НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

“До захисту допущено”
Завідувач кафедри ЦТЕ

_____
Наталія АУШЕВА

«__»_____2025р.

Дипломна робота
на здобуття ступеня бакалавра

За освітньою програмою “Цифрові технології в енергетиці”
Зі спеціальності 122 “Комп’ютерні науки”
на тему: Веб-додаток відстеження біржових акцій з використанням мікросервісної архітектури та хмарних технологій

Виконав:
студент 4 курсу, групи ТР-13

Любченко Денис Миколайович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник: *асистент* каф. *ЦТЕ*
Пасічнюк Антон Олексійович _____
(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові) (підпис)

Рецензент: професор каф. ІПЗЕ,
д.т.н., професор Барабаш Олег Володимирович _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім’я, по батькові) (підпис)

Н.контроль: *асистент* каф. *ЦТЕ*
Волков Олександр Володимирович _____
(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ
Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
Наталія АУШЕВА

«__»_____2025р.

ЗАВДАННЯ
на дипломну роботу студенту

Любченку Денису Миколайовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Веб-додаток відстеження біржових акцій з використанням мікросервісної архітектури та хмарних технологій

Науковий керівник ***Пасічнюк Антон Олексійович***
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом по університету від «02» червня 2025 р. № 1875-с

2. Термін подання роботи студентом 09.06.2025
3. Вихідні дані до роботи: мови програмування Java, JavaScript та TypeScript, фреймворки Spring, мова програмування з фреймворками Spring та React, сервіс керування подіями Apache Kafka, сервіс менеджменту схем Schema Registry, СКБД PostgreSQL, платформа хмарних обчислень AWS.

4. Перелік питань, які потрібно розробити:
- провести аналіз існуючих рішень;
- спроєктувати архітектуру веб-додатку;
- розробити інтерфейс користувача та API;
- реалізувати функціональні модулі для збору, обробки та візуалізації біржової інформації;
- інтегрувати додаток з хмарними сервісами.

5. Орієнтовний перелік графічного (ілюстративного) матеріалу: ілюстрації існуючих аналогів, діаграма архітектури, схема розробленої бази даних, діаграма прецедентів, приклади роботи з інтерфейсом користувача.

6. Дата видачі завдання 13.09.2024.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Затвердження теми роботи		
2	Вивчення та аналіз задачі	14.04-17.04.2025	
3	Аналіз існуючих рішень для відстеження біржових акцій.	18.04-20.04.2025	
4	Проектування архітектури веб-додатка.	21.04-27.04.2025	
5	Розробка інтерфейсу користувача та API.	28.04-01.05.2025	
6	Реалізація функціональних модулів для збору, обробки та візуалізації біржової інформації.	02.05-08.05.2025	
7	Інтеграція з хмарними сервісами для зберігання та обробки даних.	09.05-11.05.2025	
8	Оформлення пояснювальної записки	12.05-18.05.2025	
9	Захист програмного продукту	15.05.2025	
10	Передзахист	28.05.2025	
11	Подання готової роботи на кафедру	02.06-09.06.2025	
12	Захист	16.06-20.06.2025	

Студент _____ **Денис ЛЮБЧЕНКО**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ **Антон ПАСІЧНЮК**
(підпис) (ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 65 сторінках, містить 17 ілюстрацій, 1 додаток, 29 джерел у переліку посилань.

Мета роботи – створення веб-додатку для відстеження біржових акцій з використанням мікросервісної архітектури та хмарних технологій.

Методи та засоби: мікросервісна архітектура, мови програмування Java, JavaScript та TypeScript, фреймворки Spring, мова програмування з фреймворками Spring та React, сервіс керування подіями Apache Kafka, сервіс менеджменту схем Schema Registry, СКБД PostgreSQL, платформа хмарних обчислень AWS.

Результат – реалізований веб-додаток, що дозволяє здійснювати моніторинг, аналіз та візуалізацію фінансових даних фондового ринку в режимі реального часу.

Ключові слова: БІРЖОВІ АКЦІЇ, ВЕБ-ДОДАТОК, МІКРОСЕРВІСНА АРХІТЕКТУРА, ХМАРНІ ТЕХНОЛОГІЇ, SPRING BOOT, KAFKA, POSTGRESQL, REACT.

ABSTRACT

The thesis is presented on 65 pages, includes 17 illustrations, 1 appendix, and 29 references in the bibliography.

The aim of the work is to develop a web application for tracking stock market shares using microservice architecture and cloud technologies.

Methods and tools: microservice architecture; programming languages Java, JavaScript, and TypeScript; Spring and React frameworks; Apache Kafka for event streaming; Schema Registry for schema management; PostgreSQL DBMS; AWS cloud computing platform.

Result – a functional web application that enables real-time monitoring, analysis, and visualization of stock market financial data.

Keywords: STOCK SHARES, WEB APPLICATION, MICROSERVICE ARCHITECTURE, CLOUD TECHNOLOGIES, SPRING BOOT, KAFKA, POSTGRESQL, REACT.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ 8
ВСТУП 9
1 ЗАДАЧА ПОБУДОВИ ВЕБ-ДОДАТКУ ДЛЯ ВІДСТЕЖЕННЯ БІРЖОВИХ АКЦІЙ 11
1.1 Завдання системи відстеження біржових акцій 11
1.2 Огляд існуючих рішень для відстеження біржових акцій 12
1.3 Порівняльний аналіз існуючих веб-додатків відстеження біржових акцій .. 16
2 МІКРОСЕРВІСНА АРХІТЕКТУРА ТА ХМАРНІ ТЕХНОЛОГІЇ 18
2.1. Мікросервісна архітектура 18
2.1.1. Концепція та принципи мікросервісів 18
2.1.2. Використання мікросервісів у веб-додатку відстеження біржових акцій 20
2.2. Чиста та Гексагональна архітектури 21
2.2.1. Принципи Чистої архітектури 21
2.2.2. Застосування Чистої архітектури у серверній частині 22
2.3. Протоколи передачі даних у реальному часі 23
2.3.1. Протокол WebSocket 24
2.3.2. Технологія Server-Sent Events 24
2.3.3. Вибір та реалізація SSE 25
2.4. Компонентний підхід у розробці клієнтської частини 26
2.5. Інтеграція методів та технологій у системі відстеження біржових акцій 28
2.5.1. Подієво-орієнтована архітектура 28
2.5.2. Розгортання та управління мікросервісами в хмарному середовищі 30
3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ 32
3.1 Засоби розробки серверної частини веб-додатку 32
3.2 Засоби розробки користувацького інтерфейсу веб-додатку 34
3.3 Хмарні технології веб-додатку 36

3.4. Діаграма прецедентів 39
3.5. Архітектура системи 41
3.5. Реалізація серверної частини 43
3.6. Схема бази даних 44
3.7. Реалізація клієнтської частини 45
4 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ 47
4.1 Системні вимоги 47
4.2 Приклад роботи з користувацьким інтерфейсом системи 47
ВИСНОВКИ 57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 58
ДОДАТОК А 61

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) – інтерфейс прикладного програмування, набір інструкцій, які дозволяють різним програмам та сервісам взаємодіяти між собою та обмінюватися даними.

REST (Representational State Transfer) – архітектурний стиль створення веб-сервісів, що використовує HTTP-протокол.

СКБД – система керування базами даних.

AWS (Amazon Web Services) – хмарна платформа від Amazon, яка надає широкий спектр сервісів для розробки, розгортання та масштабування додатків.

IAM (Identity and Access Management) – сервіс для управління ідентифікацією та доступом у AWS.

VPC (Virtual Private Cloud) – віртуальна приватна хмара, яка забезпечує ізольоване мережеве середовище у AWS.

MSK (Managed Streaming for Kafka) – керований сервіс потокової передачі даних на основі Apache Kafka.

SSE (Server-Sent Events) – технологія надсилання даних з сервера до клієнта через HTTP.

JSON (JavaScript Object Notation) – текстовий формат обміну даними.

JDBC (Java Database Connectivity) – стандартний інтерфейс Java для доступу до реляційних баз даних.

R2DBC (Reactive Relational Database Connectivity) – реактивний API для роботи з реляційними базами даних.

UI (User Interface) – користувацький інтерфейс.

DOM (Document Object Model) – об’єктна модель документа HTML або XML.

JSX (JavaScript XML) – синтаксичне розширення JavaScript, що дозволяє використовувати HTML-подібний код в React.

FE (Front-End) – клієнтська (фронтенд) частина веб-додатка.

Тикер – коротка назва котируваних інструментів (акцій, облігацій, індексів).

ВСТУП

У сучасному динамічному світі фінансових технологій моніторинг та аналіз біржових акцій набуває дедалі більшого значення для приватних інвесторів, професійних трейдерів та фінансових аналітиків. Постійні зміни на фінансових ринках, зростаюча складність прийняття інвестиційних рішень та необхідність оперативного реагування на ринкові тренди зумовлюють гостру потребу в ефективних інструментах аналізу та моніторингу фінансової інформації в режимі реального часу.

Актуальність розробки веб-додатку для відстеження біржових акцій викликана кількома ключовими факторами. По-перше, існуючі рішення на ринку демонструють значні обмеження у функціональності та гнучкості архітектури, що ускладнює їх адаптацію до швидкозмінних вимог фінансового сектору. По-друге, більшість наявних платформ не забезпечує комплексного підходу до роботи з біржовою інформацією, зосереджуючись лише на окремих аспектах фінансового аналізу. По-третє, обмежена доступність потужних аналітичних інструментів для звичайних користувачів створює бар'єр для входу приватних інвесторів на фінансові ринки. Врешті-решт, недостатня інтеграція сучасних технологій обробки даних у реальному часі знижує ефективність прийняття інвестиційних рішень та реагування на ринкові зміни.

Метою роботи є розробка веб-додатку для моніторингу та аналізу біржових акцій з використанням сучасних підходів побудови розподілених систем та технологій обробки даних у реальному часі, який забезпечує комплексне вирішення завдань фінансового аналізу та підвищує доступність інвестиційних інструментів для широкого кола користувачів.

Для досягнення поставленої мети визначено наступні завдання. Першим завданням є аналіз підходів, методів та алгоритмів розв'язання задач моніторингу та аналізу фінансових даних у режимі реального часу, включаючи дослідження архітектурних патернів розподілених систем та методів ефективної обробки потокових даних. Другим завданням постає аналіз програмних засобів для

реалізації програмної системи моніторингу біржових акцій, що охоплює оцінку технологій серверної та клієнтської розробки, систем управління базами даних та інструментів побудови розподілених архітектур. Третім завданням є розробка програмного забезпечення веб-додатку для відстеження та аналізу біржових акцій з реалізацією функціональності моніторингу цін у реальному часі, візуалізації фінансових показників та інтеграції новинного контенту. Четвертим завданням визначається тестування розробленого програмного забезпечення з метою верифікації коректності реалізації функціональних вимог та оцінки продуктивності системи під навантаженням.

Структура дипломної роботи включає перелік скорочень, умовних позначень і термінів, вступ, чотири розділи основної частини, висновки, список використаних джерел та додатки. У першому розділі розглянуто постановку задачі побудови веб-додатку для відстеження біржових акцій із використанням мікросервісної архітектури та хмарних технологій, визначено призначення та задачі системи, а також здійснено огляд і порівняльний аналіз існуючих рішень. Другий розділ присвячено методам вирішення задачі, зокрема мікросервісній, чистій та гексагональній архітектурам, протоколам передачі даних у реальному часі, підходам до реалізації клієнтської частини, а також інтеграції технологій у межах системи. У третьому розділі наведено опис програмної реалізації, обґрунтовано вибір засобів розробки, подано архітектуру системи, реалізацію серверної та клієнтської частин і структуру бази даних. Четвертий розділ описує взаємодію користувача з програмною системою, її системні вимоги та приклад роботи з інтерфейсом.

1 ЗАДАЧА ПОБУДОВИ ВЕБ-ДОДАТКУ ДЛЯ ВІДСТЕЖЕННЯ БІРЖОВИХ АКЦІЙ

У цьому розділі формулюється задача розроблення веб-додатку для відстеження біржових акцій. Визначено призначення та основні задачі системи. А також представлено порівняння існуючих веб-додатків для відстеження біржових акцій.

1.1 Завдання системи відстеження біржових акцій

Необхідно створити веб-додаток для моніторингу, аналізу та візуалізації даних фондового ринку в режимі реального часу. Система має бути побудована з урахуванням принципів розподіленої архітектури, що забезпечує гнучкість, масштабованість і надійність у роботі з великими обсягами фінансових даних. Інформація про ціни акцій, ринкову капіталізацію, фінансові показники компаній тощо повинна оновлюватися оперативно та бути доступною для кінцевого користувача через зручний інтерфейс.

Особливу увагу слід приділити реалізації інструментів для аналізу ринку, які мають бути інтуїтивно зрозумілими та функціональними. Система повинна відображати зміни цін та інші фінансові показники за допомогою графічного інтерфейсу.

Цільовою аудиторією є приватні інвестори, професійні трейдери, фінансові аналітики, а також представники освітніх установ, які можуть використовувати систему як навчальний інструмент із фінансової грамотності та ринкового аналізу.

Основною задачею є розробка системи, здатної здійснювати постійний моніторинг акцій у режимі реального часу, забезпечуючи доступ до історичних даних, візуалізацію фінансових показників та можливість перегляду новин, що стосуються обраних акцій.

