МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
Катерина НЕСТЕРЕНКО
“____”_____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСНИКА ОСВІТНЬОГО СТУПЕНЯ
“БАКАЛАВР”

Тема: “Програмний застосунок електронної торгівлі криптовалютою”

Виконавець: Білик Роман Юрійович

Керівник: к.т.н Талалаєв Володимир Опанасович

Нормоконтролер: Варнавський В’ячеслав Володимирович

Київ 2024

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення
Освітній ступінь бакалавр
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Програмне забезпечення систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Катерина НЕСТЕРЕНКО
"___" _____ 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи студента
Білика Романа Юрійовича

1. Тема проекту: «Програмний застосунок електронної торгівлі криптовалютою»
затверджена наказом ректора від 8.12.2023 р. № 2483/ст

2. Термін виконання проекту: з 3.01.2024 р. до 29.02.2024 р.

3. Вихідні дані до проекту: програмний продукт розробити у вигляді плагіну для графічного редактору Figma, з використанням методів машинного навчання.

4. Зміст пояснювальної записки:
   1. Аналіз предметної галузі та контекст використання застосунку
   2. Вимоги до застосунку та його архітектурні параметри
   3. Реалізація модульної реактивної архітектури застосунку та його компонентів

5. Перелік обов'язкових слайдів презентації:
   1. Акторна модель
   2. Графік динаміки цін активів на високо-волатильному ринку
   3. Огляд архітектурного паттерна розподілених робітників
   4. Огляд циклу роботи застосунку

6. Календарний план-графік

№ пор	Завдання	Термін виконання	Відмітка про виконання
1.	Складання та затвердження графіку кваліфікаційної роботи Написання 1 розділу, представлення керівнику	03.01.24 – 09.01.24	
2.	Попередній друк 1 розділу та допоміжних сторінок (чорновик) - титульної, завдання, графіка, реферат, список скорочень, зміст, вступ, список джерел, 1-й нормо-контроль.	10.01.24 – 17.01.24	
3.	Написання 2 розділу, представлення керівнику	18.01.24 – 22.01.24	
4.	Написання 3 розділу, представлення керівнику	23.01.24 – 26.01.24	
5.	Написання 4 розділу, представлення керівнику	27.01.24 – 31.01.24	
6.	Загальне редагування та друк пояснювальної записки, графічного матеріалу	01.02.24 – 09.02.24	
7.	Проходження нормо-контролю, перевірка на антиплагіат, перепліт пояснювальної записки.	01.02.24 – 05.02.24	
8.	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації	05.02.24 – 09.02.24	
9.	Отримання відгуку керівника, рецензії.	10.02.24 – 22.02.24	
10.	Підготовка матеріалів для передачі секретарю ДЕК (ПЗ, CD-R з електронними копіями ПЗ, презентації, відгук керівника, рецензія) в папці	23.02.24 – 29.02.24	

7. Дата видачі завдання 3.01.2024
Керівник: к.т.н. Володимир ТАЛАЛАЄВ
Завдання прийняв до виконання: Роман БІЛИК
Дата

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Програмний застосунок електронної торгівлі криптовалютою»: 41 с., 5 рис., 2 табл., 26 інформаційних джерел.

МОДЕЛЬ АКТОРІВ, ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ, ДИСТРИБУТИВНІ СИСТЕМИ

Об’єкт розробки – застосунок, який використовує модель акторів для ефективної обробки великого масив даних в дистрибутивній обчислювальній системі.

Мета роботи – створити ефективний відмовостійкий застосунок, з високим ступенем кастомізації, який стане ефективним допоміжним інструментом в роботі криптотрейдерів.

ABSTRACT

Explanatory note to the thesis "Software Application for Electronic Cryptocurrency Trading": 41 p. , 5 Fig. , 2 table. , 26 information sources.

ACTOR MODEL, PARALLEL COMPUTING, DISTRIBUTED SYSTEMS

Property development – application that uses the actor model for efficient processing of a large data arrays in a distributed computing system.

Purpose – create an efficient, fault-tolerant application with a high degree of customization, which will become an effective auxiliary tool in the work of crypto traders.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

HFT – high frequency trading

ЦА – цільова аудиторія

АЗ – автоматизований застосунок

ШІ – штучний інтелект

АС – автоматизована система

ВСТУП

У сфері сучасних фінансових операцій криптовалюти займають значуще місце, забезпечуючи ефективний обмін, анонімність транзакцій та децентралізоване управління. Актуальність криптовалют зумовлена інноваційними розробками у сфері розподілених обчислень та паралельної обробки даних, що відкриває перспективи для створення спеціалізованого програмного забезпечення. Тим не менш, процес розробки ефективних, інтуїтивно зрозумілих та надійних застосунків для електронної торгівлі криптовалютами вимагає глибокого аналізу як технологічних, так і користувацьких аспектів.

У контексті створення програмного забезпечення для криптовалютної торгівлі важливо застосувати модель акторів, що дозволяє ефективно управляти обсягами даних у розподілених системах. Застосування цього підходу сприяє масштабуванню та адаптивності систем, що є вирішальним для задоволення потреб фахівців у сфері криптовалютних інвестицій.

Керівником проекту були поставлені наступні завдання:

1. Визначення та опис контексту проекту розробки: "Програмний застосунок електронної торгівлі криптовалютою".
2. Техніко-економічне обгрунтування доцільності розробки: "Програмний застосунок електронної торгівлі криптовалютою".
3. Проведення обстеження та моделювання ІТ-потреб предметної області: "Програмний застосунок електронної торгівлі криптовалютою".
4. Виявлення і опис вимог користувачів (за моделлю Вігерса): "Програмний застосунок електронної торгівлі криптовалютою".
5. Формування концепції (стратегії) створення програмного застосунку: "Програмний застосунок електронної торгівлі

криптовалютою" і визначення його основних ключових архітектурних параметрів.

6. Вибір і обгрунтування методології і стеку технологій для архітектурного і детального проектування: "Програмний застосунок електронної торгівлі криптовалютою".

7. Вибір і опис архітектури: "Програмний застосунок електронної торгівлі криптовалютою".

8. Розробка прототипу: "Програмний застосунок електронної торгівлі криптовалютою".

Програмний застосунок для електронної торгівлі, заснований на принципах паралельних обчислень та кластеризації, надасть користувачам можливість швидкого доступу до ринкової інформації, управління транзакціями та аналізу ринкових умов. Реалізація такого застосунку спростить процеси торгівлі та інвестування в криптовалюти, забезпечуючи користувачам конкурентні переваги через інтеграцію новітніх технічних рішень.

Розробка даного застосунку передбачає аналіз існуючих криптовалютних трейдингових платформ, визначення користувацьких вимог до програми, і створення реактивної архітектури з використанням моделі акторів для забезпечення високого рівня реактивності системи.

Завдяки такому підходу планується розробити програмне забезпечення, що відповідає вимогам до швидкодії (з англ. performance) та надійності.

РОЗДІЛ 1.

АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА КОНТЕКСТ ВИКОРИСТАННЯ ЗАСТОСУНКУ

1.1. Основні функціональні можливості автоматизованих систем електронної торгівлі криптовалютою

Автоматизована торгівля, відома також як алгоритмічна торгівля, є сучасним підходом до криптовалютних ринків, що дозволяє трейдерам використовувати спеціалізоване програмне забезпечення для виконання торгових операцій. Ці системи використовують комплексні алгоритми, які можуть аналізувати ринкові дані в реальному часі та автоматично реалізовувати торгові стратегії на основі визначених критеріїв. Застосунки для автоматизованої торгівлі можуть використовувати різноманітні підходи, включаючи статистичний аналіз, машинне навчання, технічний аналіз та інші методи для прогнозування змін на криптовалютних ринках та оптимізації торгових рішень. В останні роки застосунки для електронної торгівлі криптовалютою значно змінили ландшафт сучасних криптовалютних ринків і стали невід'ємними компонентами для трейдерів та інвесторів [1].

Мета

Основною метою розробки автоматизованих торговельних застосунків та платформ є підвищення ефективності та прибутковості торгівлі

криптовалютами та мінімізація впливу людського фактору на процес торгівлі.

