміни ринкових умов:** Перевірка адекватності реакції бота на зміни цін активів та виконання відповідних торгових операцій.

3. **Налагодження помилок:**
    3.1. **Обробка виключень:** Відлов помилок за допомогою конструкцій ***try...except*** та вивід інформативних повідомлень про помилки.
    3.2. **Виправлення логічних та синтаксичних помилок:** Виявлення та виправлення помилок у логіці та синтаксисі коду для забезпечення його коректної роботи.

Технічне забезпечення включає як апаратні, так і програмні компоненти, необхідні для стабільної та надійної роботи торгового бота.

1. **Апаратне забезпечення:**
    1.1. **Комп'ютер або сервер з доступом до Інтернету:** Для забезпечення постійного доступу до ринку Binance та виконання торгових операцій у режимі реального часу.
    1.2. **Мінімальні вимоги до обчислювальних ресурсів:**
        1.2.1. **Процесор:** Частота не менше 2.0 ГГц для забезпечення достатньої швидкості обробки даних.
        1.2.2. **Оперативна пам'ять:** Не менше 4 ГБ для забезпечення стабільної роботи програми та обробки великих обсягів даних.
2. **Програмне забезпечення:**
    2.1. **Операційна система:** Підтримка Windows, macOS або Linux для забезпечення сумісності з різними апаратними платформами.
    2.2. **Мова програмування та інтерпретатор:** Python 3.7 або новіше для забезпечення сумісності з використаними бібліотеками та модулями.
    2.3. **Бібліотеки та модулі:** Встановлення необхідних бібліотек, таких як ***python-binance*** та ***time***.
3. **Безпека та захист даних:**
    3.1. **Використання безпечних API ключів:** Зберігання API ключів у безпечному місці та уникнення їх публічного розголошення.

3.2. **Захист конфіденційних даних:** Впровадження механізмів шифрування та захисту даних для запобігання несанкціонованому доступу.

Регулярне оновлення програмного забезпечення: Здійснення регулярних оновлень для забезпечення актуальності та безпеки програмного забезпечення.

ВИСНОВКИ

У рамках дипломної роботи розроблено торгового бота для роботи з криптовалютною біржею Binance, який використовує бібліотеку ***binance.client.Client*** для взаємодії з біржею. Основною метою розробки є створення інструменту, здатного автоматизувати процеси купівлі та продажу криптовалют на основі ринкових умов, визначених користувачем.

Розробка цього бота дозволила глибше зрозуміти принципи роботи з API криптовалютних бірж, особливості торгівлі криптовалютами та методи автоматизації торгових процесів. Практичне застосування таких інструментів може значно підвищити ефективність торгівлі, зменшити ризики, пов'язані з людським фактором, та забезпечити своєчасне виконання торгових операцій.

Отже, створений бот є корисним інструментом для автоматизації торгових процесів на біржі Binance і може бути використаний як основа для подальшого розширення функціональності, наприклад, додавання алгоритмів технічного аналізу, інтеграції зі сторонніми сервісами для отримання новин та прогнозів, а також підвищення безпеки через використання більш складних механізмів автентифікації та шифрування даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Binance API Documentation, Binance. (n.d.). Binance API Documentation. [Електронний ресурс] – Режим доступу до ресурсу: https://binance-docs.github.io/apidocs/spot/en/ (дата звернення: 25.05.2024);
2. Python Binance Library, Sammchardy. (n.d.). python-binance library on GitHub. [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/ sammchardy/python-binance (дата звернення: 25.05.2024);
3. Python Documentation, Python Software Foundation. (n.d.). Python Documentation. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/ (дата звернення: 25.05.2024);
4. API Key and Secret Management, Binance Academy. (n.d.). What Is an API Key?. [Електронний ресурс] – Режим доступу до ресурсу: https://academy.binance.com/ en/articles/what-is-an-api-key (дата звернення: 25.05.2024);
5. Error Handling in Python, Python Software Foundation. (n.d.). Errors and Exceptions. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/ tutorial/errors.html (дата звернення: 25.05.2024);
6. Floating Point Arithmetic, Python Software Foundation. (n.d.). Floating Point Arithmetic: Issues and Limitations. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/tutorial/floatingpoint.html (дата звернення: 25.05.2024);
7. Input and Output Operations, Python Software Foundation. (n.d.). Input and Output. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/ tutorial/inputoutput.html (дата звернення: 25.05.2024);
8. Time Module, Python Software Foundation. (n.d.). time — Time Access and Conversions. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/library/time.html (дата звернення: 25.05.2024);
9. "Mastering Bitcoin: Programming the Open Blockchain" by Andreas M. Antonopoulos – O'Reilly Media, 2017.