Архітектура системи має бути побудована на основі модульного підходу, що передбачає створення окремих компонентів для зчитування та передавання біржових даних, їх зберігання й обробки, а також надання програмного інтерфейсу для взаємодії з користувацьким інтерфейсом. Завдяки незалежності кожного компонента система має бути здатною до гнучкої модифікації й оновлення без порушення загальної працездатності.

Окрема увага приділяється забезпеченню простоти інтеграції з новими джерелами фінансових даних та зовнішніми інтерфейсами. Для зберігання даних повинно бути використано рішення, що забезпечує високу продуктивність при обробці великих обсягів інформації, надійність зберігання та швидкий доступ до історичних даних. Система управління форматами даних має забезпечити узгодженість структур інформації та коректну взаємодію між окремими модулями.

Для забезпечення обробки потоків даних в режимі реального часу необхідно застосувати рішення, яке підтримує високу пропускну здатність, горизонтальне масштабування та гарантовану доставку повідомлень. Система повинна ефективно обробляти як періодичні оновлення цін, так і масові потоки даних під час активних торгових сесій.

Реалізована система повинна надавати широкі аналітичні можливості – перегляд фінансової статистики, графіки змін цін, інформацію про торговельні обсяги, а також налаштування параметрів виводу та вибір часових періодів для аналізу. Важливо забезпечити функціонал для перегляду новин, що дозволить користувачам своєчасно реагувати на важливі події, які можуть впливати на динаміку акцій.

1.2 Огляд існуючих рішень для відстеження біржових акцій

Огляд Stock Alarm

Stock Alarm – це веб-додаток, який дозволяє користувачам встановлювати сповіщення про зміни цін акцій та отримувати їх через push-повідомлення,

електронну пошту, SMS або дзвінки. Додаток підтримує понад 65 000 активів по всьому світу.

Функціонал сайту включає моніторинг акцій та ETF з відображенням графіків цін у реальному часі, систему сповіщень для встановлення нагадувань при досягненні заданих цін на певні акції, детальні інтерактивні графіки з можливістю вибору різних часових періодів, таблицю основних індексів NASDAQ, S&P 500, Dow Jones з поточними значеннями та відсотковими змінами, а також систему реєстрації та авторизації користувачів. Інтерфейс веб-додатку представлено на рисунку 1.1.

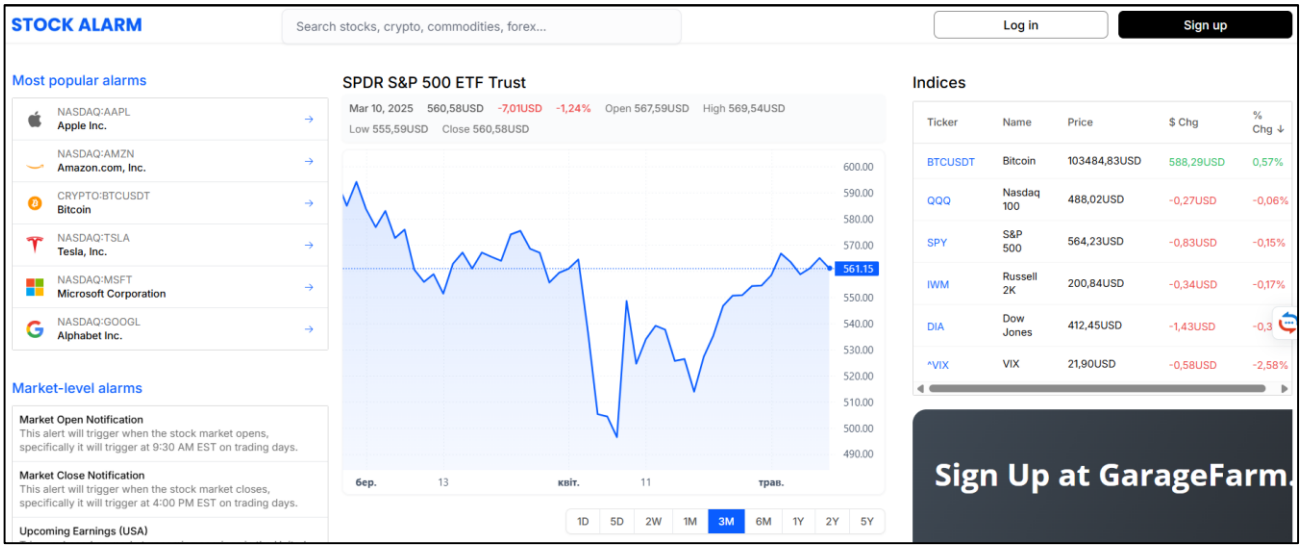

Рисунок 1.1 – Графік біржових акцій Stock Alarm

Переваги:

- широке покриття ринків та фінансових інструментів по всьому світу;
- детальна фінансова інформація про компанії;
- великий обсяг аналітичних матеріалів та експертних оцінок;
- можливість встановлення нагадувань при зміні ціни.

Недоліки:

- відсутність стрічки новин;
- відсутність АПІ для розробників.

Огляд Snowball Analytics

Snowball Analytics – це сучасний веб-додаток, призначений для відстеження та аналізу інвестиційних портфелів. Його основний фокус – допомогти інвесторам отримати чітке уявлення про ефективність їхніх інвестицій, диверсифікацію, доходи від дивідендів та загальний фінансовий прогрес. Також дозволяє аналізувати розподіл активів, історичну дохідність, прогнозувати дивідендні потоки та порівнювати ефективність з ринковими індексами. Інтерфейс веб-додатку представлено на рисунку 1.2.

Рисунок 1.2 – Графік біржових акцій Snowball Analytics

Функціонал сайту включає детальний аналіз цінової історії акцій з інтерактивними графіками, що дозволяють переглядати динаміку цін за різні періоди від одного дня до п'яти років, панель з ключовими фінансовими показниками включаючи поточну ціну, оцінки аналітиків, прогнози росту та дохідності, відображення темпів зростання за різні періоди з відсотковими змінами, прогнозування майбутньої вартості акцій на основі аналітичних моделей, інформацію про дивідендну політику компанії з історією виплат, а також інструменти для порівняння показників ефективності інвестицій.

Переваги:

- підтримка різних класів активів.

Недоліки:

- відсутність налаштувань вигляду графіку;
- неможливо виміряти зміну ціну на конкретному часовому проміжку;
- неоптимізований інтерфейс, що споживає багато ресурсів.

Огляд Simple Portfolio

Simple Portfolio – це базовий веб-додаток для відстеження інвестиційного портфеля, орієнтований на починаючих інвесторів. Платформа забезпечує прості можливості для моніторингу вартості портфеля, відстеження основних даних про акції та отримання базової аналітики. Додаток дозволяє користувачам створювати віртуальні портфелі, додавати акції та відстежувати їхню загальну продуктивність з плином часу. Інтерфейс веб-додатку представлено на рисунку 1.3.

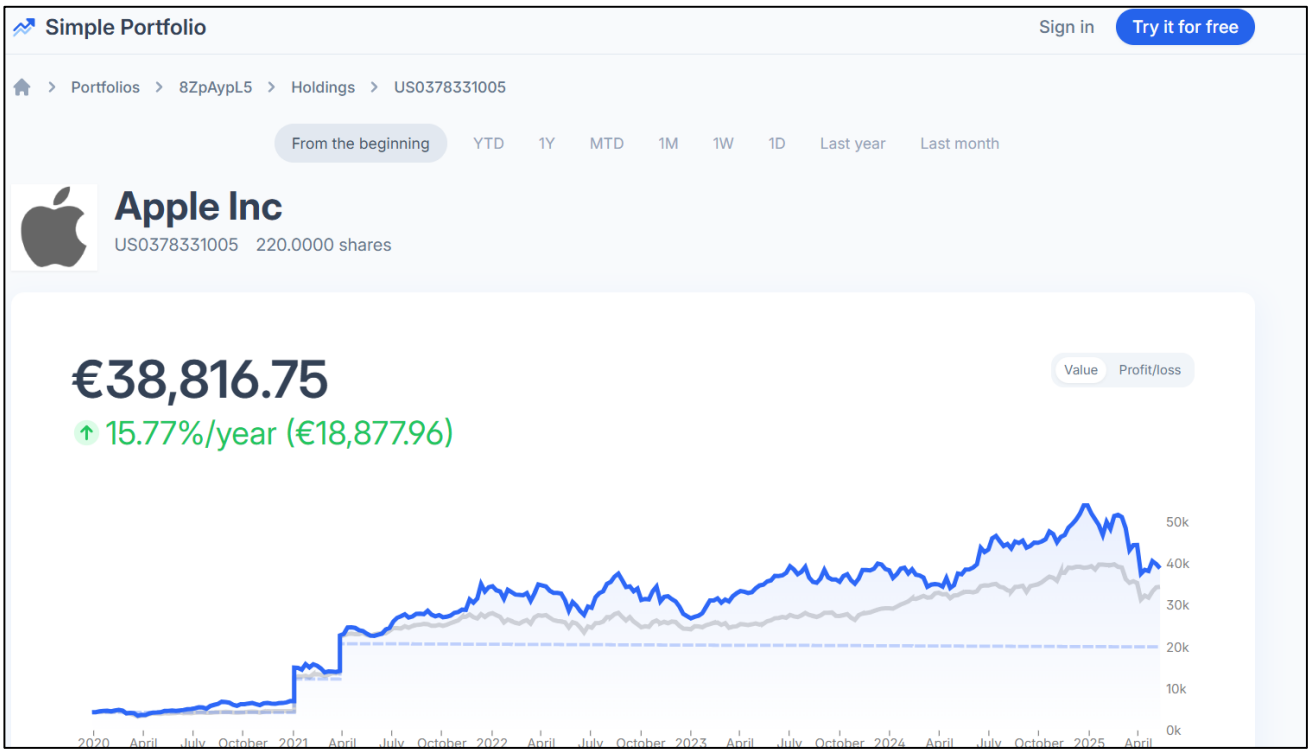

Рисунок 1.3 – Графік біржових акцій Simple Portfolio

Функціонал сайту включає управління інвестиційним портфелем з можливістю відстеження власних активів, детальний аналіз окремих позицій

портфеля з відображенням кількості акцій та їх поточної вартості, показники прибутковості з відсотковими та абсолютними значеннями доходності за рік, інтерактивні графіки ефективності портфеля з можливістю перегляду динаміки за різні періоди від початку інвестування до поточного моменту, а також можливість порівняння ефективності інвестицій з ринковими індексами.

Переваги:

- легкий у використанні та інтуїтивно зрозумілий для початківців інтерфейс;
- можливість налаштування власного портфоліо.

Недоліки:

- відсутність новин та галузевої інформації;
- неможливість детально налаштовувати графік акцій.

1.3 Порівняльний аналіз існуючих веб-додатків відстеження біржових акцій

Аналіз сучасного ринку систем моніторингу біржових акцій виявляє деякі прогалини у функціональності існуючих рішень, які повинна усунути розроблювана система. Більшість наявних платформ спеціалізується на окремих аспектах фінансового аналізу, не забезпечуючи комплексного підходу до роботи з біржовою інформацією.

Існуючі системи, такі як Snowball Analytics та Simple Portfolio, демонструють базові можливості відстеження цін акцій, проте мають суттєві обмеження у функціональності візуалізації. Недостатньою є можливість гнучкого налаштування графіків, обмежені опції масштабування та значна складність інтерфейсу. Ці недоліки ускладнюють проведення детального технічного аналізу та знижують ефективність прийняття інвестиційних рішень.

Особливо критичною є відсутність у більшості рішень спеціалізованих інструментів для точного вимірювання цінових змін. Аналітики та трейдери потребують можливості точного обчислення різниці цін на конкретних часових

проміжках, що дозволило б оцінювати волатильність та прогнозувати потенційні зміни. Розроблювана система має включати інструмент "Лінійка" для забезпечення таких обчислень з високою точністю.

Інтеграція новинного контенту є ще однією слабкою стороною наявних рішень. Більшість систем або повністю ігнорують інформаційний фон, або надають новини без прив'язки до конкретних акцій. Розроблювана платформа має включати стрічку релевантних новин, що дозволить користувачам оцінювати вплив інформаційних подій на динаміку цін та приймати більш обґрунтовані рішення.

Архітектурні обмеження існуючих рішень ускладнюють їх інтеграцію з іншими системами та розширення функціональності. Розроблювана платформа має забезпечити відкритий програмний інтерфейс для розробників, що дозволить інтегрувати її функціонал у власні системи та додатки. Модульна архітектура повинна передбачати легке розширення функціоналу та адаптацію до нових ринкових вимог, забезпечуючи довгострокову актуальність рішення.

Таким чином, розроблювана система має заповнити критичні прогалини ринку, забезпечивши комплексний підхід до моніторингу та аналізу біржових акцій, підвищену доступність аналітичних інструментів та гнучкість архітектури для майбутнього розвитку.

2 МІКРОСЕРВІСНА АРХІТЕКТУРА ТА ХМАРНІ ТЕХНОЛОГІЇ

Цей розділ присвячений детальному аналізу архітектурних підходів, методів та алгоритмів, що були використані при розробці веб-додатку для відстеження біржових акцій.

2.1. Мікросервісна архітектура

Цей підрозділ присвячений детальному розгляду мікросервісної архітектури як фундаментального підходу до побудови розподілених систем, зокрема веб-додатку для відстеження біржових акцій. А також проаналізовані її ключові концепції, принципи, а також обґрунтований вибір даної архітектури для реалізації системи.