Функціональні можливості автоматизованих торговельних застосунків охоплюють широкий спектр алгоритмічних і аналітичних інструментів, спрямованих на оптимізацію торговельних стратегій і управління ризиками на фінансових ринках [2, 3]:

- **Алгоритмічна торгівля:** Використання передових алгоритмів для автоматизації торгових рішень на основі аналізу ринкових умов та заданих торговельних параметрів. Алгоритмічна торгівля забезпечує швидке виконання торговельних операцій, що значно підвищує можливості для використання короткострокових ринкових можливостей. Особливо виділяється спеціалізований напрямок високочастотної торгівлі (HFT), що включає в себе використання комплексних алгоритмів для здійснення великої кількості транзакцій на надвисоких швидкостях, що в свою чергу відкриває можливості експлуатувати короткострокові ринкові варіації [4].
- **Технічний аналіз:** Автоматизація аналізу цінових графіків і об'ємів торгів з використанням технічних індикаторів, таких як рухомі середні, індекс відносної сили (RSI) та інші, для ідентифікації тенденцій ринку та потенційних точок входу або виходу з торгів.
- **Машинне навчання:** Застосування алгоритмів машинного навчання для ідентифікації складних ринкових закономірностей та адаптації стратегій в режимі реального часу, що дозволяє системі навчатися та покращувати торговельні рішення.
- **Управління ризиками:** Інтеграція механізмів управління ризиками, таких як автоматичне встановлення стоп-лосів та тейк-профітів, для захисту капіталу та оптимізації прибутковості.
- **Бек-тестування:** Можливість бек-тестування торгових стратегій на історичних даних для оцінки їхньої потенційної ефективності перед застосуванням у реальній торгівлі, що дозволяє ідентифікувати найбільш обіцяючі стратегії.

Цільова аудиторія

ЦА користувачів автоматичних застосунків для криптовалютної торгівлі представляє собою інвесторів та трейдерів на ринку криптовалют. Для фахівців у галузі трейдингу автоматизовані торговельні системи є не лише зручним інструментарієм, а й важливим компонентом для забезпечення конкурентоспроможності. Переваги, що надаються цими системами у вигляді прискорення та підвищення ефективності торговельних операцій, є недосяжними за умов виконання аналогічних завдань вручну [5]. Автоматизовані системи, використовуючи алгоритмічні стратегії, дозволяють трейдерам реалізовувати торговельні можливості, які були б недоступні через обмеження, які визначаються людським фактором. Крім того, здатність системи до безперервної роботи гарантує застосування торговельних стратегій незалежно від фізичної присутності трейдера. Можливість бек-тестування торговельних стратегій на історичних даних також розширює можливості трейдерів, дозволяючи їм відточувати та адаптувати свої підходи до запуску на реальному ринку. Цей аспект сприяє мінімізації ризиків та оптимізації потенціалу прибутковості стратегій [6].

1.2. Огляд ефективних стратегій для торговельних автоматизованих застосунків у криптовалютному секторі

Ефективність торговельних АЗ залежить від вибору та оптимізації торговельних стратегій, які можуть адаптуватися до швидкозмінних умов криптовалютних ринків. Опираючись на результати аналізу наукових робіт були ідентифіковані ключові стратегії, які рекомендується інтегрувати у програмне забезпечення автоматизованого застосунку для торгівлі криптовалютами:

Трикутна арбітражна стратегія

Хоча конкретні результати варіюються, кілька досліджень вказують, що добре спроектовані торговельні АЗ та системи HFT можуть бути прибутковими, особливо при використанні можливостей арбітражу та експлуатації неефективностей ринку, які поширені на фрагментованих ринках криптовалют.

Однією зі стратегій, що являється прибутковою, є трикутна арбітражна стратегія, яка використовується АЗ для конвертації між двома криптовалютами через проміжну валюту, отримуючи вигідний курс обміну порівняно з прямим обміном. Ця стратегія показала, що вона може забезпечити кращий обмінний курс на 0.144% або 14.4 базисних пунктів (bps) порівняно з прямим курсом обміну, і 2.71% усіх транзакцій на Binance приписуються цій стратегії [7].

Фронтраннінг і MEV

Дослідження практики фронтраннінгу та можливості майнерів витягувати додаткову вартість (Miner Extractable Value - MEV) у контексті децентралізованих бірж підкреслює активне використання арбітражних АЗ для використання неефективностей у системі DEX, що загрожує справедливості та прозорості ринкових операцій. [8].

Ціноутворення за допомогою інтелектуальної автоматизованої системи

Застосунок SPA Bot, який застосовує динамічні методи ціноутворення на основі аналізу руху цін, демонструє, що традиційні підходи можуть бути ефективно інтегровані в автоматизовані торговельні стратегії, підвищуючи їхню ефективність на криптовалютних ринках. [9].

Аналіз цих досліджень показав, що для того, щоб досягнути ефективності зазначених стратегій було застосоване програмне забезпечення, яке включає в себе елементи високочастотної торгівлі (HFT), оскільки вони використовують складні стратегії, такі як трикутна арбітражна стратегія, фронтраннінг і MEV, а також динамічне ціноутворення, для отримання прибутку на неефективностях ринку криптовалют.

1.3. Проблематика застосування автоматизованих систем для торгівлі та обмежуючі фактори

АЗ для торгівлі на ринках криптовалют стикаються з рядом унікальних викликів та обмежень, які формують специфічний контекст цієї сфери фінансових операцій. Висока волатильність криптовалют призводить до швидких і різких коливань цін, що значно ускладнює застосування попередньо

запрограмованих торгових стратегій (див. Рис. 1.1.). Така непередбачуваність може призвести не тільки до втрати потенційно прибуткових можливостей, а й до реалізації транзакцій, які вже не виправдовують себе з точки зору прибутковості.

Рис. 1.1. Приклад різкого падіння ціни активу в середині червня

Крім того, на високо волатильних ринках криптовалют відмічається явище так званого «ковзання» (англ. 'slippage'), коли існує значна різниця між очікуваною та фактичною ціною угоди, що негативно впливає на ефективність та прибутковість торгових операцій [10]. Ринок криптовалют також характеризується миттєвими обвалами, коли ціни на криптовалюти стрімко падають і так само швидко відновлюються протягом дуже короткого часу. Це може бути посилено високошвидкісною природою автоматизованої торгівлі та великим обсягом замовлень, спричиняючи значну ринкову нестабільність.

Правовий аспект є ще однією важливою сферою викликів, оскільки регулювання криптовалют та автоматизованої торгівлі все ще знаходиться на початковій стадії. Різні юрисдикції мають різний рівень регулювання обігу

криптовалют, що створює складний правовий ландшафт для трейдерів та розробників торговельних систем [11].

Технічні та операційні ризики також відіграють важливу роль, оскільки низька затримка є ключовою для успіху стратегій високочастотної торгівлі, а затримки у виконанні замовлень, викликані затримками в мережі або обробці даних, можуть призвести до значного ковзання та потенційних втрат.

Торговельні АЗ також мають високий ризик технічних збоїв та відмови системи, де програмні помилки, збої обладнання або проблеми з підключенням можуть спричинити непередбачувані транзакції, дублювання замовлень або втрату торговельних можливостей. Вразливості безпеки становлять окрему загрозу, оскільки автоматизовані торговельні системи, платформи та гаманці можуть бути потенційними цілями для кіберзлочинців, що може призвести до крадіжки криптовалют, несанкціонованого доступу до торговельних стратегій та маніпуляції ринковими даними.

Ці виклики та обмеження підкреслюють необхідність постійних досліджень та застосування надійних архітектурних моделей для забезпечення відмовостійкого дизайну системи з метою зниження ризиків, пов'язаних з автоматизованою торгівлею на ринках криптовалют. Трейдерам та розробникам систем необхідно слідкувати за передовими розробками у цій галузі, щоб забезпечити стабільність та цілісність автоматизованих торговельних систем.

Висновки

У даному розділі кваліфікаційної роботи було виконано наступні ключові завдання для розробки програмного застосунку електронної торгівлі криптовалютою:

1. Було визначено та описано контекст проекту, зосереджуючись на створенні платформи для електронної торгівлі криптовалютами. Основна увага була приділена визначенню потреб ринку та користувачів, а також особливостям криптовалютного сектору.

2. Проведене техніко-економічне обгрунтування проекту, включаєло аналіз прибуткових стратегій, які можна інтегрувати, щоби застосунок був рентабельним.

3. Сформовано уявлення про цільову аудиторію додатку, яка представляє собою трейдерів та інвесторів у сфері криптовалютної торгівлі, якім потрібні ефективні інструменти, здатні нівелювати людський фактор в їх роботі та покаращити її ефективність.