10. "Cryptocurrency Trading: Complete Guide to Trading Altcoins" by Aimee Bateman – ClydeBank Media LLC, 2022.
11. "Python for Finance: Mastering Data-Driven Finance" by Yves Hilpisch – O'Reilly Media, 2018.
12. "Hands-On Cryptocurrency Trading: Learn the Fundamentals of Trading Cryptocurrencies and Building Automated Cryptocurrency Trading Bots" by Amin Shokri – Packt Publishing, 2022.
13. "Binance Cryptocurrency Exchange: An Insider's Guide to Trading Cryptocurrencies on Binance" by CryptoTrading Pro – CryptoTrading Pro LLC, 2021.
14. "Programming Fundamentals. Python. Part 1" by Yakovenko A. V. – 2018.
15. "Foundations of Python Network Programming: The Comprehensive Guide to Building Network Applications with Python", Second Edition by John Goerzen, Brandon Rhodes – Apress, 2010

ДОДАТКИ

ДОАТОК А

```
def get_lot_size(symbol):
    info = client.get_symbol_info(symbol)
    if info is not None:
        for filt in info['filters']:
            if filt['filterType'] == 'LOT_SIZE':
                return {
                    'stepSize': float(filt['stepSize']),
                    'minQty': float(filt['minQty']),
                    'maxQty': float(filt['maxQty'])
                }
    return None


def request_purchase_info():
    while True:
        asset_symbol = input("Введіть символ активу для купівлі (наприклад, BTCUSDT): ")
        if client.get_symbol_info(asset_symbol) is not None:
            break
        print("Актив не знайдено на Binance. Будь ласка, спробуйте інший символ.")


    usdt_balance = get_balance('USDT')
    print(f"Ваш доступний баланс: {usdt_balance} USDT")
    while True:
            input_type = input("Введіть суму в USDT, відсоток від балансу або 'all' для використання всього балансу (наприклад, 100, 50%, all): ").strip().lower()
```

```
    if input_type.endswith('%'):
        try:
            percentage = float(input_type[:-1]) / 100
            amount_usdt = usdt_balance * percentage
            if amount_usdt <= usdt_balance:
                break
            print("Введений відсоток перевищує доступний баланс.")
        except ValueError:
            print("Неправильний відсоток. Будь ласка, спробуйте знову.")
    elif input_type == 'all':
        amount_usdt = usdt_balance
        break
    else:
        try:
            amount_usdt = float(input_type)
            if amount_usdt <= usdt_balance:
                break
            else:
                print("Недостатньо коштів на вашому гаманці. Будь ласка, введіть меншу суму.")
        except ValueError:
            print("Неправильний формат суми. Будь ласка, введіть число, відсоток або 'all'.")

buy_price = float(input("Введіть ціну, за якою бажаєте купити актив: "))
return asset_symbol, amount_usdt, buy_price
```

```
def request_sell_info():
    symbol = input("Введіть символ активу для продажу (наприклад, BTCUSDT): ")
    asset = symbol.replace('USDT', '')
    asset_balance = get_balance(asset)
    print(f"Ви маєте {asset_balance} {asset}")

    def get_quantity(prompt):
        while True:
            input_type = input(prompt).strip().lower()
            if input_type.endswith('%'):
                percentage = float(input_type[:-1]) / 100
                quantity = asset_balance * percentage
            elif input_type == 'all':
                quantity = asset_balance
            else:
                try:
                    quantity = float(input_type)
                except ValueError:
                    print("Введіть правильну кількість.")
                    continue
            if quantity > asset_balance:
                print("Введена кількість перевищує ваш баланс.")
                continue
            return quantity
```

```
    lower_quantity = get_quantity("Введіть кількість активу для продажу за нижньою межею ціни (наприклад, 1.5, 50%, all): ")
    upper_quantity = get_quantity("Введіть кількість активу для продажу за верхньою межею ціни (наприклад, 1.5, 50%, all): ")

    lower_price, upper_price = request_target_prices()

    lot_size_info = get_lot_size(symbol)
    if lot_size_info is None:
        print("Не вдалося отримати інформацію про лот для даного символу.")
        return None, None, None, None, None

    lower_quantity = adjust_quantity(lower_quantity, lot_size_info['stepSize'])
    upper_quantity = adjust_quantity(upper_quantity, lot_size_info['stepSize'])

    return symbol, asset, lower_quantity, upper_quantity, lower_price, upper_price

def trading_logic():
    while True:
        action = input("Введіть 'buy' для купівлі, 'sell' для продажу, або 'exit' для виходу: ").lower()
        if action == 'buy':
            try:
                symbol, amount, buy_price = request_purchase_info()
                lot_size_info = get_lot_size(symbol)
                quantity = adjust_quantity(amount / buy_price, lot_size_info['stepSize'])
                result = buy_asset_limit(symbol, quantity, buy_price)
```

```
            print("Результат купівлі:", result)
        except Exception as e:
            print(f"Сталася помилка при купівлі: {e}")
    elif action == 'sell':
        try:
            symbol, asset, lower_quantity, upper_quantity, lower_price, upper_price = request_sell_info()
            if symbol:
                while True:
                    current_price = get_price(symbol)
                    print(f"Поточна ціна {symbol}: {current_price}")
                    if current_price <= lower_price:
                                print(f"Виконується продаж {lower_quantity} {asset} за нижньою ціною {current_price}")
                        result = sell_asset(symbol, lower_quantity)
                        print("Результат продажу за нижньою межею:", result)
                        break
                    elif current_price >= upper_price:
                                print(f"Виконується продаж {upper_quantity} {asset} за верхньою ціною {current_price}")
                        result = sell_asset(symbol, upper_quantity)
                        print("Результат продажу за верхньою межею:", result)
                        break
                    time.sleep(10)
        except Exception as e:
            print(f"Сталася помилка при продажу: {e}")
    elif action == 'exit':
```

```
    print("Вихід з програми.")
    break
else:
    print("Введено неправильну команду. Будь ласка, введіть 'buy', 'sell' або 'exit'.")
```