2.1.1. Концепція та принципи мікросервісів

Мікросервісна архітектура є сучасним підходом до розробки програмного забезпечення, при якому додаток будується як набір невеликих, незалежних сервісів, що взаємодіють між собою. Кожен сервіс сфокусований на виконанні однієї чітко визначеної бізнес-функції, є слабозв’язаним та може бути розгорнутий, масштабований і керований незалежно від інших [1].

Ключові характеристики мікросервісів включають їх незалежність розгортання, що дозволяє розгортати кожен мікросервіс самостійно, без необхідності перезапуску всього додатка. Вони також відзначаються високою масштабованістю, оскільки можуть бути масштабовані горизонтально шляхом додавання нових екземплярів окремих сервісів, які потребують більшої обчислювальної потужності. Це дозволяє системі ефективно реагувати на зростання навантаження, наприклад, на сплески трафіку на біржовому ринку.

Важливою характеристикою є відмовостійкість. Завдяки слабкій зв’язаності між компонентами, збій в одному мікросервісі не призводить до відмови всього додатку. Проблеми локалізуються в межах одного сервісу, що підвищує загальну стійкість системи до відмов [1].

Крім того, мікросервіси демонструють технологічну агностичність, дозволяючи кожній команді обирати найбільш відповідні технології (мови програмування, фреймворки, бази даних) для конкретного сервісу, на відміну від традиційних монолітних додатків, які зазвичай використовують єдиний технологічний стек. Врешті, мікросервіси є розподіленими системами, що передбачає комунікацію між ними через мережу, яка може бути як синхронною (наприклад REST), так і асинхронною (наприклад через брокери повідомлень Apache Kafka) [1].

Мікросервіси також забезпечують масштабованість за потребою, дозволяючи горизонтальне масштабування без зайвих зусиль, що є ідеальним для складних програмних систем, де масштабованість є пріоритетом. Розробка додатків у мікросервісному стилі полегшує реагування на сплески трафіку та допомагає підтримувати загальну стабільність системи. Компоненти є невеликими та модульними, тому їх можна розгортати набагато швидше, ніж у монолітній архітектурі, що є значною перевагою для продуктів, які потребують проактивного реагування на зміни ринку [1].

Однак, мікросервісний підхід має і певні недоліки. Він збільшує початкову складність. Управління даними стає складнішим, оскільки мікросервіси зазвичай підтримують власні бази даних або сховища даних, що може призвести до дублювання даних та неузгодженості, вимагаючи ретельного планування та стратегій для забезпечення цілісності даних [1].

Крім того, виникають проблеми безпеки та затримки, оскільки комунікація між сервісами відбувається через мережу, що може призвести до збільшення затримки та потенційних проблем безпеки, які необхідно враховувати та вирішувати [1].

2.1.2. Використання мікросервісів у веб-додатку відстеження біржових акцій

Відстеження біржових акцій у реальному часі передбачає постійний потік даних з високою частотою оновлень. Монолітна архітектура зіткнулася б зі значними труднощами при незалежному масштабуванні різних функціональних частин, таких як збір даних, їх обробка та надання інтерфейсу користувача. Мікросервіси дозволяють кожному спеціалізованому сервісу (продюсеру, споживачу, користувацькому інтерфейсу) масштабуватися відповідно до його специфічного навантаження. Це запобігає виникненню вузьких місць у продуктивності та забезпечує оперативність для даних у реальному часі, що є критично важливим для біржового ринку.

Розроблена система моніторингу фондового ринку реалізована на основі подієво-орієнтованої архітектури з використанням мікросервісного підходу. Це передбачає чітке розділення відповідальностей між трьома різними компонентами. Сервіс Stock Market Producer відповідає за збір актуальних даних з фондових ринків через зовнішні API. Отримавши інформацію, він публікує ці дані у центральну шину повідомлень Apache Kafka для подальшої обробки. Сервіс Stock Data Consumer споживає потокові дані з Kafka, виконує необхідну обробку та зберігає їх у реляційній базі даних PostgreSQL. Він також може надавати API для взаємодії з користувацьким інтерфейсом. Компонент Stock Chart Frontend є клієнтською частиною, яка відповідає за візуалізацію даних для кінцевих користувачів. Він надає інтерактивний інтерфейс для перегляду графіків цін акцій, пошуку акцій та отримання детальної інформації про компанії.

Такий розподіл функціоналу на окремі, незалежні сервіси є ключовим для забезпечення стійкості системи. У мікросервісній архітектурі сервіси є слабозв’язаними. Якщо, наприклад, Stock Market Producer тимчасово виходить з ладу, кластер Kafka може буферизувати події, дозволяючи Stock Data Consumer продовжувати обробку історичних даних або наздогнати їх після відновлення продюсера. Ця ізоляція відмов є критично важливою для фінансового додатку, де безперервність даних має першочергове значення. Подієво-орієнтована природа

системи додатково посилює цю перевагу, надаючи асинхронну комунікаційну основу, що дозволяє компонентам працювати незалежно та реагувати на події, а не на прямі запити.

2.2. Чиста та Гексагональна архітектури

Цей підрозділ детально розкриває принципи Чистої та Гексагональної архітектур, які є ключовими для забезпечення модульності та гнучкості серверної частини веб-додатку.

2.2.1. Принципи Чистої архітектури

Чиста Архітектура (Clean Architecture) є набором керівних принципів, спрямованих на організацію залежностей у системі таким чином, щоб сприяти гнучкості та легкості підтримки програмного забезпечення. Її основна ідея полягає у відокремленні бізнес-логіки та доменного коду від інфраструктури, мінімізуючи вплив змін у вимогах або технологіях [2].

В основі Чистої архітектури лежить ідея поділу системи на шари, де кожен шар має чітко визначені відповідальності та залежності. Найважливішим правилом є правило Залежностей: залежності завжди повинні вказувати всередину, до основної бізнес-логіки. Це означає, що внутрішні шари не повинні знати про деталі зовнішніх шарів (бази даних, UI, зовнішні API). Залежність від зовнішніх компонентів досягається за допомогою інтерфейсів та абстрактних класів, що розв’язують бізнес-логіку від конкретних технічних деталей [2].

Чиста Архітектура надає низку переваг. Вона полегшує розробку, оскільки кодова база стає легшою для підтримки завдяки чітко розділеним задачами та модульному дизайну. Зміни в одній частині системи мають мінімальний вплив на інші. Тестування також стає легшим, адже ізоляція бізнес-логіки від інших частин системи дозволяє легко писати юніт-тести без залежності від інфраструктури, що

робить тестовий код простішим та швидшим у виконанні. Архітектура підвищує гнучкість та повторне використання, оскільки поділена на незалежні модулі, що спрощує повторне використання конкретних компонентів або функцій в інших проєктах. Зміна технологічних стеків може бути здійснена без впливу на інші шари системи. Крім того, забезпечується незалежність від технічних деталей, оскільки бізнес-логіка залишається незалежною від конкретних реалізацій баз даних, UI або зовнішніх API. Це дозволяє відкладати рішення щодо вибору конкретних технологій до пізніших етапів розробки. Нарешті, Чиста Архітектура забезпечує чітку структуру завдяки чіткому розділенню задач, що полегшує новим членам команди розуміння коду та навігацію по ньому [2].

Однак, існують і недоліки. Впровадження Чистої Архітектури збільшує початкові витрати на розробку, оскільки вимагає значної попередньої роботи з проєктування, включаючи визначення шарів, інтерфейсів та розділення відповідальностей. Загальна складність проєкту може зрости з кількома залученими шарами [2].

2.2.2. Застосування Чистої архітектури у серверній частині

Серверна частина системи відстеження біржових акцій реалізована на мові програмування Java з використанням фреймворку Spring Boot.

Код у шари, що відповідають принципу залежності. Внутрішній шар – це сутності, що містить фундаментальні бізнес-правила та доменні об’єкти. Ці сутності є чистими Java-об’єктами і не повинні залежати від будь-яких зовнішніх фреймворків чи деталей інфраструктури. У системі відстеження акцій це можуть бути сутності, що представляють основні атрибути та поведінку біржових даних, такі як Stock, Company, Trade та BasicFinancials.

Наступний шар – сервіси, що містить специфічні для додатку бізнес-логіку. Варіанти використання оркеструють потік даних до та від сутностей, визначаючи, як система автоматизує свої операції. Вони також не залежать від зовнішніх деталей, таких як бази даних або UI.

Шар адаптерів відповідає за перетворення даних між форматом, зручним для бізнес-логіки, та форматом зручним для зовнішніх сервісів таких як база даних Postgres, Kafka та зовнішні API. До них належать адаптери даних, які реалізують з'єднання для взаємодії з базою даних.

Для передачі оновлень цін акцій від сервера до клієнта використовується реалізація Server-Sent Events у Spring WebFlux, що також є адаптером, який перетворює внутрішні події системи на потік даних, придатний для трансляції через HTTP.

Забезпечення розв'язаності досягається за допомогою Принципу Інверсії залежностей. Внутрішні шари залежать від абстракцій, а не від конкретних реалізацій, які знаходяться у зовнішніх шарах. Ізолюючи основні бізнес-правила в доменному шарі та шарі варіантів використання, зміни в джерелах даних або механізмах збереження можуть бути зроблені шляхом простої зміни адаптерів, не впливаючи на основну логіку.

Таким чином, шар сутностей інкапсулює основні бізнес-правила, представляючи такі об'єкти, як акції, компанії та торгові операції, і є незалежним від фреймворків та інфраструктури. Шар сервісів використання містить логіку додатку, обробляючи запити на отримання даних, аналіз фінансових показників та збереження даних, і залежить від абстракцій через інверсію залежностей. Адаптери інтерфейсів перетворюють дані для взаємодії між внутрішніми та зовнішніми шарами, включаючи REST контролери для API та репозиторії для баз даних, адаптуючись до зовнішнього світу. Імплементації адаптерів до REST контроллера та консюмера представлено в додатку А.

2.3. Протоколи передачі даних у реальному часі

Цей підрозділ аналізує протоколи передачі даних у реальному часі, WebSocket та Server-Sent Events, з акцентом на їх застосування у веб-додатку для відстеження біржових акцій.

2.3.1. Протокол WebSocket

WebSocket — це комунікаційний протокол, що забезпечує повнодуплексний, двосторонній зв’язок між клієнтом і сервером через одне, довготривале TCP-з’єднання. На відміну від традиційного HTTP, де кожен запит вимагає нового з’єднання, WebSocket підтримує постійне з’єднання, дозволяючи як клієнту, так і серверу надсилати повідомлення в будь-який час без необхідності повторного встановлення зв’язку. Це значно зменшує накладні витрати, пов’язані з встановленням з’єднання та передачею заголовків [3].

WebSocket має значні переваги, такі як низька затримка та висока продуктивність завдяки постійному з’єднанню та мінімізації заголовків, що особливо корисно для чатів та оновлень даних у реальному часі. Двостороння комунікація дозволяє як клієнту, так і серверу надсилати дані в будь-який час, що робить його ідеальним для інтерактивних додатків. Протокол також підтримує передачу як текстових, так і бінарних даних, розширюючи його застосування для потокової передачі медіа [3].

Проте, WebSocket є більш складним у реалізації порівняно з SSE, вимагаючи більше ресурсів та складнішої логіки для управління з’єднаннями. Також відсутня вбудована підтримка автоматичного перепідключення при втраті з’єднання [3].

2.3.2. Технологія Server-Sent Events

Server-Sent Events (SSE) — це легка технологія, яка дозволяє серверам надсилати оновлення в реальному часі веб-клієнтам через одне, довготривале HTTP-з’єднання. На відміну від WebSockets, SSE забезпечує лише односторонній потік даних — від сервера до клієнта. Сервер постійно надсилає дані клієнту, а клієнт отримує ці події та може відповідно реагувати на них. SSE працює поверх стандартного HTTP-протоколу, що робить його простішим у використанні та інтеграції [4].

Серед переваг SSE варто відзначити простоту реалізації, оскільки вони використовують стандартний протокол HTTP, що дозволяє розробникам швидко

інтегрувати функціональність реального часу. Важливою особливістю є вбудоване автоматичне перепідключення: у випадку втрати з’єднання SSE автоматично намагається відновити підключення до сервера, що значно спрощує розробку клієнтської частини та підвищує надійність. SSE ідеально підходять для односторонніх оновлень, оскільки технологія розроблена спеціально для сценаріїв, де потрібні лише оновлення від сервера до клієнта [4].

Однак, SSE мають і певні недоліки. Їх основне обмеження – формат даних. SSE можуть надсилати лише текстові дані (UTF-8), не підтримуючи бінарні дані. Браузери можуть накладати обмеження на кількість одночасних SSE-з’єднань, що може бути проблемою для додатків, які відкривають багато вкладок з SSE [4].

2.3.3. Вибір та реалізація SSE

Для веб-додатку відстеження біржових акцій основним завданням є оперативна трансляція змін цін та іншої фінансової інформації від сервера до клієнта. Це є типовим сценарієм односторонньої передачі даних.

Вибір SSE замість WebSockets був обґрунтований його простотою реалізації та вбудованою функціональністю автоматичного перепідключення. Хоча WebSockets пропонують двосторонню комунікацію, оновлення цін акцій за своєю природою є односпрямованими. SSE простіші в реалізації для цього конкретного випадку використання та забезпечують вбудоване автоматичне перепідключення, що зменшує складність на стороні клієнта. Це дозволяє мінімізувати затримки при оновленні даних на клієнтській стороні та забезпечити користувачів актуальною інформацією.