Цей аналітичний підхід дозволив закласти міцний фундамент для подальшої розробки програмного застосунку, орієнтованого на високочастотну торгівлю криптовалютами, з урахуванням всіх ключових аспектів та викликів, що супроводжують цей процес.

РОЗДІЛ 2.
ВИМОГИ ДО ЗАСТОСУНКУ ТА ЙОГО АРХІТЕКТУРНІ ПАРАМЕТРИ

2.1 Аналіз та порівняння різних моделей автоматизованих застосунків для криптовалютної торгівлі

У зв'язку з прийнятим рішенням щодо розробки автоматизованої системи торгівлі з акцентом на високочастотну торгівлю (HFT), заплановано провести комплексний аналіз існуючих моделей HFT-додатків. Основна мета аналізу полягатиме у визначенні оптимального набору технологій та архітектурних рішень, які сприятимуть підвищенню конкурентоспроможності розроблюваного додатку на основі ключових показників ефективності для HFT-систем, зокрема: продуктивності, масштабованості та надійності. Ці параметри є вирішальними для досягнення високої ефективності в роботі HFT-додатків, забезпечуючи їм здатність оперативно обробляти великі об'єми даних, адаптуватися до змін на ринку та забезпечувати безперебійну роботу.

Порівняльний аналіз моделей застосунків та фреймворків для криптовалютної торгівлі сприятиме об'єктивній оцінці при визначенні функціональних та нефункціональних вимог, які в свою чергу також будуть визначними критеріями у виборі оптимального технологічного стеку та архітектури для розробки ефективного застосунку, який задовольнить головну мету кваліфікаційної роботи:

1. **Hummingbot** - це open-source фреймворк для створення

автоматизованих торгових стратегій, або АЗ, що працюють на криптовалютних біржах. Підтримує з'єднання як з централізованими так і з децентралізованими біржами [12]. Продуктивність залежить від обраної стратегії та конфігурації, але фреймворк забезпечує ефективну основу для швидкої розробки та тестування стратегій.

2. **Blackbird Bitcoin Arbitrage -** це система для автоматичного арбітражу між біржами Bitcoin, яка використовує C++ для торгівлі. Система розроблена з метою досягнення прибутку з тимчасових різниць у цінах між біржами, будучи ринково-нейтральною [2]. Продуктивність висока, з урахуванням арбітражних можливостей та ринкової нейтральності стратегії. Використання C++ вказує на потребу в високій швидкості виконання та ефективності, що є критично важливим в арбітражній торгівлі, де можливості можуть бути швидкоплинними.

3. **Axial-LOB-High-Frequency-Trading-with-Axial-Attention** репозиторій містить реалізацію PyTorch дослідження, що використовує технології машинного навчання для високочастотної торгівлі. Включає в себе повний пайплайн для тренування та тестування моделі. Продуктивність залежить від якості використовуваних датасетів і точності моделі, але машинне навчання може забезпечити високу точність в аналізі ринкових даних. [13].

4. **Magic8bot -** це торговий застосунок для криптовалютної торгівлі, якийпідтримує обміни"coinbasepro" та "binance". Розробка АЗ ведеться з акцентом на мікросервісну архітектуру для покращення масштабованості та гнучкості. Продуктивність може варіюватися в залежності від обраної стратегії та налаштувань, але мікросервісна архітектура сприяє ефективному розподілу ресурсів і швидкому виконанню операцій [14].

4. **Octobot -** це потужний відкритий криптовалютний торговий АЗ, який дозволяє високу кастомізацію за допомогою своєї конфігурації та системи "tentacles". Застосунок підтримує інтеграцію з машинним навчанням для створення потужних торгових стратегій. Широкі можливості кастомізації та інтеграції з ШІ можуть призвести до створення ефективних стратегій, але

продуктивність залежить від якості реалізації цих стратегій та використання даних.[15].

5. **PingPong** - це фреймворк для створення торгових АЗ, написаний на мові Julia. Особливість фреймворку в тому, що він повністю написаний на Julia, що забезпечує високу продуктивність як під час бектестінгу, так і в реальній торгівлі [16]. Продуктивність є високою завдяки використанню мови програмування Julia, що дозволяє ефективно обробляти великі обсяги даних і швидко виконувати торгові операції.

Розглянуті фреймворки та АЗ представляють широкий спектр підходів до автоматизованої торгівлі на криптовалютних ринках: від гнучких фреймворків, як Hummingbot, до спеціалізованих рішень для арбітражу між біржами, як Blackbird. Сучасні технології машинного навчання, використані в Axial-LOB-HFT, демонструють інноваційний підхід до високочастотної торгівлі, тоді як модульна архітектура Magic8bot і висока кастомізація OctoBot надають трейдерам гнучкість у розробці та адаптації стратегій. PingPong.jl вирізняється завдяки обчислювальній ефективності, яку забезпечує мова програмування Julia, пропонуючи високу ефективність операцій в режимі реальному часі. Кожен застосунок або фреймворк має свої сильні сторони, і вибір залежить від конкретних потреб та вимог трейдера до продуктивності, кастомізації та специфіки торговельних стратегій (див. табл. 2.1.).

Таблиця 2.1.

Особливість/Аспект	Hummingbot	Blackbird	Magic8bot	OctoBot	PingPong.jl
Основна Мова	Python	C++	TypeScript, JavaScript	Python	Julia
Фокус	Автоматизація торгівлі	Арбітраж між біржами	Торгівля на підтримуваних біржах	Кастомізована торгівля	Створення торгових АЗ
Продуктивність	Залежить від стратегії	Висока, залежить від арбітражу	Залежить від стратегії	Висока, залежить від конфігу	Висока, завдяки Julia
Паралелізм	Підтримується	Обмежений	Підтримується	Підтримується	Підтримується
Масштабованість	Висока	Обмежена	Висока	Висока	Висока
Надійність	Залежить від конфігурації	Висока, з ринковою нейтральністю	Залежить від конфігурації	Висока	Висока, з додатковими перевірками
Інтерфейс Користувача	CLI, веб-інтерфейс	CLI	Веб-інтерфейс	Веб-інтерфейс, Telegram	CLI, можливість розширення
Обробка даних в реальному часі	Підтримується	Обмежена	Підтримується	Підтримується	Підтримується

2.2. Вимоги до застосунку для торгівлі криптовалютою з високою частотою

На основі отриманих даних про характеристики різних моделей застосунків та автоматизованих систем для криптовалютної торгівлі, а також аналізу відгуків трейдерів згідно методології Вігерса [17] були визначені наступні вимоги:іол

1.1. Основні функціональні характеристики

1.1.1. Аналіз ринкових індикаторів - алгоритмічні торговельні системи (АТС) обробляють та аналізують дані ринку в режимі реального часу, включно з великими об'ємами інформації про ціни, зміни та обсяги торгів для швидкого прийняття рішень щодо торгівлі.

1.1.2. Виконання операцій - швидкість та точність у виконанні торговельних операцій визначають ефективність АТС.

1.1.3. Управління ризиками - імплементація механізмів управління ризиками, включаючи стоп-лосс ліміти, управління розміром позицій та диверсифікацію портфеля, є ключовою для мінімізації потенційних збитків

1.2. Функціональні характеристики, специфічні для користувача

1.2.1. Конфігурація стратегій - система надає можливість користувачам кастомізувати та адаптувати торговельні стратегії відповідно до їхніх унікальних вимог та переваг.

1.2.2. Сповіщення в режимі реального часу користувачі можуть отримувати інформацію про ринкові події, завершення операцій або алерти щодо ризиків негайно, що сприяє ефективному прийняттю рішень.

2.1. Нефункціональні Вимоги:

2.1.1. Обчислювальна продуктивність - АТС демонструє високу обчислювальну продуктивність з мінімальним часом відгуку та високою пропускною здатністю для швидкої обробки даних та виконання операцій.

2.1.2 Масштабованість - система спроектована для ефективного масштабування та обробки збільшеного обсягу даних, особливо під час пікових навантажень.

2.1.3 Надійність та відмовостійкість - АТС розроблена з урахуванням вимог до надійності та здатності до безперервної роботи та відновлення після збоїв.

2.1.4 Безпека - з урахуванням фінансових ризиків, система включає передові механізми захисту для забезпечення цілісності та конфіденційності даних користувачів та торговельних операцій.