Проте, клієнт WebSocket реалізований у модулі продюсера для зчитування змін у цінах біржових акцій. Так як на сьогодні більшість сервісів з надання брокерської інформації використовують саме WebSocket з’єднання для трансляції біржових даних у реальному часі.

Реалізація SSE на серверній частині використовує реактивний веб-фреймворк Spring WebFlux для неблокуючої моделі обробки запитів. Це є критично важливим

для систем, які потребують ефективної обробки паралельних з’єднань, як у випадку з біржовим додатком, що підтримує багато одночасних клієнтів. Flux є реактивним потоком даних, який дозволяє безперервно надсилати події клієнту. Дані з зовнішнього API біржових даних зчитуються в реальному часі продюсером, приводяться до вигляду, зазначеного в контракті Avro, та надсилаються в Kafka. Потім ці дані споживаються з Kafka та через SSE передаються на фронтенд. Реактивне програмування зі Spring WebFlux є ключовим для ефективної обробки SSE-потоків на сервері, особливо враховуючи високий обсяг даних біржового ринку в реальному часі. SSE підтримує постійне з’єднання, і для біржового додатку це означає потенційно багато одночасних з’єднань, що надсилають безперервні оновлення. Традиційне блокуюче введення/виведення швидко стало б вузьким місцем. Неблокуюча, асинхронна природа Spring WebFlux дозволяє серверу обробляти велику кількість одночасних SSE-з’єднань з меншою кількістю потоків, забезпечуючи високу продуктивність та чутливість для потокової передачі даних у реальному часі без вичерпання ресурсів сервера.

Використання EventSource API на клієнтській частині передбачає, що фронтенд-компонент використовує вбудований у браузер EventSource API для встановлення з’єднання з SSE-ендпоінтом сервера та обробки потоку подій. useEffect хук React використовується для ініціалізації EventSource та підписки на події onmessage, onopen та onerror. При отриманні нових даних стан додатку оновлюється через Redux , що автоматично призводить до перемальовування відповідних компонентів інтерфейсу. Для забезпечення стійкості з’єднання реалізовано логіку автоматичного перепідключення, що дозволяє клієнту спробувати відновити з’єднання після тимчасових збоїв мережі.

2.4. Компонентний підхід у розробці клієнтської частини

Основна ідея компонентно-орієнтованої архітектури в React базується на принципі поділу інтерфейсу користувача на менші, самодостатні та

взаємопов’язані сегменти. Кожен компонент ізольовано аналізується та розробляється, маючи єдину відповідальність за свій потік даних та зовнішній вигляд. Це дозволяє створювати повторно використовувані UI-компоненти та ефективно керувати станом додатку.

Ключові принципи компонентного дизайну включають модульність, повторне використання та гнучкість. Модульність означає, що компоненти повинні бути одноцільовими, легко керованими та повторно використовуваними. Це спрощує побудову та підтримку складних інтерфейсів, дозволяючи розробникам ефективно організовувати код та запобігати дублюванню. Повторне використання передбачає створення компонентів, які можна використовувати в різних частинах додатку або навіть у різних проєктах, забезпечуючи послідовність та прискорюючи розробку. Гнучкість компонентів досягається завдяки їх проєктуванню таким чином, щоб їх можна було легко налаштовувати для різних випадків використання.

Props (properties) дозволяють передавати дані та функції зворотного виклику від батьківських компонентів до дочірніх, роблячи компоненти гнучкими та легкими у налаштуванні. Це дозволяє змінювати поведінку та зовнішній вигляд компонента без зміни його внутрішньої логіки. Найкращі практики використання props включають:

- використання чітких, описових імен, щоб зробити їх зрозумілими;

- встановлення значень за замовчуванням для props, щоб компоненти коректно працювали, навіть якщо props не надані;

- використання перевірки типів за допомогою TypeScript для раннього виявлення помилок.

Разом це підвищує надійність коду, особливо у складному додатку, що працює з різноманітними структурами фінансових даних.

2.5. Інтеграція методів та технологій у системі відстеження біржових акцій

В цьому розділі йдеться про інтеграіцю системи з хмарними технологіями. Зокрема про подієво-орієнтовану архітектуру з брокером повідомлень та Schema Registry та розгортання системи на системі хмарних обчислень.

2.5.1. Подієво-орієнтована архітектура

Центральним елементом архітектури розробленої системи моніторингу фондового ринку є Apache Kafka, яка виступає в ролі розподіленого журналу подій. Kafka забезпечує високу пропускну здатність, стійкість до відмов та надійний обмін повідомленнями між різними компонентами системи. Це є критично важливим для обробки значних обсягів біржових даних у реальному часі, де оперативність отримання інформації про зміни цін акцій має першочергове значення.

Kafka організує дані через теми (topics), які можна розділяти на партиції (partitions) для паралельної обробки, що уможливлює горизонтальне масштабування. Розподілення тем на партиції дозволяє розподіляти навантаження між кількома брокерами, забезпечуючи високу пропускну здатність та паралельну обробку даних. Система зберігає повідомлення на диску протягом налаштованого періоду, забезпечуючи відтворення історичних даних. Kafka також забезпечує стійкість до відмов через механізми реплікації даних між кількома брокерами. Кожна партиція має лідера та послідовників, що гарантує збереження даних навіть у випадку відмови окремих вузлів кластера. Для серіалізації та десеріалізації повідомлень у Kafka використовується формат Avro у поєднанні з Confluent Schema Registry. Приклад контракту-схеми з вказанням назв обов’язкових полів та їх короткого опису зображено на рисунку 2.1.

Взаємодія Producer та Consumer сервісів з Kafka відбувається наступним чином. Stock Market Producer відповідає за збір даних з фондових ринків через

зовнішні API. Дані з зовнішнього API через вебсокети зчитуються у реальному часі продюсером, приводяться до вигляду, зазначеного у контракті Avro, та надсилаються у Kafka. Stock Data Consumer споживає дані з Kafka, обробляє їх та зберігає в базі даних PostgreSQL. Для цього використовується Spring Kafka, що надає зручний інтерфейс для обробки повідомлень. Для забезпечення масштабованості та ефективної обробки даних використовується підхід з групами споживачів, що дозволяє розподілити навантаження між кількома екземплярами сервісу за необхідності.

```
{
  "type": "record",
  "name": "Trade",
  "namespace": "edu.kpi.diploma.schema",
  "doc": "Represents a trade event containing stock symbol, price, volume, and timestamp.",
  "fields": [
    {
      "name": "symbol",
      "type": "string",
      "doc": "Stock symbol associated with the trade."
    },
    {
      "name": "price",
      "type": "double",
      "doc": "Trade price of the stock."
    },
    {
      "name": "volume",
      "type": "double",
      "doc": "Trade volume in terms of stock units."
    },
    {
      "name": "timestamp",
      "type": "string",
      "doc": "Timestamp of the trade event in date-time format without a time-zone in the ISO-8601 calendar system."
    }
  ]
}
```

Рисунок 2.1 – Avro схема для сутності Trade

Kafka та Schema Registry є основоположними для можливостей системи в реальному часі та цілісності даних у розподіленому середовищі. Дані біржового ринку в реальному часі вимагають безперервного, високооб’ємного прийому та обробки. Розподілена, розділена на партиції та реплікована природа Kafka гарантує, що навіть при сплесках ринкової активності дані не будуть втрачені та можуть бути оброблені з низькою затримкою. Schema Registry з Avro забезпечує,

що, коли структура даних еволюціонує (наприклад, додаються нові фінансові показники), продюсери та споживачі можуть залишатися сумісними, запобігаючи пошкодженню даних та забезпечуючи безперебійні оновлення аналітичних можливостей системи без простоїв. Це критично важливий фактор для основної функції системи.

2.5.2. Розгортання та управління мікросервісами в хмарному середовищі

Розгортання та управління мікросервісами в хмарному середовищі є ключовим для забезпечення масштабованості, надійності та безпеки системи відстеження біржових акцій. Використання керованих сервісів AWS значно спрощує ці процеси.

Для автоматизованого розгортання та масштабування серверів використовується AWS Elastic Beanstalk — це повністю керований сервіс PaaS (Platform as a Service), який дозволяє розробникам легко розгортати та масштабувати веб-застосунки. Він автоматично обробляє надання ресурсів, балансування навантаження, автоматичне масштабування та моніторинг стану програми. Elastic Beanstalk підтримує розгортання Docker-контейнерів, що спрощує розгортання мікросервісів. Через AWS Console або CLI можна налаштовувати тип інстансів, параметри масштабування та перевірки.

Роль AWS IAM, VPC та Security Groups для безпеки та мережевої ізоляції є критично важливою. AWS IAM дозволяє централізовано керувати користувачами, групами безпеки та ролями, а також встановлювати детальні дозволи для контролю доступу до сервісів та ресурсів AWS. Це забезпечує, що лише авторизовані користувачі та сервіси можуть взаємодіяти з компонентами системи. Amazon VPC дозволяє створювати ізольовану секцію хмари AWS, де можна запускати ресурси у визначеній віртуальній мережі. У VPC можна повністю контролювати віртуальне мережеве середовище, включаючи вибір власного діапазону IP-адрес, створення підмереж, налаштування таблиць маршрутизації та мережевих шлюзів. Це забезпечує можливість створення різних мережевих топологій (публічні, приватні

та гібридні мережі). AWS Security Groups діють як віртуальний міжмережевий екран для ресурсів AWS, контролюючи вхідний і вихідний трафік. Вони дозволяють налаштовувати правила, які визначають, який мережевий трафік може досягти та залишити підключені ресурси, на основі протоколу, порту та джерела/призначення. Найкращі практики включають обмеження вхідного та вихідного доступу та мінімізацію діапазонів портів, що є критично важливим для безпеки мікросервісів. Дозволяючи лише необхідні порти та джерела/призначення, можна запобігти несанкціонованому доступу та зменшити поверхню атаки.

Amazon MSK (Managed Streaming for Kafka) — це повністю керований сервіс, що полегшує створення, оновлення та керування кластерами Apache Kafka у хмарі. MSK автоматично надає та налаштовує кластери, постійно відстежує їх стан, автоматично замінює несправні вузли та спрощує налаштування безпеки з шифруванням у стані спокою та під час передачі.

Комплексне використання сервісів AWS (Elastic Beanstalk, IAM, VPC, Security Groups, MSK) забезпечує безпечне, масштабоване та операційно ефективне середовище для мікросервісів. Розгортання мікросервісів вносить складнощі розподілених систем. AWS Elastic Beanstalk спрощує це, автоматизуючи управління інфраструктурою. VPC та Security Groups є критично важливими для мережевої ізоляції та безпеки між мікросервісами та зовнішньою комунікацією, запобігаючи несанкціонованому доступу. IAM забезпечує детальний контроль доступу. Amazon MSK знімає задачу управління кластерами Kafka. Разом ці сервіси формують цілісну хмарну стратегію, яка відповідає ключовим нефункціональним вимогам, таким як масштабованість, безпека та операційні накладні витрати, що є критично важливими для фінансового додатка виробничого рівня.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ

Розділ містить детальний опис технічної реалізації системи моніторингу фондового ринку, зокрема представлені засоби розробки, подієво-орієнтована мікросервісна архітектура з Apache Kafka як центральним компонентом, а також описані всі ключові елементи системи, їх призначення та взаємодія для забезпечення ефективного збору, обробки та візуалізації даних фондового ринку в реальному часі.

3.1 Засоби розробки серверної частини веб-додатку

Мова програмування Java

Java є провідною мовою програмування для серверних додатків, що характеризується високою надійністю, масштабованістю та універсальністю. Функціонування Java забезпечується віртуальною машиною (JVM), яка гарантує незалежність виконання програм від операційної системи [5].

Java пропонує ефективний інструментарій для розробки великомасштабних, надійних додатків з можливістю безперервної інтеграції та розгортання. Платформа надає доступ до численних API, що спрощують інтеграцію з різноманітними системами та сервісами. Модель безпеки Java забезпечує кілька рівнів захисту, включаючи перевірку байт-коду, завантажувачі класів та менеджер безпеки [6].

Фреймворк Spring

Spring Framework виступає комплексним інструментарієм для розробки корпоративних Java-додатків, що надає широкий набір функціональних можливостей та модулів. Архітектура Spring базується на принципах інверсії контролю та впровадження залежностей, що суттєво оптимізує процес створення модульних, тестованих додатків [7].

Spring Boot, як розширення базового фреймворку, забезпечує автоматичну конфігурацію та вбудовані сервери додатків, що дозволяє створювати виробничі додатки з мінімальними налаштуваннями. Spring Boot втілює підхід “convention over configuration”, значно скорочуючи час на розробку та запуск додатків. Готові до використання стартери для різних технологій полегшують інтеграцію з базами даних, системами обміну повідомленнями, сервісами безпеки та іншими компонентами [8].

Реактивний веб-фреймворк WebFlux реалізує неблокуючу модель обробки запитів, що є критично важливим для систем, які потребують ефективної обробки паралельних з’єднань. WebFlux забезпечує високу продуктивність при меншому споживанні ресурсів у порівнянні з традиційними блокуючими підходами. Фреймворк підтримує реактивні потоки даних, використовуючи Project Reactor, що дозволяє обробляти асинхронні події та потоки даних з високою ефективністю [9].

Специфікація R2DBC надає реактивний API для взаємодії з реляційними базами даних, принципово відмінний від традиційного блокуючого JDBC. Це забезпечує можливість здійснення неблокуючих операцій введення-виведення, що має принципове значення для високопродуктивних застосунків. R2DBC підтримує асинхронну обробку запитів та потокову передачу результатів, що дозволяє ефективно обробляти великі обсяги даних з мінімальним використанням ресурсів [10].

СУБД PostgreSQL

PostgreSQL являє собою потужну об’єктно-реляційну систему управління базами даних з відкритим вихідним кодом, що вирізняється надійністю, стабільністю та підтримкою складних типів даних [11].

Система характеризується повною відповідністю ACID-принципам, що забезпечує цілісність даних навіть у випадку системних збоїв. PostgreSQL підтримує складні SQL-запити, включаючи спільні табличні вирази, віконні функції, рекурсивні запити та розширені аналітичні можливості [11].

Наявність матеріалізованих представлень та потужних механізмів реплікації робить PostgreSQL ідеальним вибором для високонавантажених середовищ з

вимогами до високої доступності. PostgreSQL забезпечує ефективну роботу з JSON-даними, підтримує часові ряди та геопросторові дані, демонструє високу ефективність масштабування як на окремому сервері, так і через горизонтальне розширення [11].

3.2 Засоби розробки користувацького інтерфейсу веб-додатку

Мова програмування JavaScript

JavaScript є високорівневою, інтерпретованою мовою програмування, що підтримує різні парадигми: об’єктно-орієнтовану, функціональну та прототипно-орієнтовану. Первісно розроблена для клієнтської частини веб-сайтів, JavaScript наразі застосовується також на серверній стороні, у мобільній розробці та створенні десктопних додатків [12].

JavaScript відзначається динамічною типізацією, що надає гнучкість під час розробки та дозволяє швидко прототипувати додатки. Мова має першокласні функції, що можуть бути передані як аргументи, повернуті з інших функцій та присвоєні змінним, що забезпечує потужну функціональну парадигму програмування [12].

Мова підтримується всіма сучасними браузерами та має регулярні оновлення через стандарт ECMAScript, що забезпечує додавання нових можливостей та покращень [12].

Бібліотека React

React є JavaScript-бібліотекою для побудови користувацьких інтерфейсів. Бібліотека використовує компонентний підхід, що дозволяє створювати повторно використовувані UI-компоненти та ефективно керувати станом додатку [13].

Принципова особливість React — віртуальний DOM, який оптимізує оновлення інтерфейсу шляхом порівняння віртуальних DOM-дерев та оновлення лише змінених частин реального DOM. Це забезпечує високу продуктивність при відображенні складних інтерфейсів та оновленні значних обсягів даних [14].

Компонентна архітектура React забезпечує інкапсуляцію логіки та представлення в незалежні, повторно використовувані блоки. Це спрощує побудову та підтримку складних інтерфейсів, дозволяючи розробникам ефективно організовувати код та запобігати дублюванню [14].

Бібліотека підтримує JSX — розширення синтаксису JavaScript, що дозволяє писати HTML-подібні структури безпосередньо в JavaScript-коді. Це спрощує створення та розуміння компонентів, об’єднуючи логіку та розмітку в одному файлі [14].

Мова програмування TypeScript

TypeScript є мовою програмування з відкритим вихідним кодом, розробленою Microsoft, що надбудовою над JavaScript. Основна відмінність TypeScript — додавання статичної типізації, що підвищує якість коду, полегшує рефакторинг та виявлення помилок на етапі компіляції [15].

TypeScript підтримує модульність та просторові імена для організації коду, що покращує структуру великих проектів та сприяє організації логічних компонентів у розподілені модулі. Мова забезпечує підтримку enum для роботи з множинами констант, що підвищує читабельність коду та зменшує ймовірність помилок [15].

Система типів TypeScript включає типи узагальнення (generics), що дозволяють створювати повторно використовувані компоненти, які можуть працювати з різними типами даних, забезпечуючи при цьому типову безпеку. Це розширює можливості створення універсальних алгоритмів та структур даних [15].

TypeScript підтримує типізацію класів, наслідування та інтерфейси, що забезпечує повноцінне об’єктно-орієнтоване програмування з перевіркою типів. Це дозволяє розробникам застосовувати принципи SOLID та інші методології проектування, зберігаючи при цьому безпеку типів [15].

3.3 Хмарні технології веб-додатку

Платформа Apache Kafka

Apache Kafka є розподіленою платформою потокової обробки, що забезпечує високу пропускну здатність та стійкість до відмов. Платформа дозволяє створювати системи, здатні обробляти значні обсяги подій [16].

Kafka організує дані через теми, які можна розділяти для паралельної обробки, що уможливлює горизонтальне масштабування. Розподілення тем на партиції дозволяє розподіляти навантаження між кількома брокерами, забезпечуючи високу пропускну здатність та паралельну обробку даних [16].

Система зберігає повідомлення на диску протягом налаштованого періоду, забезпечуючи відтворення історичних даних. Це дозволяє реалізувати різноманітні сценарії обробки, включаючи аналіз даних, потокову обробку та інтеграцію систем [16].

Apache Kafka забезпечує стійкість до відмов через механізми реплікації даних між кількома брокерами. Кожна партиція має лідера та послідовників, що гарантує збереження даних навіть у випадку відмови окремих вузлів кластера [16].

Сервіс Confluent Schema Registry

Confluent Schema Registry є централізованим репозиторієм схем, призначеним для зберігання та управління версіями схем даних для подій Kafka. Сервіс забезпечує сумісність форматів даних, що можуть еволюціонувати з часом [17].

Schema Registry надає механізми для валідації схем повідомлень, що запобігає передачі невідповідних даних у системі. Це критично важливо для забезпечення надійності та цілісності потоків даних у складних розподілених системах [17].

Сервіс підтримує різні типи сумісності схем, включаючи зворотну, пряму та повну сумісність. Це дозволяє розробникам контролювати еволюцію схем даних, запобігаючи проблемам при зміні форматів повідомлень [17].

Schema Registry інтегрується з різними компонентами екосистеми Kafka, включаючи клієнтські бібліотеки, Kafka Connect та Kafka Streams. Це забезпечує єдиний підхід до управління схемами даних по всій системі [17].

Apache Avro — це система серіалізації даних, яка забезпечує компактне бінарне кодування та підтримує еволюцію схем. Вона зберігає схему поряд із даними, що дозволяє читачам інтерпретувати дані без необхідності знати схему заздалегідь [18].

Схеми Avro визначаються у форматі JSON, що робить їх зрозумілими для людей та легкими для обробки машинами. Самі дані серіалізуються у компактному бінарному форматі, що забезпечує ефективне зберігання та передачу. Ключовою перевагою Avro є вбудована підтримка еволюції схем, що дозволяє змінювати структуру даних з часом без порушення сумісності з існуючими даними [18].

Платформа Docker

Docker є платформою для розробки, доставки та запуску додатків у контейнерах. Технологія контейнеризації забезпечує ізоляцію програмного забезпечення у стандартизованих блоках, що включають увесь необхідний для роботи додатку інструментарій [19].

Docker використовує легкі контейнери, що поділяють ядро операційної системи, але працюють як ізольовані процеси. Це забезпечує високу ефективність використання ресурсів порівняно з традиційними віртуальними машинами, дозволяючи запускати більше контейнерів на одному фізичному або віртуальному хості [19].

Сервіс AWS Elastic Beanstalk

AWS Elastic Beanstalk — це повністю керований сервіс, яка дозволяє розробникам легко розгортати та масштабувати веб-застосунки. Розробники можуть просто завантажити свій код, а Elastic Beanstalk автоматично опрацьовує розгортання, від надання ресурсів, балансування навантаження та автоматичного масштабування до моніторингу стану програми [20].

Elastic Beanstalk надає повний контроль над ресурсами AWS, що використовуються для запуску додатків, дозволяючи розробникам за необхідності

доступатися до базових ресурсів. Також підтримує автоматичне масштабування, реагуючи на зміни навантаження на додаток. Це забезпечує оптимальне використання ресурсів та підтримку продуктивності системи під час пікових навантажень [20].

Сервіс керування ідентифікацією та доступом AWS IAM

IAM — це веб-сервіс, який допомагає безпечно контролювати доступ до ресурсів AWS. За допомогою IAM можна централізовано керувати користувачами, групами безпеки та ролями, а також встановлювати детальні дозволи для контролю доступу до сервісів та ресурсів AWS [21].

IAM дозволяє створювати та керувати користувачами AWS, призначати облікові дані безпеки, такі як ключі доступу, паролі та багатофакторна автентифікація. Сервіс забезпечує гнучкий механізм управління дозволами через політики, які визначають, які дії дозволені або заборонені для конкретних ресурсів [22].

Віртуальна приватна хмара Amazon

Amazon VPC дозволяє створювати ізольовану секцію хмари AWS, де можна запускати ресурси у визначеній віртуальній мережі. У VPC можна повністю контролювати віртуальне мережеве середовище, включаючи вибір власного діапазону IP-адрес, створення підмереж, налаштування таблиць маршрутизації та мережевих шлюзів. Сервіс забезпечує можливість створення різних мережевих топологій, включаючи публічні, приватні та гібридні мережі [23].

VPC забезпечує покращену безпеку, дозволяючи створювати групи безпеки та списки контролю доступу до мережі для контролю доступу до ресурсів у кожній підмережі. Це дозволяє реалізувати багаторівневу архітектуру безпеки та відповідати різним регуляторним вимогам [24].

Групи безпеки AWS

Групи безпеки AWS діють як віртуальний міжмережевий екран для Elastic Beanstalk, контролюючи вхідний і вихідний трафік. Вони дозволяють налаштовувати правила, які визначають, який мережевий трафік може досягти і залишити підключені ресурси [25].

Кожна група безпеки містить набір правил, які фільтрують трафік на основі протоколу, порту та джерела/призначення. Можна налаштувати групи безпеки для контролю доступу на рівні екземпляру або підмережі, забезпечуючи глибший рівень безпеки для застосунків у хмарі [26].

Сервіс Amazon MSK

Amazon MSK — це повністю керована Сервіс, яка полегшує створення, оновлення та керування кластерами Apache Kafka у хмарі. Kafka є популярною платформою для опрацювання потокових даних у реальному часі, і MSK дозволяє розробникам використовувати Apache Kafka для опрацювання потокових даних без необхідності керувати базовою інфраструктурою [27].

MSK автоматично надає та налаштовує кластери Apache Kafka, постійно відстежує стан кластера, автоматично замінює несправні вузли та спрощує налаштування безпеки з шифруванням у стані спокою та під час передачі [28].

Сервіс AWS Amplify

AWS Amplify — це набір інструментів та сервісів, які пришвидшують розробку мобільних та веб-застосунків. Amplify включає бібліотеки специфічні для конкретних платформ, компоненти інтерфейсу користувача, інструменти командного рядка та консоль для розробки, тестування та розгортання застосунків [29].

3.4. Діаграма прецедентів

Діаграма прецедентів, зображена на рисунку 3.1, відображає функціональні можливості системи з точки зору користувача. На діаграмі представлено основні варіанти використання системи:

Пошук акцій (Search stocks) – дозволяє користувачам знаходити акції за назвою компанії, тикером або іншими параметрами. Цей прецедент є одним із базових і часто використовується як початковий крок при роботі з системою.

Перегляд попереднього огляду акцій (View stock preview) – надає короткий огляд інформації про акції, включаючи поточну ціну та зміну ціни.

Перегляд повної інформації про акції (View full stock info) – забезпечує доступ до детальної інформації про вибрані акції. Цей прецедент включає: перегляд інформації про компанію, перегляд графіка цін акцій, завантаження новин щодо акцій та перегляд фінансової статистики. Також це включає прецедент перегляду інформації про компанію (View company info), що надає деталі про компанію-емітента, включаючи біржу, на якій торгується акція, ринкову капіталізацію, галузь та контактну інформацію та перегляд фінансової статистики (View financial statistics) – забезпечує доступ до ключових фінансових показників, включаючи квартальну статистику та показники ефективності.

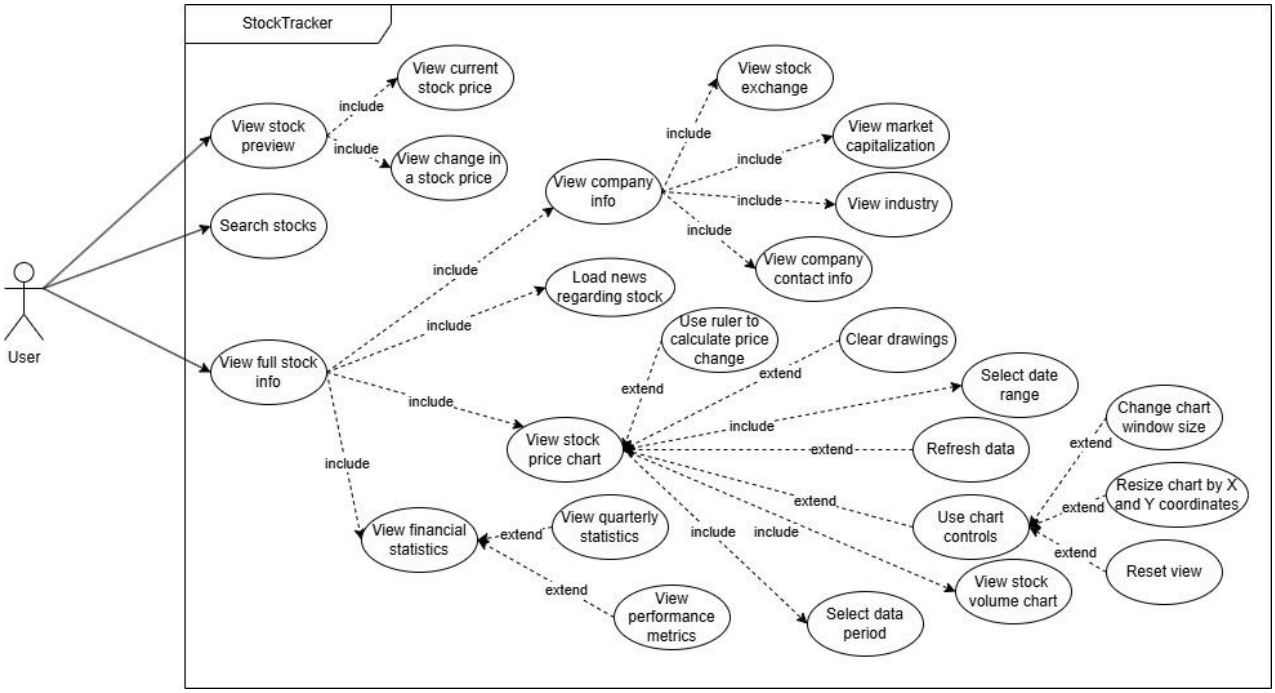

Рисунок 3.1 – Діаграма прецедентів

Перегляд графіка цін акцій (View stock price chart) – центральний прецедент, який дозволяє користувачам візуалізувати історичні ціни акцій у вигляді графіків. Цей прецедент має численні розширення, включаючи використання лінійки для розрахунку зміни ціни, вибір періоду даних, оновлення даних, використання

елементів керування графіком, перегляд графіка обсягу торгів та вибір діапазону дат.