2.3. Вибір технологічного стека та переваги над іншими моделями

Опираючись на визначені функціональні та нефункціональні вимоги, а також аналіз інших моделей застосунків для криптовалютної торгівлі, було прийнято рішення використовувати фреймворк Akka та асоційовані з ним технології такі як Streams, Persistence та Cluster. Akka представляє собою комплексний інструментарій і середовище для розробки високоефективних розподілених систем з високим рівнем відмовостійкості, що реалізуються на платформі JVM (Java Virtual Machine). Розробка Akka фреймворку була здійснена за допомогою мови Scala [18].

Переваги застосування Akka стають особливо помітними при порівнянні з іншими підходами до архітектури систем. Одним із ключових аспектів є підвищення ефективності паралелізму та масштабованості через модель акторів Akka, яка дозволяє з легкістю розробляти системи, оптимізовані для одночасної обробки великих обсягів операцій, що надзвичайно важливо для HFT стратегій. Водночас, інші мови програмування, такі як Python та JavaScript, хоч і підтримують паралелізм, але можуть не надавати такої ж масштабованості та оптимального використання ресурсів, а C++ хоча і вирізняється високою продуктивністю, вимагає значних зусиль для ефективного управління паралельними обчисленнями.

Відмовостійкість є ще однією перевагою Akka, де модель "Let it Crash" дозволяє створювати системи, здатні до самовідновлення шляхом перезапуску компонентів, що зазнали збоїв, без негативного впливу на загальну стабільність системи, в той час як традиційні підходи до обробки помилок можуть не забезпечити аналогічний рівень відмовостійкості.

Обробка потоків даних з Akka Streams відкриває можливості для створення комплексних пайплайнів, здатних витримувати високі навантаження,

що є вирішальним для аналізу ринкових даних в режимі реального часу. Це контрастує з можливими втратами даних або проблемами з продуктивністю в інших системах, що не забезпечують подібної ефективності обробки.

Додатково, Akka Cluster сприяє створенню розподілених застосунків, здатних до горизонтального масштабування на кількох машинах, що розширює можливості обчислювальних систем у порівнянні з рішеннями, що не підтримують розподілене виконання в базовій конфігурації.

На завершення, модульність Akka сприяє розробці добре структурованих та легко підтримуваних систем, де окремі компоненти можуть бути розроблені, протестовані та розгорнуті незалежно, знижуючи час розробки та спрощуючи процес підтримки, на відміну від більш складних і монолітних архітектур.

Таким чином, архітектура на базі Akka та супутніх технологій забезпечує значні переваги у продуктивності, масштабованості, відмовостійкості та легкості підтримки порівняно з альтернативними технологічними рішеннями, виправдовуючи свій вибір для автоматизованої системи криптовалютної торгівлі.

2.4. Опис моделі акторів та вибір архітектури застосунка

Враховуючи, що Akka фреймворк було розроблено для імплементації акторної моделі, розглянемо її основні властивості:

Акторна модель є концептуальним фреймворком, який спрощує розробку конкурентних та розподілених систем [19]. Вона пропонує високорівневу абстракцію для управління станом та поведінкою в системі, де актори є основними одиницями обчислення. В цій моделі, актори є незалежними одиницями, які коммунікують, відправляючи повідомлення один одному. Кожен актор є унікальним, має поштову скриньку для вхідних повідомлень, поведінку, яка визначає, як він відповідає на повідомлення та ізольований стан (див. Рис 2.1).

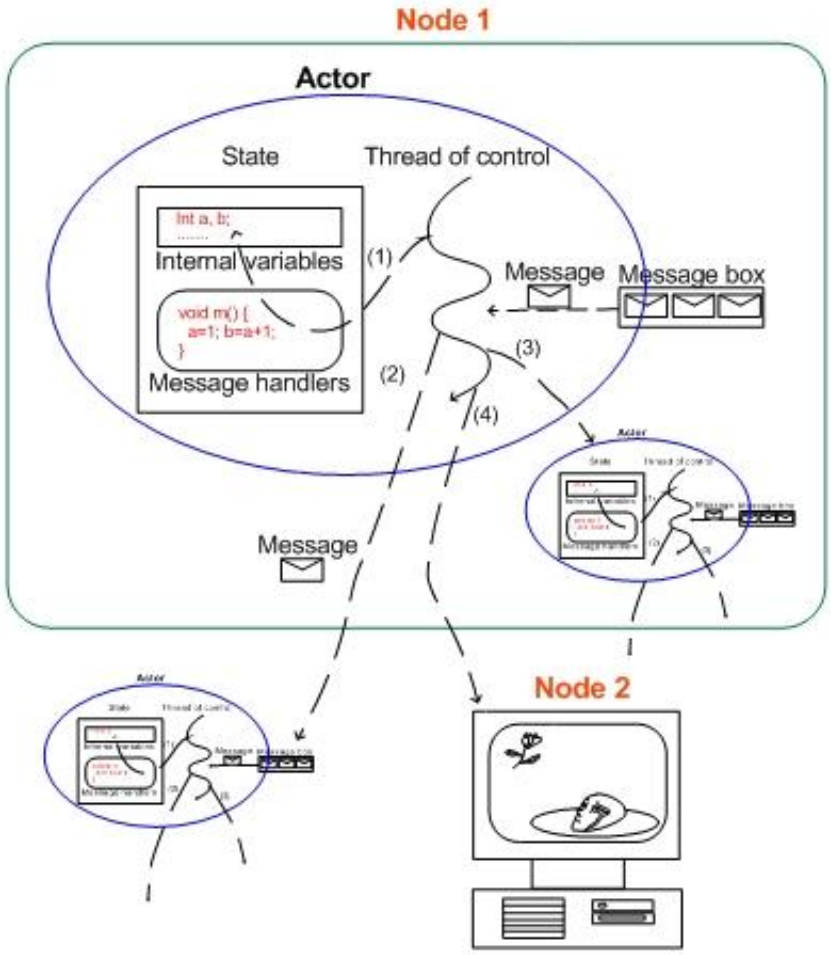

Рис. 2.1 Процес асинхронної комунікації між акторами

Основні поняття та компоненти:

- Актори: Найменші одиниці обчислення в акторній моделі. Кожен актор може виконувати завдання, приймати локальні рішення, створювати більше акторів і відправляти повідомлення іншим акторам.
- Повідомлення: Єдиний спосіб спілкування акторів та взаємодії між ними. Повідомлення є незмінними (immutable), що забезпечує відсутність спільного стану або стану гонитви (race condition).
- Поштові скриньки: Черги, асоційовані з кожним актором, які зберігають вхідні повідомлення до того, як актор їх обробить. Це забезпечує послідовну та неблокуючу обробку повідомлень.
- Нагляд: Ієрархія акторів, де батьківські актори можуть наглядати та управляти життєвим циклом своїх дочірніх акторів, забезпечуючи

структурований спосіб обробки помилок та підтримання стійкості системи.

Для того щоб в повній мірі використати переваги акторів в розробці HFT застосунку, була вибрана модульна реактивна модель архітектури, яка є орієнтованою на розподіл задач між модулями і використовує паттерн «розподілених робітників» [20].