Ці прецеденти та їх взаємозв'язки показують, як користувачі можуть взаємодіяти з системою для отримання необхідної інформації про фондовий ринок, аналізу цін акцій та прийняття інвестиційних рішень.

3.5. Архітектура системи

Розроблена система моніторингу фондового ринку реалізована на основі подієво-орієнтованої архітектури з використанням мікросервісного підходу. Архітектура системи, зображена на рисунку 3.2 включає кілька ключових компонентів, які працюють разом для забезпечення збору, обробки та візуалізації даних фондового ринку в реальному часі.

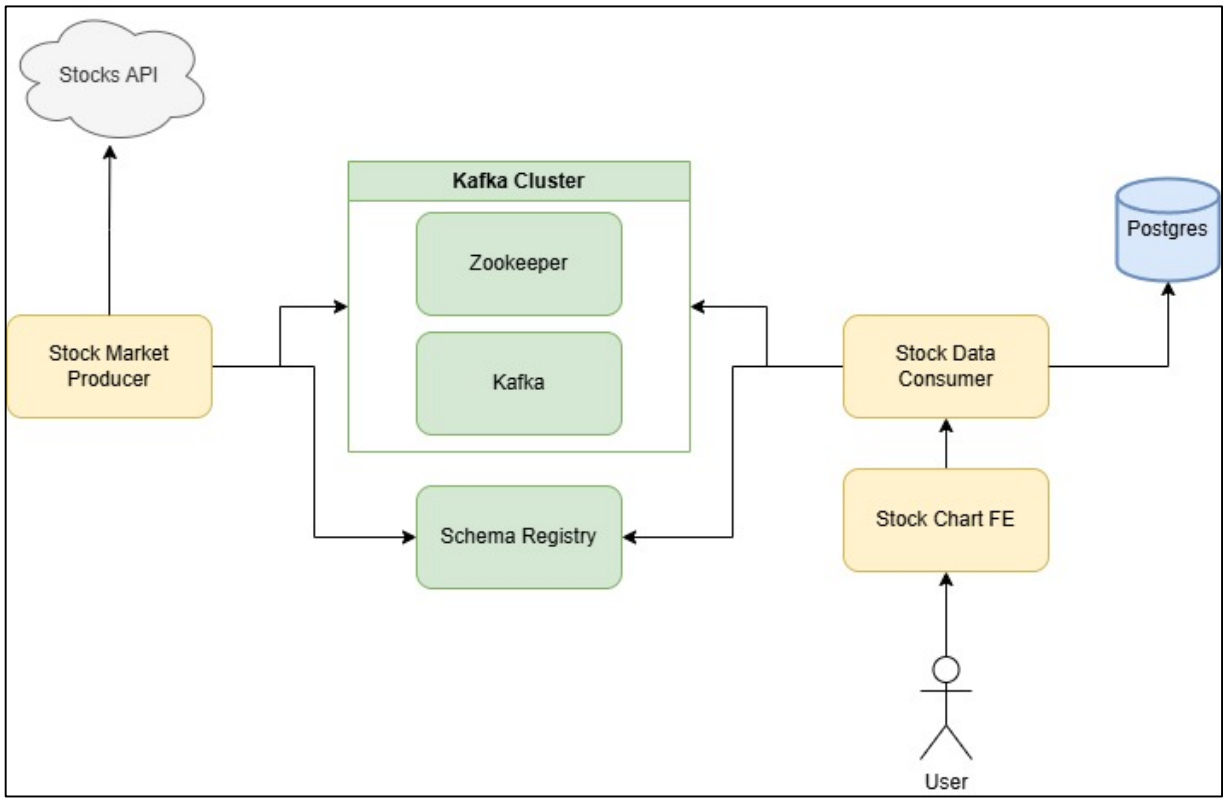

Рисунок 3.2 – Діаграма архітектури системи

Центральним елементом архітектури є Apache Kafka, який виступає в ролі розподіленого журналу подій і забезпечує надійний обмін повідомленнями між різними компонентами системи. Kafka дозволяє ефективно обробляти потоки даних у реальному часі, що є критичним для системи моніторингу фондового ринку, де оперативність отримання інформації про зміни цін акцій має першочергове значення.

Stock Market Producer – сервіс, який відповідає за збір даних з фондових ринків через Stocks API. Цей компонент постійно опитує зовнішні API для отримання актуальної інформації про ціни акцій і публікує ці дані в Kafka для подальшої обробки.

Kafka Cluster – центральна шина для передачі подій, що включає сам брокер Kafka та Zookeeper для управління кластером. Kafka забезпечує високу пропускну здатність і стійкість до відмов, що дозволяє системі обробляти великі обсяги даних в реальному часі без втрат.

Schema Registry – компонент, який відповідає за управління схемами подій, що публікуються в Kafka. Використання Schema Registry забезпечує сумісність даних між різними версіями продюсерів і споживачів, а також дозволяє еволюціонувати схеми даних з часом без порушення роботи системи.

Stock Data Consumer – сервіс, який споживає дані з Kafka, обробляє їх та зберігає в базі даних Postgres. Цей компонент реалізує бізнес-логіку обробки даних фондового ринку, включаючи розрахунок різних статистичних показників і метрик, а також містить АПІ для взаємодії з користувацьким інтерфейсом.

Postgres – реляційна база даних, яка використовується для зберігання історичних даних про ціни акцій та іншої релевантної інформації. Вибір Postgres обумовлений його надійністю, продуктивністю та підтримкою складних SQL-запитів, що важливо для аналізу даних фондового ринку.

Stock Chart FE – фронтенд-компонент системи, який забезпечує візуалізацію даних для кінцевих користувачів та надає інтерактивний інтерфейс для перегляду графіків цін акцій, пошуку акцій та отримання детальної інформації про компанії.

3.5. Реалізація серверної частини

Серверна частина системи реалізована на мові програмування Java з використанням фреймворку Spring Boot, що забезпечує надійний фундамент для розробки корпоративних додатків. Архітектура серверної частини побудована за принципами чистої архітектури (Clean Architecture), що дозволяє досягти високого рівня модульності, тестованості та масштабованості коду.

При розробці серверної частини особлива увага була приділена ефективній взаємодії з Kafka. Producer-компонент використовує Kafka Producer API для публікації даних про ціни акцій у відповідні теми.

Дані з зовнішнього АПІ через вебсокети зчитуються у реальному часі прод'юсером, приводяться до вигляду зазначеного у контракті Avro та надсилаються у Kafka.

Consumer-компонент також реалізований з використанням Spring Kafka, що надає зручний інтерфейс для обробки повідомлень з Kafka. Для забезпечення масштабованості та ефективної обробки даних використовується підхід з групами споживачів, що дозволяє розподілити навантаження між кількома екземплярами сервісу за необхідності.

Для серіалізації та десеріалізації повідомлень в Kafka використовується формат Avro у поєднанні з Schema Registry. Це забезпечує компактне представлення даних та строгу типізацію, що значно зменшує ймовірність помилок при обміні даними між різними компонентами системи.

Schema Registry зберігає та управляє еволюцією схем, що дозволяє різним версіям продюсерів та консьюмерів працювати разом без конфліктів.

Для реалізації API серверної частини використовується RESTful підхід, що забезпечує стандартизований інтерфейс для взаємодії з іншими компонентами системи. Для забезпечення передачі даних у реальному часі між сервером і клієнтом був реалізований механізм: Server-Sent Events, що застосовується для односторонньої передачі даних від сервера до клієнта, зокрема для трансляції оновлень цін акцій у реальному часі.

3.6. Схема бази даних

Система моніторингу фондового ринку використовує реляційну базу даних PostgreSQL для ефективного зберігання та швидкого доступу до інформації про акції, компанії та динаміку цін. Схему розробленої бази даних представлено на рисунку 3.3.

База даних складається з шести основних таблиць, кожна з яких відповідає за зберігання специфічної інформації:

companies – зберігає основну інформацію про компанії-емітенти, включаючи їхні назви, біржі на яких вони торгуються, країни реєстрації, ринкову капіталізацію, галузі діяльності та контактну інформацію.

stocks – містить інформацію про акції компаній, їх тикерні символи та зв’язки з відповідними компаніями-емітентами. Ця таблиця є центральною у моделі даних, оскільки більшість інших таблиць посилаються на неї через тикер символ.

trades – зберігає детальну інформацію про торгові операції з акціями, включаючи ціни, обсяги торгів та часові мітки. Ця таблиця має складений первинний ключ з тикерного символу та часової мітки, що дозволяє ефективно зберігати та швидко отримувати хронологічні дані про ціни.

company_news – містить новини, пов’язані з компаніями та їхніми акціями. Кожна новина має заголовок, короткий зміст, джерело, категорію та посилання на повний текст і зображення.

financial_time_series – зберігає фінансові показники компаній у форматі часових рядів. Різні метрики (такі як квартальний дохід, прибуток на акцію, тощо) зберігаються з прив’язкою до періоду та частоти вимірювання.

financial_point_metrics – призначена для зберігання точкових (разових) фінансових показників, які не формують часові ряди, але є важливими для аналізу компаній.

Для забезпечення високої продуктивності при роботі з великими обсягами даних у базі даних створено ряд стратегічних індексів, що гарантують швидкодію операцій читання, які є найбільш частими в системі моніторингу фондового ринку,

особливо при побудові графіків та відображенні аналітичної інформації на фронтенд-частині.

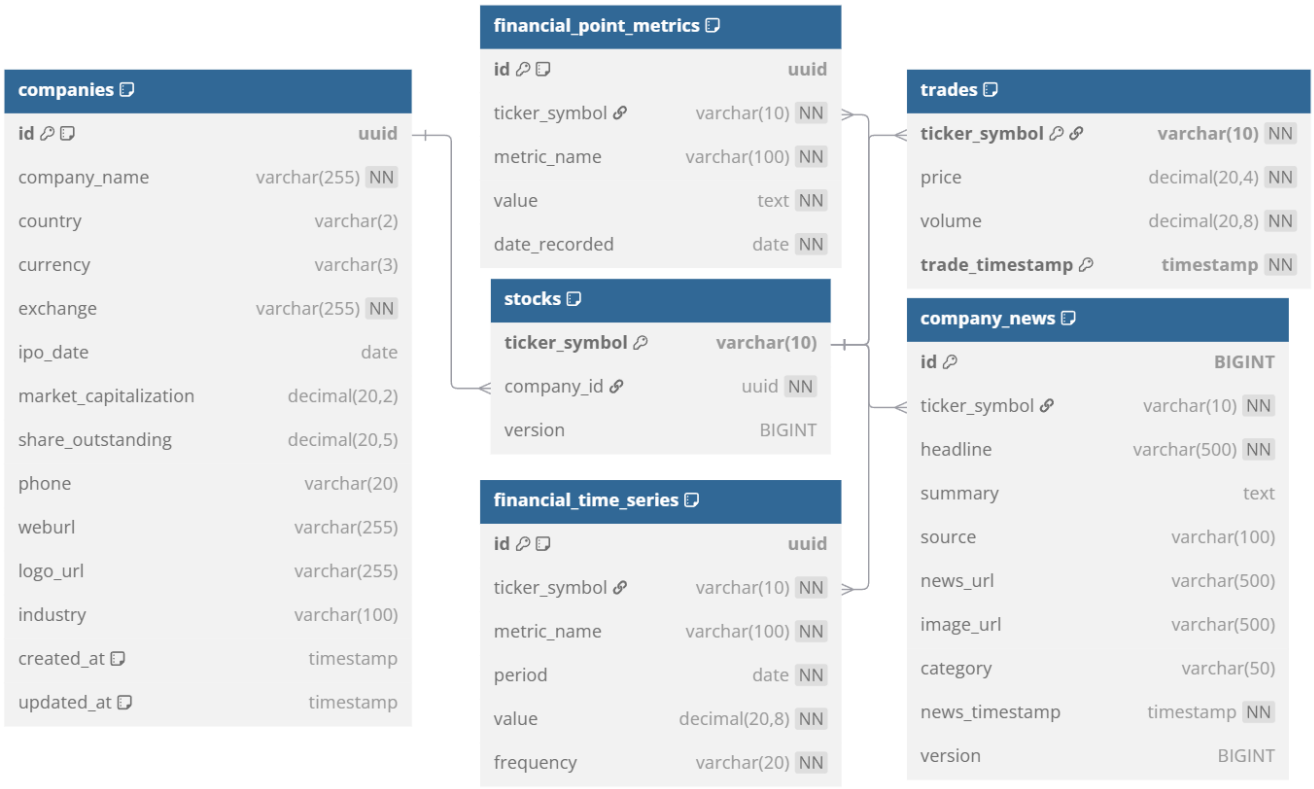

Рисунок 3.3 – Схема розробленої бази даних

Така структура дозволяє легко отримувати комплексну інформацію через SQL-запити з об’єднанням таблиць. Наприклад, можна отримати всі новини та торгові операції для конкретної компанії, або проаналізувати кореляцію між фінансовими показниками та динамікою цін акцій.