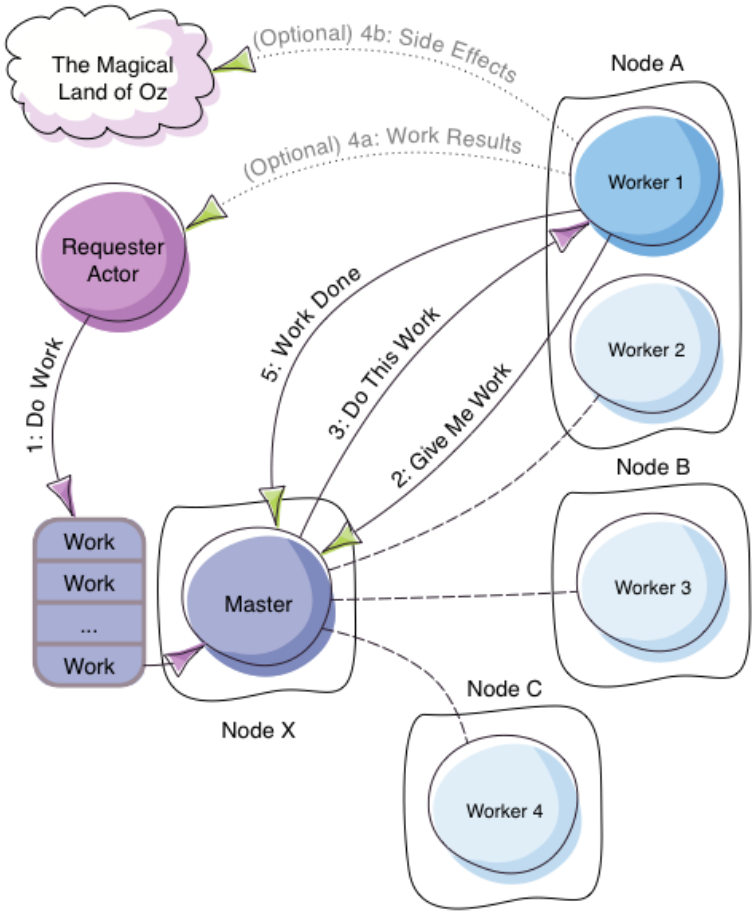

Рис. 2.2. Схема роботи паттерну розподілених робітників

Архітектура програмної системи є критичним фактором її ефективності, особливо у вимогливій сфері АЗ для торгівлі з високою частотою. Унікальні виклики HFT, такі як потреба у швидкій обробці даних, високій надійності та масштабованості, вимагають відмовостійкої та гнучкої архітектури. Архітектура реалізована за допомогою паттерна «розподілених робітників» є тим рішенням, яке задовольняє необхідній критерії (див. табл. 2.2).

Таблиця 2.2.

Аспект	Розподілена архітектура працівників з Akka	Традиційна монолітна архітектура	Архітектура мікросервісів
Продуктивність	Висока продуктивність завдяки моделі акторів, що забезпечує ефективні, одночасні операції.	Потенційно обмежена однопотоковими процесами або складними багатопотоковими.	Висока продуктивність, але мережева затримка між сервісами може бути перешкоджаючим фактором.
Масштабованість	Висока, підтримує як вертикальне, так і горизонтальне масштабування безперебійно з Akka Cluster.	Масштабування часто вимагає значних архітектурних змін або обмежене вертикальним масштабуванням.	Висока горизонтальна масштабованість, оскільки сервіси можуть масштабуватися незалежно.
Відмовостійкість	Висока, з вбудованими механізмами, такими як нагляд за акторами та Akka Persistence для відновлення стану.	Відмовостійкість має бути спеціально імплементована, що може призвести до складної обробки помилок.	Висока, оскільки збій в одному сервісі не обов'язково впливає на інші, але вимагає ретельного проектування міжсервісного зв'язку.
Затримка	Низька завдяки ефективній передачі повідомлень між акторами та гнучкості розгортання з Akka Cluster.	Може бути значною через менш оптимізовані механізми зв'язку та обробки.	Може бути оптимізована в межах сервісів, але міжсервісний зв'язок може внести додаткову затримку.
Використання ресурсів	Ефективне завдяки неблокуючим асинхронним вводу/виводу операціям та акторно-орієнтованій одночасності.	Потенційна недостатність використання через блокування вводу/виводу або надмірне використання через неефективну обробку.	Загалом ефективне, але накладні витрати на численні сервіси та міжсервісний зв'язок можуть вплинути на використання ресурсів.
Безпека	Може бути адаптована з конкретними механізмами безпеки, інтегрованими в систему акторів.	Реалізації можуть бути спорадичними та непослідовними у застосуванні.	Може бути надійною, якщо сервіси розроблені з урахуванням безпеки, але вимагає ретельного управління дозволами між сервісами.
Обробка даних в реальному часі	Відмінно з Akka Streams для створення пайпланів обробки даних.	Може бути обмеженою здатністю архітектури ефективно обробляти великі обсяги даних.	Висока в межах окремих сервісів, але агрегація та обробка даних між сервісами може бути складною.

2.5. Взаємодія з API біржі через WebSocket за допомогою Akka Streams

Інтеграція з API фінансових бірж є критично важливою для HFT застосунків, які потребують швидкого та ефективного з’єднання для торгівлі в умовах постійних змін ринкових даних. Даний застосунок використовує WebSockets у поєднанні з Akka Streams для цієї мети, відповідаючи стандартам з’єднання для різних торгівельних платформ.

За допомогою WebSockets встановлюється постійний двосторонній комунікаційний канал, який Akka Streams оптимізує, ефективно управляючи обміном повідомлень та мінімізуючи накладні витрати. Це забезпечує своєчасні оновлення ринкових даних та книги замовлень (order book), дозволяючи HFT застосунку швидко реагувати на зміни на ринку.

Використання WebSockets, підтримуване реактивною обробкою Akka Streams, не тільки значно знижує затримку при виконанні замовлень, але й спрощує комунікацію, підтримуючи одне постійне з'єднання, уникаючи затримок та витрат ресурсів на відключення та підключення.

Вибір WebSockets у поєднанні з Akka Streams зумовлений потребою в ультранизькій затримці, що є критичним у чутливому до часу середовищі HFT. Ця комбінація підтримує одночасну обробку кількох потоків даних через одне з'єднання, що є важливим також для масштабованості, оскільки HFT застосункам потрібно одночасно проводити моніторинг даних та проводити транзакції на різних ринках або з різними активами. Більше того, це полегшує динамічний підхід до торгівлі, дозволяючи застосунку в реальному часі коригувати свої стратегії на основі поточних ринкових умов.

2.6. Інтеграція бази даних у розподілену робочу архітектуру

У контексті високочастотної торгівлі, ефективна обробка значних обсягів даних є ключовим фактором для забезпечення високої продуктивності торговельних систем. Даний застосунок базується на розподіленій архітектурі, оптимізованій для використання потенціалу фреймворку Akka, та інтегрує високонадійне рішення для управління базами даних, щоб задовольнити

зазначені вимоги. Обрана система управління базами даних Apache Cassandra, сервер якої знаходиться на локальній машині та попередньо конфігурований за відповідною схемою, необхідна для взаємодії з Akka persistence [21,22]

Cassandra має виняткову масштабованість та відмовостійкість, що робить її ідеальним вибором для вимог HFT застосунків.

Розподілена архітектура Cassandra дозволяє гармонійно інтегруватися з розподіленою архітектурою застосунку, забезпечуючи можливість горизонтального масштабування та здатність системи ефективно обробляти великі масиви ринкових даних без втрати продуктивності. Важливою перевагою Cassandra є також її здатність до реплікації даних на різних вузлах, що значно підвищує надійність системи, сприяючи забезпеченню високого рівня відмовостійкості.

Основним аспектом інтеграції бази даних є її синергія з Akka Persistence, ключовою складовою інструментарію Akka, яка дозволяє забезпечити стійке зберігання станів акторів. Актори в системі Akka відповідають за капсулювання торговельних стратегій та логіки прийняття рішень, тому збереження їх стану в Cassandra через Akka Persistence є критично важливим для забезпечення можливості відновлення системи після збоїв. Це дозволяє акторам відновлювати свою діяльність з останніх відомих консистентних станів, мінімізуючи ризики втрати даних і забезпечуючи безперебійність торговельних операцій.

2.7. Інтерфейс та налаштування стратегій

Розроблений інтерфейс, зосереджений на принципах мінімалізму та функціональності, базується на командному рядку, який слугує як основний механізм взаємодії з системою. Такий консольний інтерфейс підтримує виконання різноманітних команд, які уможливлюють налаштування, запуск, зупинку та перевірку стану торговельних ботів, а також управління конфігураціями стратегій.

Доступність консольного інтерфейсу гарантує, що користувачі можуть швидко та ефективно керувати застосунком без потреби в складних графічних інтерфейсах користувача.

В основі гнучкості та адаптивності системи лежить механізм управління конфігурацією стратегій. Система використовує конфігураційний файл на основі JSON, який визначає торговельні стратегії, які повинен виконувати актор (Worker) відповідальний за безпосереднє виконання роботи в паттерні «розподілених робітників» [20]. Такий підхід дозволяє досягти високого ступеня кастомізації та динаміки у розгортанні стратегій.

Висновки

У ході вивчення різноманітних моделей криптовалютних застосунків, особлива увага була приділена вибору технологічного стеку, що включає Akka фреймворк та його бібліотеки, що дозволяє інтегрувати ключові переваги HFT систем і забезпечити конкурентні переваги застосунку в сфері продуктивності. Обрано модульну та реактивну архітектурну модель для оптимального використання потенціалу стеку в контексті HFT-застосунків, що дозволяє масштабувати обчислювальні ресурси та застосовувати ефективні стратегії арбітражу, забезпечуючи високу відмовостійкість системи через кластерну розподілену структуру.

У процесі розробки було виконано наступні додаткові завдання:

4. **Виявлення і опис вимог користувачів (за моделлю Вігерса)**: Було детально проаналізовано та описано вимоги користувачів до програмного застосунку електронної торгівлі криптовалютою, включаючи функціональні, нефункціональні вимоги, а також вимоги до інтерфейсу і безпеки.
5. **Формування концепції (стратегії) створення програмного застосунку**: Була розроблена концепція програмного застосунку електронної торгівлі криптовалютою, включаючи визначення його ключових архітектурних параметрів, що забезпечують відповідність високим вимогам швидкості, безпеки та масштабованості.

6. **Вибір і обгрунтування методології і стеку технологій для архітектурного і детального проектування**: Опрацьовано та обґрунтовано вибір методологій і технологічного стеку для проектування архітектури програмного застосунку, виходячи з потреб проекту та сучасних ІТ-тенденцій.

7. **Вибір і опис архітектури**: Розроблено детальний опис архітектури програмного застосунку електронної торгівлі криптовалютою, що враховує необхідність забезпечення високої пропускної здатності, надійності та масштабованості системи.

Таким чином, здійснений комплексний аналіз та практичні кроки дозволили закласти міцний фундамент для розвитку та запуску програмного застосунку, орієнтованого на високоефективну торгівлю криптовалютами з використанням переваг HFT-технологій.

РОЗДІЛ 3.
РЕАЛІЗАЦІЯ МОДУЛЬНОЇ РЕАКТИВНОЇ АРХІТЕКТУРИ ЗАСТОСУНКУ ТА ЙОГО КОМПОНЕНТІВ.

3.1. Обмеження проекту

В рамках реалізації ключового функціоналу даного проекту було здійснено інтеграцію виключно з однією криптовалютною платформою – Bitfinex [23]. Таке рішення було обумовлено обмеженнями ресурсів проекту та комплексністю імплементації взаємодії з такою кількістю бірж, яка є необхідною для реалізації арбітражних стратегій. Водночас, впровадження інтерфейсу для Bitfinex створює основу для майбутнього розширення функціоналу шляхом систематичного додавання нових інтерфейсів для інших торговельних платформ, що дозволить горизонтальне масштабування застосунку в процесі подальшої розробки.

Важливим аспектом розробки була верифікація коректності механізму обробки ринкових даних за допомогою Akka Streams, який має забезпечувати моніторинг інформації з біржі в режимі реального часу та перетворювати в її формат, який дозволить провести подальшої обробки акторами системи.

Тестування функціоналу було здійснено з використанням унікальної функції біржі Bitfinex, що дозволяє створення суб-акаунту для торгівлі в тестовому режимі з віртуальними коштами [24]. Ця особливість відіграла ключову роль у виборі Bitfinex як первинної платформи для інтеграції. В

рамках тестування було застосовано просту скальпінгову стратегію, яка базується на використанні мінімальних цінових розбіжностей [25].

Таким чином використання обмеженої кількості інтерфейсів у початковій версії проекту дозволило сфокусуватися на точній реалізації та тестуванні ключових компонентів системи закладючи фундамент для подальшого функціоналу та інтеграції з різноманітними торговельними платформами в майбутньому.

3.2. Головні компоненти імплементації архітектурного паттерна «розподілених робітників» та їх функції

У рамках розробки даного проекту було імплементовано модульну архітектуру для асинхронного збору та обробки ринкових даних з криптовалютної біржі Bitfinex (див. Рис. 3.1):

- **FrontEnd.scala -** неперервно та асинхронно отримує ринкові дані з

біржі Biffinex по встановленому двосторонньому з'єднанню через WebSocket Api та трансформує ці дані в задачі для обробки (work items), які відправляються актору «Менеджер» (**WorkManager.scala**) для подальшої обробки. Дана конфігурація гарантує оптимізовану та масштабовану обробку потоків інформації в режимі реального часу, що сприяє миттєвому відгуку на коливання ринкових умов та уможливлює адаптивне формування завдань залежно від аналізу вхідних даних.

- **Work.scala - в**изначає клас Work, який представляє одиницю роботи

(work item), з унікальним ідентифікатором та енкапсулює ринкові дані для передачі їх між акторами. Застосовує механізми серіалізації для ефективної передачі між вузлами кластера.

- **WorkManager.scala –** головний компонент реактивної архітектури.

Координує розподіл задач між виконавцями, забезпечуючи балансування навантаження та моніторинг стану виконання. Для збереження стану менеджер

використовує можливості Akka Persistence, що дозволяє відновити процес обробки після збоїв.

- **Worker.scala - в**иступає як посередник, який делегує фактичну роботу актору WorkExecutor. Наглядає за виконанням роботи, залишається доступним для нової роботи від WorkManager та обробляє помилки при виконанні роботи.
- **WorkExecutor.scala -** безпосередньо обробляє ринкові дані та виконує торговельні операції згідно з визначеною стратегією. Логіка кожної задачі конфігурується користувачем і залежить від параметрів, заданих у конфігураційному файлі.
- **Main.scala -** точка входу в застосунок, ініціалізує і запускає всі компоненти системи, включаючи ініціацію системи акторів Akka різними ролями, створення кластеру та налаштування взаємодії між компонентами.