3.7. Реалізація клієнтської частини

Фронтенд-компонент системи (Stock Chart FE) реалізований з використанням сучасного стеку технологій: React для побудови інтерфейсу користувача, TypeScript для забезпечення строгої типізації. Архітектура клієнтської частини

побудована за принципами компонентного підходу, що забезпечує повторне використання коду та спрощує тестування і підтримку системи.

Інтерфейс користувача розроблений з фокусом на зручність використання та відповідає сучасним стандартам веб-дизайну. Для стилізації компонентів використовується комбінація CSS-модулів та Styled Components, що дозволяє писати модульний, масштабований CSS-код, уникаючи конфліктів імен класів.

Для візуалізації графіків цін акцій використовуються спеціалізовані бібліотеки, такі як Recharts, які дозволяють створювати інтерактивні, високопродуктивні графіки з багатою функціональністю. Графіки підтримують різні типи відображення (лінійний, свічковий, гістограма), масштабування, вибір часових інтервалів та інші функції, необхідні для аналізу динаміки цін.

Для роботи з SSE реалізований спеціальний клас EventSource, який встановлює з’єднання з серверним API і обробляє потік подій. При отриманні нових даних відбувається оновлення стану додатку через Redux, що автоматично призводить до перемальовування відповідних компонентів інтерфейсу.

Оптимізація продуктивності клієнтської частини досягається за рахунок використання техніки віртуалізації для відображення великих списків даних, мемоізації для запобігання зайвим перерахункам, code-splitting для зменшення розміру початкового завантаження та lazy-loading для відкладеного завантаження компонентів, які не потрібні на першому екрані.

Для забезпечення зручної навігації між різними сторінками додатку використовується React Router. Система маршрутизації дозволяє користувачам легко переходити між різними представленнями системи, такими як пошук акцій, детальна інформація про компанію, графіки цін та фінансова статистика.

4 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Розділ присвячений аспектам взаємодії користувача з розробленою системою моніторингу фондового ринку, описано вимоги для коректної роботи системи та містить демонстрацію функцій програми та взаємодію з її користувацьким інтерфейсом.

4.1 Системні вимоги

Апаратні вимоги:

- процесор – чотириядерний процесор (Intel Core i3-13100F / AMD Ryzen 3 3200G);
- пам’ять – не менше 8 ГБ;
- накопичувач – 10 ГБ;
- мережа – стабільне інтернет-з’єднання для завантаження Docker-образів та доступу до зовнішнього АПІ.

4.2 Приклад роботи з користувацьким інтерфейсом системи

Головною сторінкою веб-сайту є список акцій зображений на рисунку 4.1. Згори у правому куті відображається панель пошуку (надпис “Search stocks…”).

Окрім назв тикерів акцій також виводиться: назва компанії-емітента, ціна на даний момент та показник різниці на скільки ціна тої, чи іншої акції змінилася з моменту відкриття біржі.

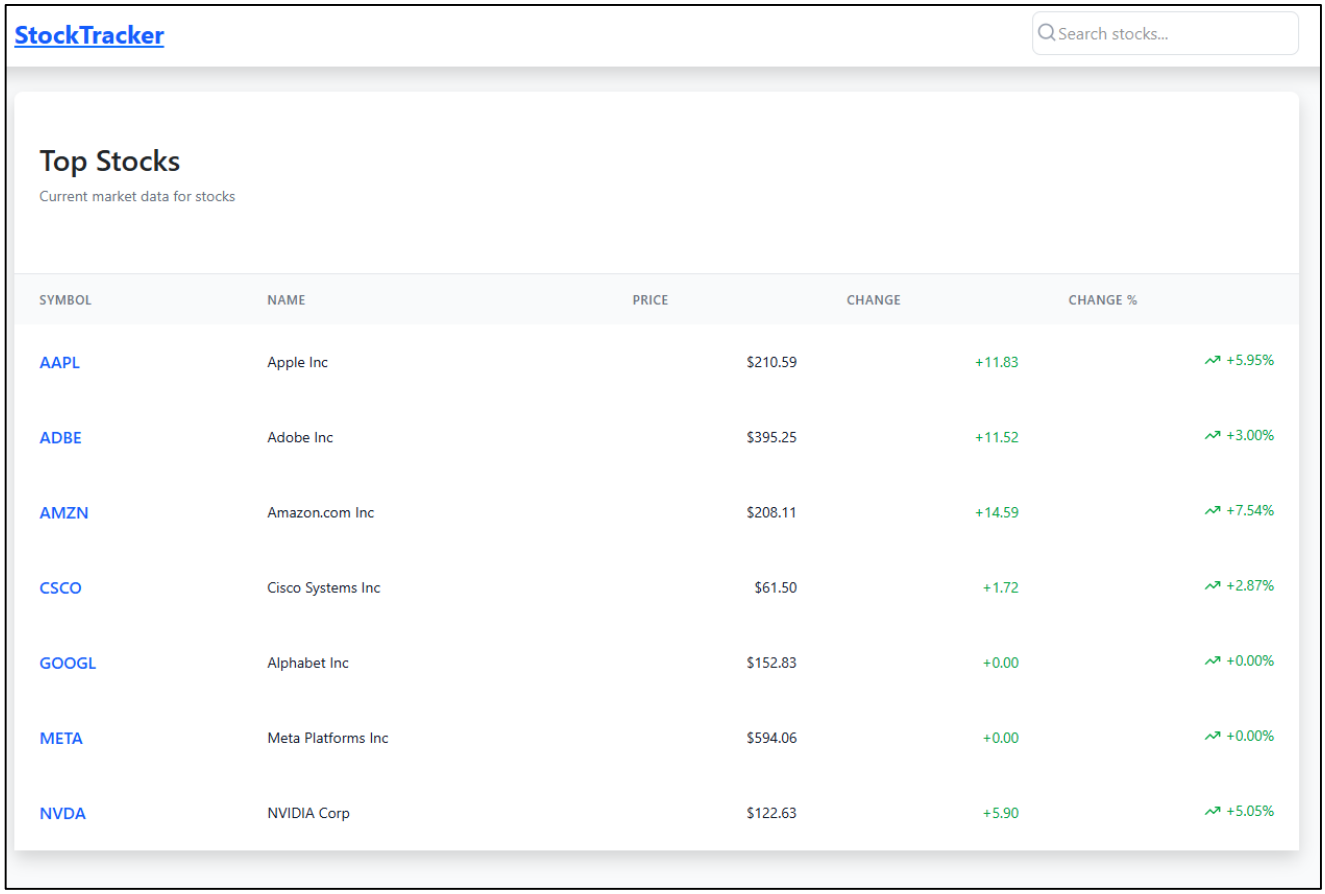

Рисунок 4.1 – Головна сторінка веб-сайту

Пошук акцій відбувається за назвою тикера. Після введення пошукового запиту натискаємо кнопку “Введення” і результати пошуку відображаються на екрані. Для швидкого стирання пошукового запиту можна скористатися хрестиком на полі введення. Приклад використання пошуку зображено на рисунку 4.2.

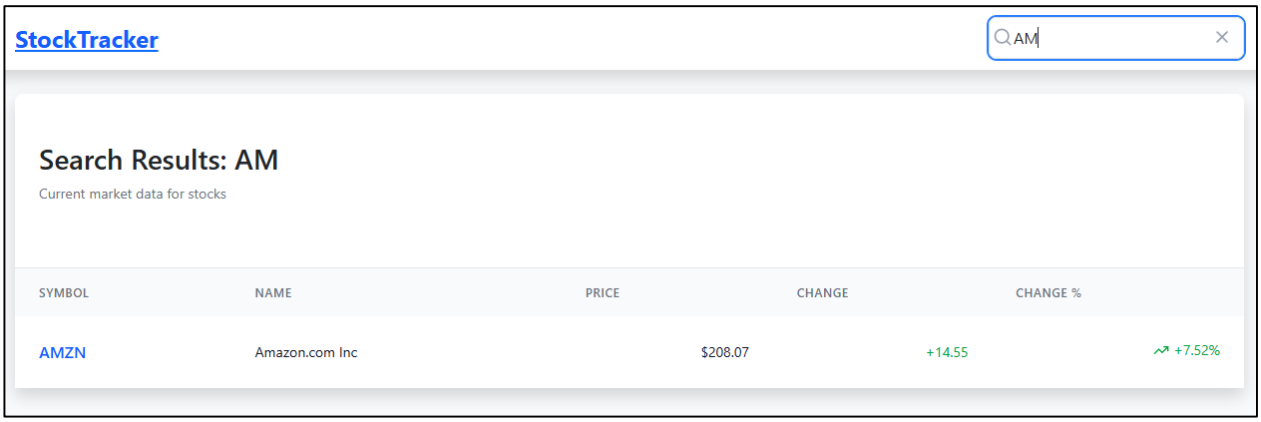

Рисунок 4.2 – Пошук акції за назвою тикера

У випадку якщо за вказаним запитом не буде знайдено жодної акції виведеться відповідне повідомлення зображене на рисунку 4.3.

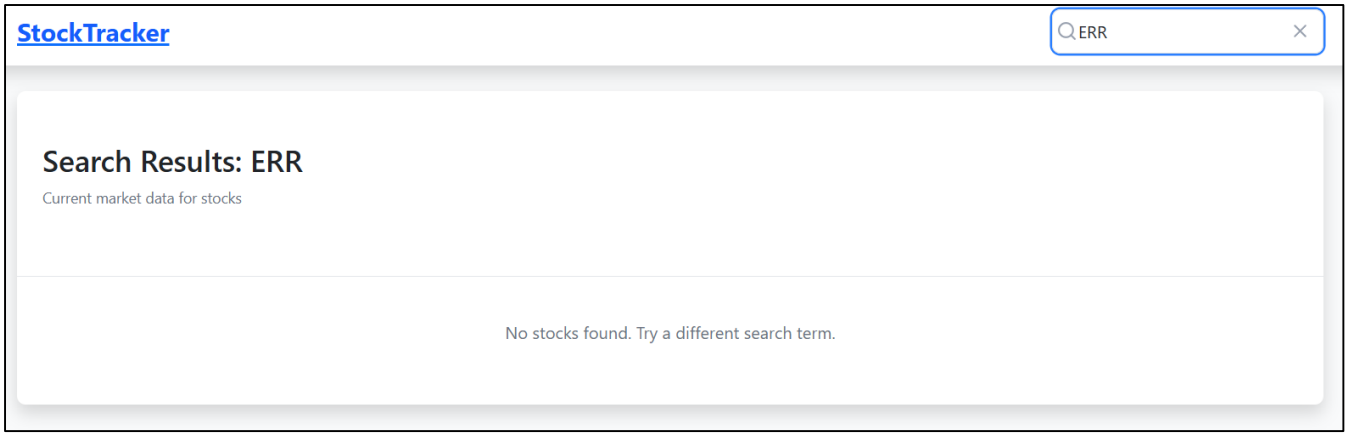

Рисунок 4.3 – Виведене повідомлення про відсутність акцій по пошуковому запиту

Натиснувши на тикер обраної акції переходимо на її сторінку зображену на рисунку 4.4. У верхній частині розташована панель з логотипом “StockTracker” зліва та полем пошуку акцій справа, де користувач може ввести назву тикера. Під навігаційною панеллю розміщена кнопка “Back to Stocks” зі стрілкою назад, що дозволяє повернутися до загального списку акцій.

Основний заголовок сторінки показує повну назву компанії з додатковою інформацією про біржу, тип ринку та сектор.

Сторінка акції має 4 вкладки: графік біржових акцій, інформація про компанію-емітента, фінансову інформацію по компанії та стірчку новин пов’язаних з даною акцією.

Почнемо з графіку. Центральну частину сторінки займає розділ “Market Overview”, що містить заголовок, поточну ціну з міткою “Live” та кнопкою “Refresh” для оновлення даних. Під ними розташована панель вибору часових інтервалів з опціями: 5 Min, 1 Day, 1 Week, 1 Month, 1 Year та All Time. Нижче розташовані 2 кнопки, перша “Лінійка”, за допомогою якої можна вимірювати на

скільки збільшилася або зменшилася ціна акції. Червона кнопка праворуч стирає усі малюнки з графіку.

Основний простір займає інтерактивний графік динаміки ціни акції, що показує поступове зростання з невеликими коливаннями. Внизу графіка відображаються обсяги торгів у вигляді зелених стовпців. Ще нижче зображено діапазон дат, за яким можна коригувати видиму частину графіка. Для зручності користувача додана панель “Chart Controls”, яка дозволяє налаштувати відображення графіка за потребою.