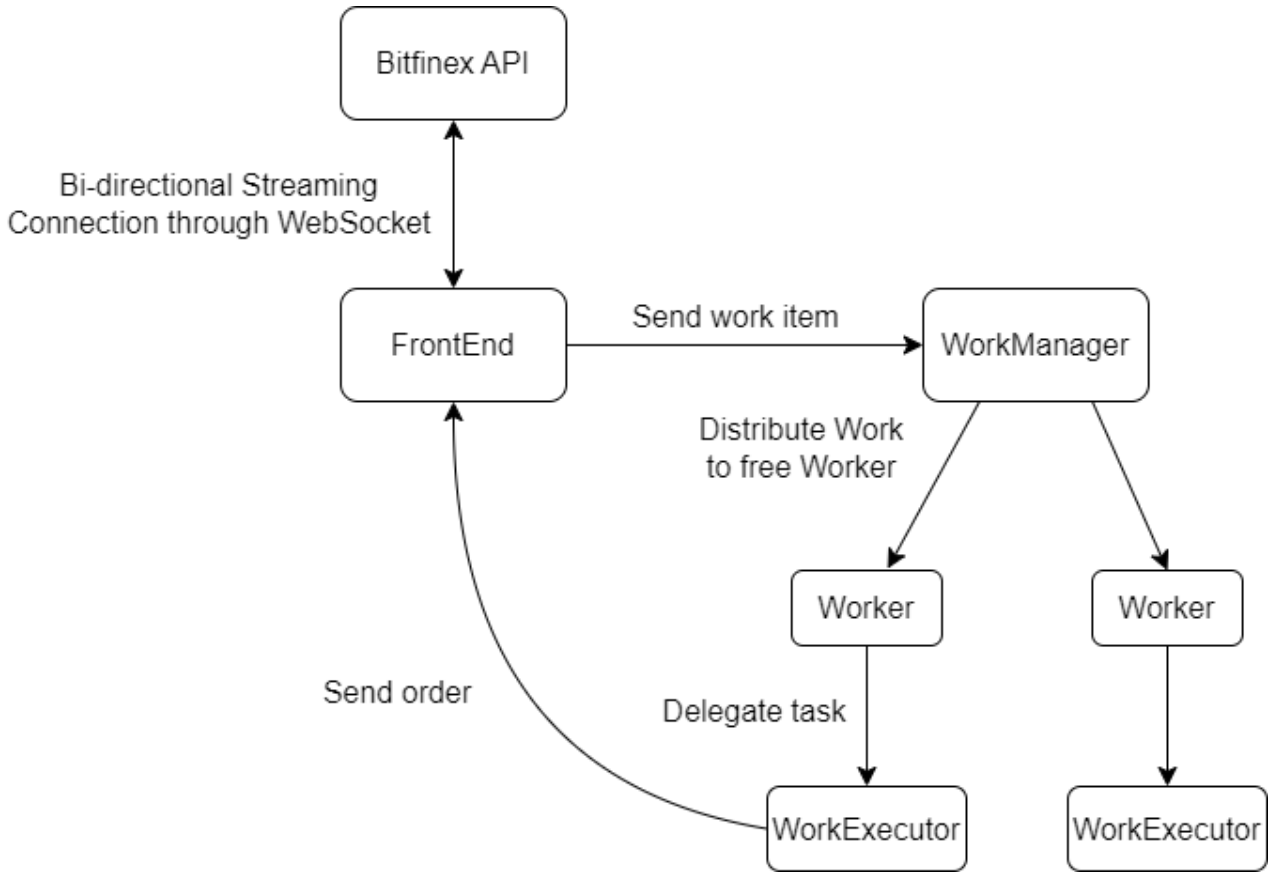

Рис. 3.1 Цикл роботи застосунку електронної торгівлі криптовалютою

3.3. Реалізація методу для двохстороннього з'єднання з BitFinex API

Метод connectWebSocket() встановлює з'єднання через WebSocket до API Bitfinex, зокрема для підписки на обробку потоків даних з біржі в режимі реального часу. Комунікацію ініціюється через протокол WebSocket із API торговельної платформи Bitfinex, зокрема, для отримання потокових даних у режимі реального часу. Ця операція запускається з активації з'єднання з сервером WebSocket, що здійснюється за допомогою ініціації клієнтського сеансу WebSocket до API Bitfinex використовуючи метод webSocketClientFlow від Akka HTTP, цілеспрямовано на URI "wss://api.bitfinex.com/ws/2", який представляє собою кінцеву точку WebSocket для API Bitfinex.

Процедура включає в себе створення джерела потоку, messageSource, яке ініціює відправку запиту на підписку до WebSocket Bitfinex. Цей запит активує підписку на отримання специфічних ринкових даних, зокрема, інформації про книгу ордерів ("book") для валютної пари BTC/USD з детальними параметрами, такими як точність і частота оновлень. Після ініціалізації підписки, система зберігає активне з'єднання, використовуючи конкатенацію з Source.maybe[Message], що дозволяє підтримувати з'єднання безперервно відкритим.

Визначається потік-джерело, messageSource, для відправлення початкового повідомлення про підписку на WebSocket Bitfinex. Це повідомлення підписує клієнта на конкретні ринкові дані, у цьому випадку - на книгу замовлень ("book") для пари BTC/USD з певними параметрами точності, частоти і т.д.. Після відправлення повідомлення про підписку, джерело утримує з'єднання відкритим, конкатенуючи його з Source.maybe[Message], ефективно утримуючи з'єднання активним.

```
def connectWebSocket(): Unit = {
  log.info("Initializing WebSocket connection to Bitfinex API with
new pattern")
  // WebSocket client flow to Bitfinex
  val webSocketFlow =
Http()(context.system.toClassic).webSocketClientFlow(WebSocketRequest
("wss://api.bitfinex.com/ws/2"))

  // Source that sends the initial subscription message and then
keeps the connection open
```

```
  val messageSource: Source[Message, Any] =

Source.single(TextMessage("""{"event":"subscribe","channel":"book","s
ymbol":"tBTCUSD","prec":"P0","freq":"F0","len":"25","subId":
123}"""))
      .concat(Source.maybe[Message]) // Keeps the connection open

  val measurementsWebSocket: Flow[Message, String, Any] =
    Flow[Message]
      .collect {
        case TextMessage.Strict(text) => Future.successful(text)
        case TextMessage.Streamed(textStream) =>
          textStream.runFold("")(_ + _).flatMap(Future.successful)
      }
      .mapAsync(1)(identity)
      .groupedWithin(1000, 1.second)
      .mapAsync(1) { messages =>
        val lastMessage = messages.last
        val orderBookUpdate = Messages.parse(messages) //
Messages.parse parses a list of messages
        FrontEnd ! orderBookUpdate // FrontEnd is an actor which
receives processed messages
      }
      .map(Messages.ack)

  val webSocketFlow =
Http()(system).webSocketClientFlow(WebSocketRequest("wss://api.bitfin
ex.com/ws/2"))

  val messageSource: Source[Message, Any] =
    Source.single(TextMessage("""{"event":"subscribe",
"channel":"ticker", "symbol":"tBTCUSD"}"""))
      .concat(Source.maybe[Message]) // Keeps the connection open


messageSource.via(webSocketFlow).via(measurementsWebSocket).to(Sink.i
gnore).run()

  log.info("WebSocket connection to Bitfinex API is established")
}
```

3.4. Інтеграція системи баз даних

В цьому проекті Cassandra інтегрована через плагіни Akka Persistence, щоб забезпечити надійне та масштабоване рішення для зберігання стану акторів в Akka Cluster. Конфігурація базується на простоті налаштування з автоматичним створенням просторів ключів та таблиць, і використовує DataStax Java Driver для з'єднання та операцій з базою даних Cassandra.

Сервер Cassandra було запущено на локальній машині, і за допомогою вбудованого плагіна cqlsh була імплементована схема для взаємодії з Akka Persistence(див. Рис. 3.2.).

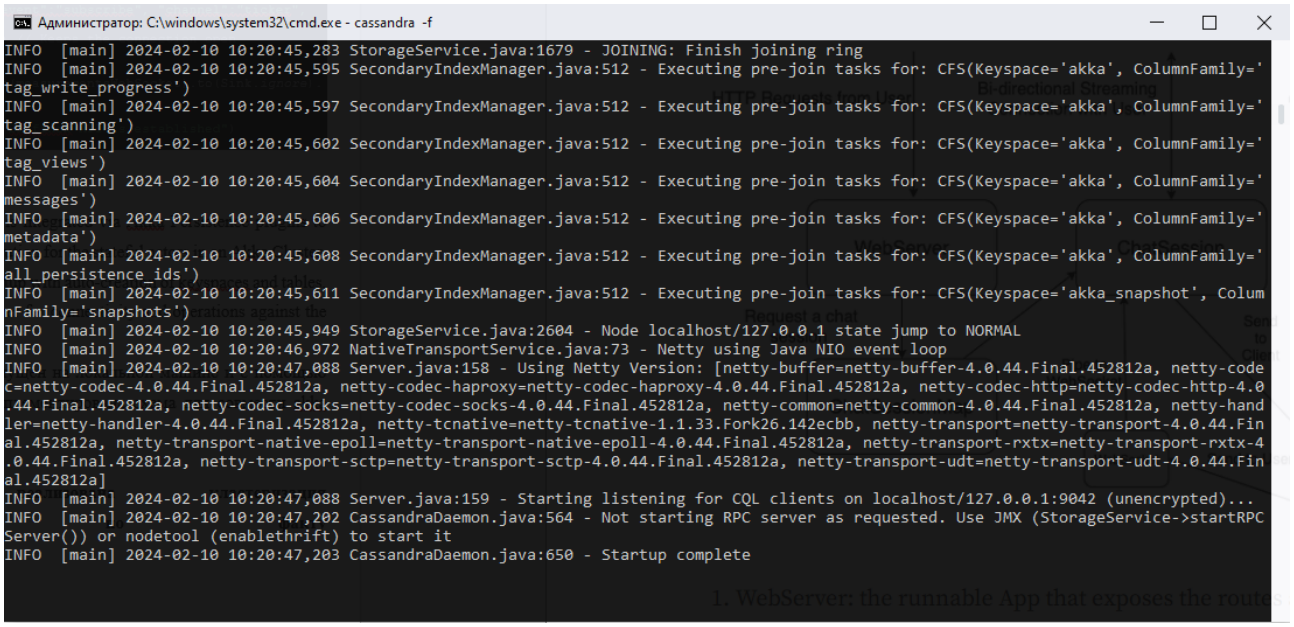

Рис. 3.2 Активний сервер Apache Cassandra

3.5. Імплементація точки входу в програму

```
def main(args: Array[String]): Unit = {
  args.headOption match {

    case None =>
      startClusterInSameJvm()

    case Some(portString) if portString.matches("""\d+""") =>
      val port = portString.toInt
      if (backEndPortRange.contains(port)) start(port, "back-end")
      else if (frontEndPortRange.contains(port)) start(port, "front-end")
      else start(port, "worker", args.lift(1).map(_.toInt).getOrElse(1))

  }
}

def startClusterInSameJvm(): Unit = {
  start(2551, "back-end")
  start(2552, "back-end")
  // two front-end nodes
  start(3000, "front-end")
  start(3001, "front-end")
  // two worker nodes with two worker actors each
```

```
  start(5001, "worker", 2)
  start(5002, "worker", 2)
}
```

У наведеному коді реалізовано запуск кластера з використанням фреймворку Akka. Код визначає точку входу в програму та логіку запуску різних компонентів кластера залежно від аргументів командного рядка або їх відсутності. Усього таким чином, загалом використовується 6 вузлів: Back-end в вузли відповідають за обробку даних, та інші задачі, які не вимагають прямої взаємодії з користувачем або зовнішніми системами. В цьому застосунку створюються два back-end вузли на портах 2551 та 2552.

1. **Front-end вузли**: Ці вузли можуть використовуватися

для інтерфейсу користувача, обробки запитів від користувачів або взаємодії з іншими вузлами та сервісами. В цьомуприкладі створюються два front-end вузли на портах 3000 та 3001.

2. **Worker вузли**: Worker вузли зазвичай використовуються для

виконання фонових задач, таких як обчислення, аналіз даних, або інші задачі, які потребують значних обчислювальних ресурсів. В цьому прикладі створюються два worker вузли на портах 5001 та 5002, причому кожен вузол містить по два актори, що здійснюють обробку задач.

Кожен вузол в кластері може виконувати свої задачі незалежно, але також співпрацювати з іншими вузлами для досягнення загальних цілей системи. Взаємодія між вузлами здійснюється через обмін повідомленнями в рамках моделі акторів, що є основою Akka. Ця система дає уявлення про те як можна ефективно масштабувати систему горизонтально і при необхідності збільшувати обчислювальну потужність та кількість інтерфейсів взаємодії з криптовалютними біржами, додаючи додаткові вузли.

Висновки

В цьому розділі було виконано наступне завдання: Розробка прототипу: "Програмний застосунок електронної торгівлі криптовалютою."

Також було представлено детальний опис розробленого прототипу програмного застосунку для електронної торгівлі криптовалютою. Прототип побудовано на основі модульної реактивної архітектури, використовуючи фреймворк Akka, що дозволяє ефективно масштабувати систему шляхом додавання нових вузлів. Кожен вузол мережі здатен функціонувати автономно, а також взаємодіяти з іншими вузлами через модель акторів Akka, забезпечуючи високу стійкість і продуктивність системи.

Особлива увага в розробці прототипу була приділена інтеграції з API Bitfinex через WebSocket за допомогою методу connectWebSocket(), що забезпечує надійне та оперативне з'єднання з торговельною платформою. Крім того, використання бази даних Cassandra в комбінації з плагінами Akka Persistence дозволяє ефективно зберігати стан акторів, забезпечуючи високу доступність та надійність збереження даних у кластері. Локальний запуск сервера Cassandra сприяє гнучкості та масштабованості системи, роблячи її ідеальною для задач високочастотної торгівлі.

Таким чином, представлений прототип демонструє ключові можливості та переваги обраного підходу до розробки програмного застосунку для електронної торгівлі криптовалютою, підкреслюючи його потенціал для майбутнього розвитку та оптимізації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Soska, K., et al. (2021). Towards Understanding Cryptocurrency Derivatives: A Case Study of BitMEX. *Доступно за посиланням*: https://researchgate.net/Towards_Understanding_Cryptocurrency_Derivatives
2. Zbikowski, K. (2016). Comparative Analysis of Automated Trading Strategies on Bitcoin Market. *Доступно за посиланням*: https://researchgate.net/Comparative_Analysis_of_Automated_Trading_Strategies
3. Alessandretti, L., ElBahrawy, A., Aiello, L. M., & Baronchelli, A. (2018). Anticipating Cryptocurrency Market Movements with Algorithmic Trading. *Доступно за посиланням*: https://researchgate.net/Anticipating_Cryptocurrency_Market_Movements
4. Kinlay, J. (2019). High Frequency Scalping Strategies. Retrieved from https://jonathankinlay.com/2019/10/high-frequency-scalping-strategies/.
5. Huang, B., et al. (2018). Automated Trading Systems: Statistical and Machine Learning Methods and Hardware Implementation: A Survey. *Journal Name*. [Link to the paper if available].
6. Vezeris, D., et al. (2020). Optimization of Backtesting Techniques in Automated High Frequency Trading Systems. *ResearchGate*. [https://www.researchgate.net/publication/326361736_Automated_trading_systems_statistical_and_machine_learning_methods_and_hardware_implementation_a_survey].
7. Grimberg, P., Lauinger, T., & McCoy, D. (2020). Empirical Analysis of Indirect Internal Conversions in Cryptocurrency Exchanges. *ResearhGate*. Retrieved from https://www.researchgate.net/publication/339551066_Empirical_Analysis_of_Indirect_Internal_Conversions_in_Cryptocurrency_Exchanges
8. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., & Juels, A. (2020). Flash Boys 2.0: Frontrunning, Transaction Reordering, and

Consensus Instability in Decentralized Exchanges. *ArXiv*. Retrieved from https://arxiv.org/abs/1904.05234

9. Jazayeriy, H. (2021). Price Action Trading in Cryptocurrency Market. *ResearchGate*. Retrieved from [https://www.researchgate.net/publication/358181973_SPA_Bot_Smart_Price-Action_Trading_Bot_for_Cryptocurrency_Market]
10. Corletto, M. (2021). Impact of Real-World Market Conditions on Returns from Deep Learning-Based Trading Strategies. *ResearchGate*. Retrieved from [https://www.researchgate.net/publication/356135834_Impact_of_Real-World_Market_Conditions_on_Returns_of_Deep_Learning_based_Trading_Strategies]
11. Auer, R. A., & Claessens, S. (2018). Regulating Cryptocurrencies: Assessing Market Reactions. *Bank for International Settlements*. Retrieved from https://www.bis.org/publ/qtrpdf/r_qt1809f.htm
12. Butor. (n.d.). Blackbird Bitcoin Arbitrage. *GitHub*. Retrieved from https://github.com/butor/blackbird
13. Berti, L. (n.d.). Axial-LOB: High Frequency Trading with Axial Attention. *GitHub*. Retrieved from https://github.com/LeonardoBerti00/Axial-LOB-High-Frequency-Trading-with-Axial-Attention
14. Magic8bot. (n.d.). magic8bot. *GitHub*. Retrieved from https://github.com/magic8bot/magic8bot
15. Drakkar-Software. (n.d.). OctoBot. *GitHub*. Retrieved from https://github.com/Drakkar-Software/OctoBot
16. Panifie. (n.d.). PingPong.jl. *GitHub*. Retrieved from https://github.com/panifie/PingPong.jl
17. Wiegers, K. E. (2013). Software Requirements, 3rd edition. *O'Reilly*
18. Lightbend. (n.d.). Akka: Build powerful reactive, concurrent, and distributed applications more easily. *Akka.io*. Retrieved from https://akka.io/

19. Varela, C. (n.d.). Introduction to SALSA 1.1.0. *Worldwide Computing Laboratory*. Retrieved from https://wcl.cs.rpi.edu/salsa/tutorial/salsa-1_1_0/node8.html
20. "Balancing Workload Across Nodes with Akka 2" from Let it crash:Anonymous. (2012, August 8). Balancing Workload Across Nodes with Akka 2. Let it crash. Retrieved from https://letitcrash.com/post/29044669086/balancing-workload-across-nodes-with-akka-2
21. Akka Persistence Cassandra Documentation - Snapshots:Lightbend. (n.d.). Snapshots. Akka Documentation. Retrieved from https://doc.akka.io/docs/akka-persistence-cassandra/current/snapshots.html
22. Akka Persistence Cassandra Documentation - Journal Schema:Lightbend. (n.d.). Journal. Akka Documentation. Retrieved from https://doc.akka.io/docs/akka-persistence-cassandra/current/journal.html#schema
23. Bitfinex homepage: Bitfinex. (n.d.). Bitfinex. Retrieved from https://www.bitfinex.com/
24. Bitfinex Paper Trading article: Bitfinex Support. (n.d.). Paper Trading at Bitfinex: Test, Learn, and Simulate Trading Strategies. Bitfinex Support. Retrieved from https://support.bitfinex.com/hc/en-us/articles/900001525006-Paper-Trading-at-Bitfinex-test-learn-and-simulate-trading-strategies
25. Binance post: Binance. (n.d.). [Best Crypto Scalping Strategies (Profitable) Part 1]. Binance Blog. Retrieved from https://www.binance.com/en/feed/post/973184