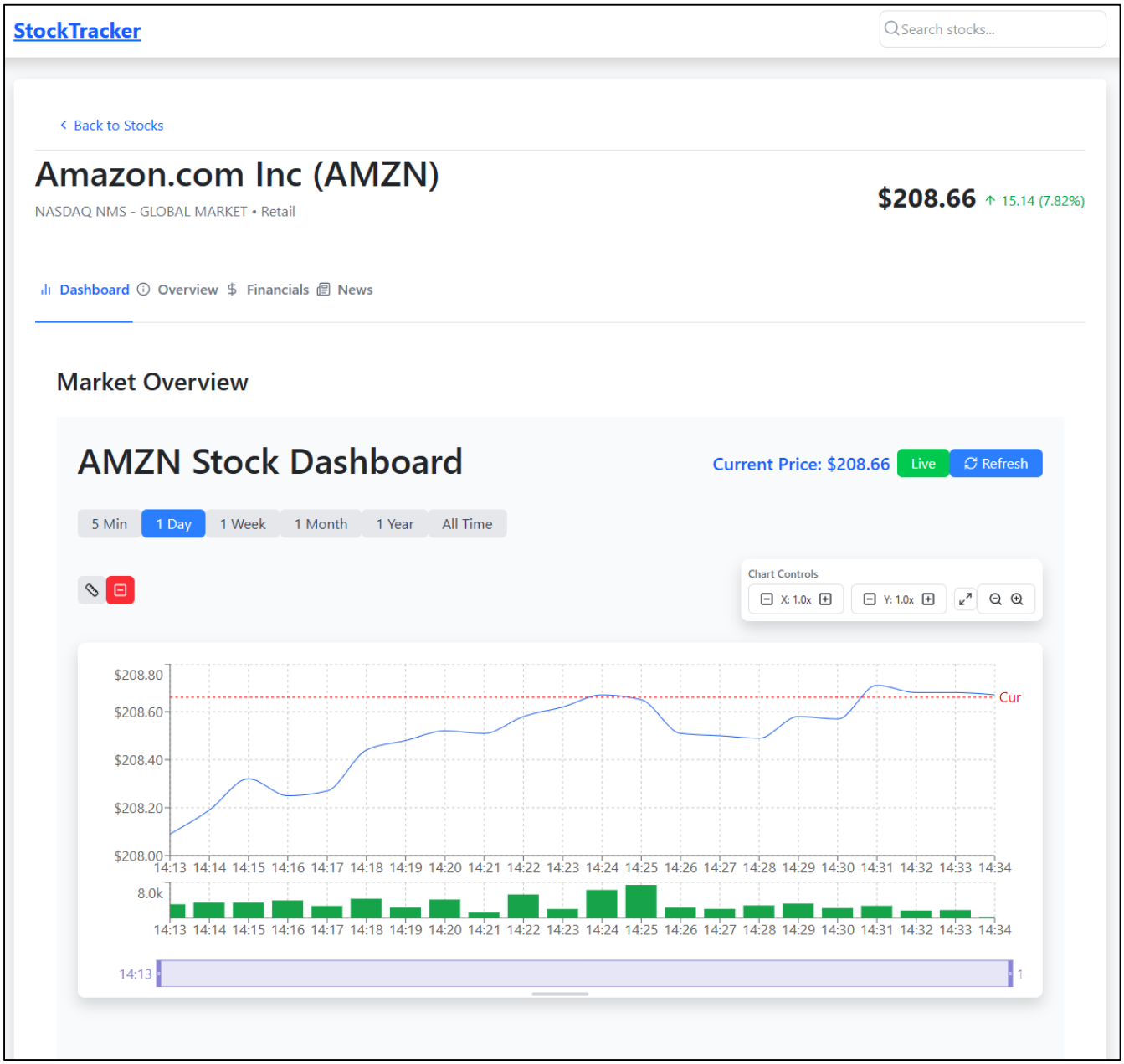

Рисунок 4.4 – Сторінка акції AMZN

Якщо користувач скористається “Лінійкою” – він перейде у режим малювання та зможе наносити на графік лінії, що відображають те, на скільки змінилася ціна впродовж вказаного часу. Результати виводяться посередині лінії та згори над графіком та позначаються різними кольорами. Приклад використання лінійки зображено на рисунку 4.5.

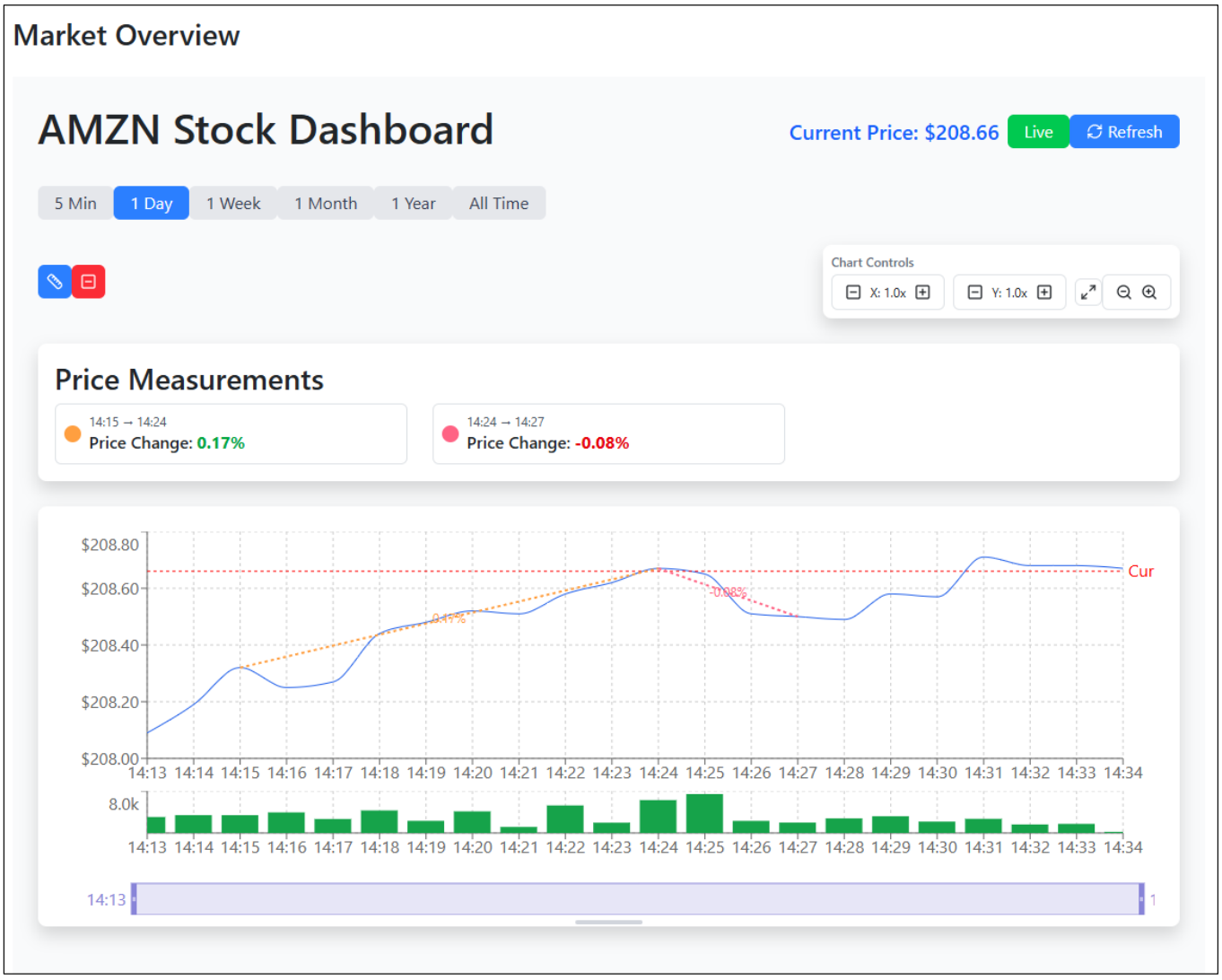

Рисунок 4.5 – Використання лінійки для замірів змін акції

При переході у перегляд цін за 1 рік відповідно зменшиться шкала на графіку і відповідно зменшиться точність вимірів. Результат проведених маніпуляцій зображено на рисунку 4.6.

Рисунок 4.6 – Вигляд графіку для виведення даних за останній рік

Скориставшись повзунком фіолетового кольору користувач може обрати період виведення даних на графіку, що забезпечує зручну фільтрацію інформації за часовим діапазоном. Для більш детального налаштування відображення система пропонує модифікатори координат X та Y, які дозволяють оптимізувати масштаб та діапазони значень по обох осях відповідно до аналітичних потреб. Додатково користувач має можливість збільшити або зменшити графік за допомогою панелі “Chart Controls”, що також містить функцію повернення до стандартних налаштувань. Комплексне використання цих інструментів дозволяє досягти найбільш інформативного представлення даних, зосередившись на значущих ділянках та тенденціях, як це продемонстровано на прикладі налаштувань виводу графіку, зображеному на рисунку 4.7.

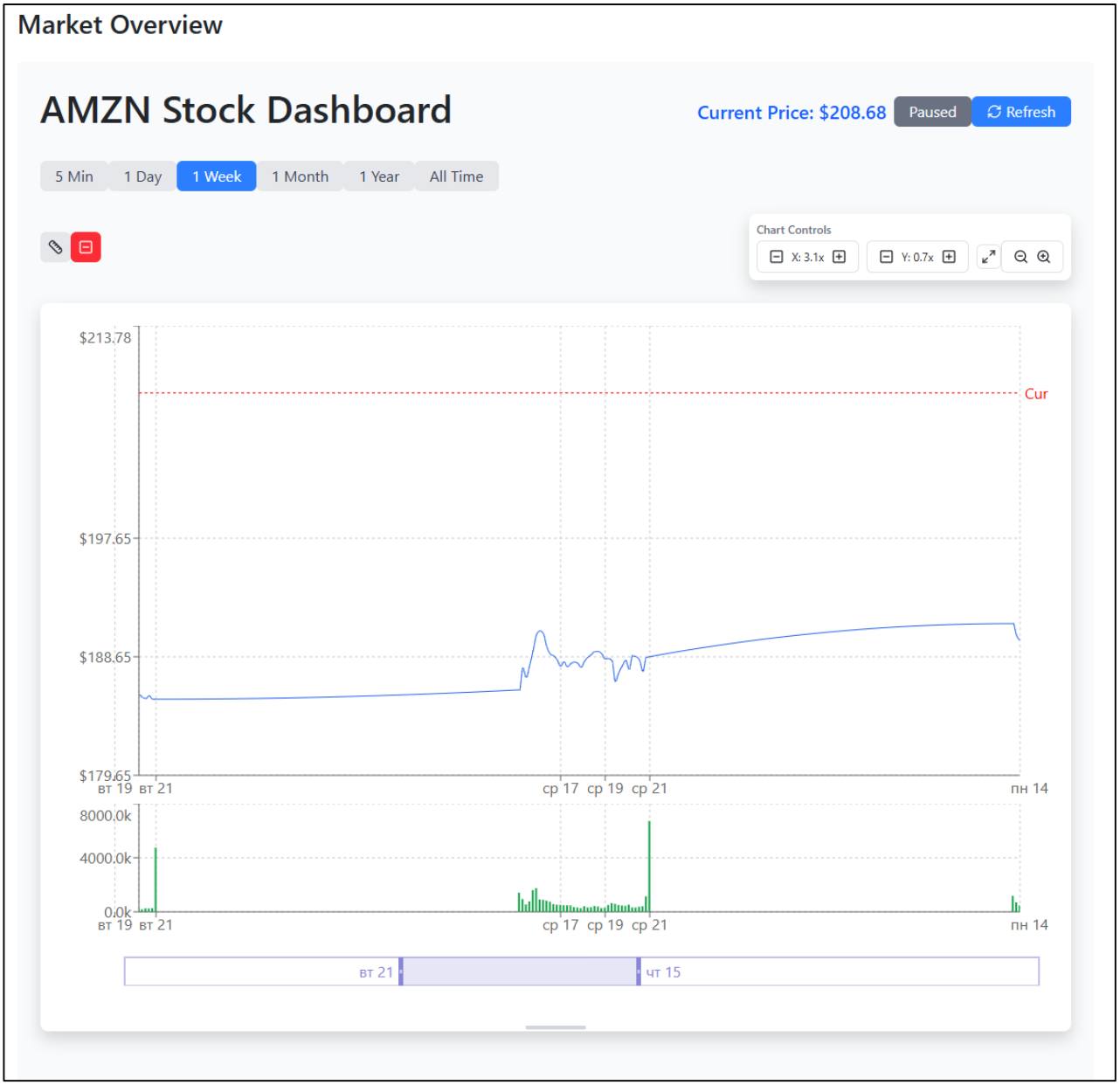

Рисунок 4.7 – Результати використання налаштувань виведення графіку

Перейдемо до вкладки “Overview”, зображеній на рисунку 4.8. На ній користувач може побачити загальну інформацію про компанію емітента. У верхній частині вкладки розміщені основні фінансові та ринкові показники компанії та акції у вигляді шести інформаційних блоків. Перший блок містить інформацію про біржу, на якій торгуються акції. Поруч знаходиться блок з ринковою капіталізацією компанії, що показує загальну вартість усіх акцій компанії на ринку. Третій блок вказує на галузь, до якої належить компанія.

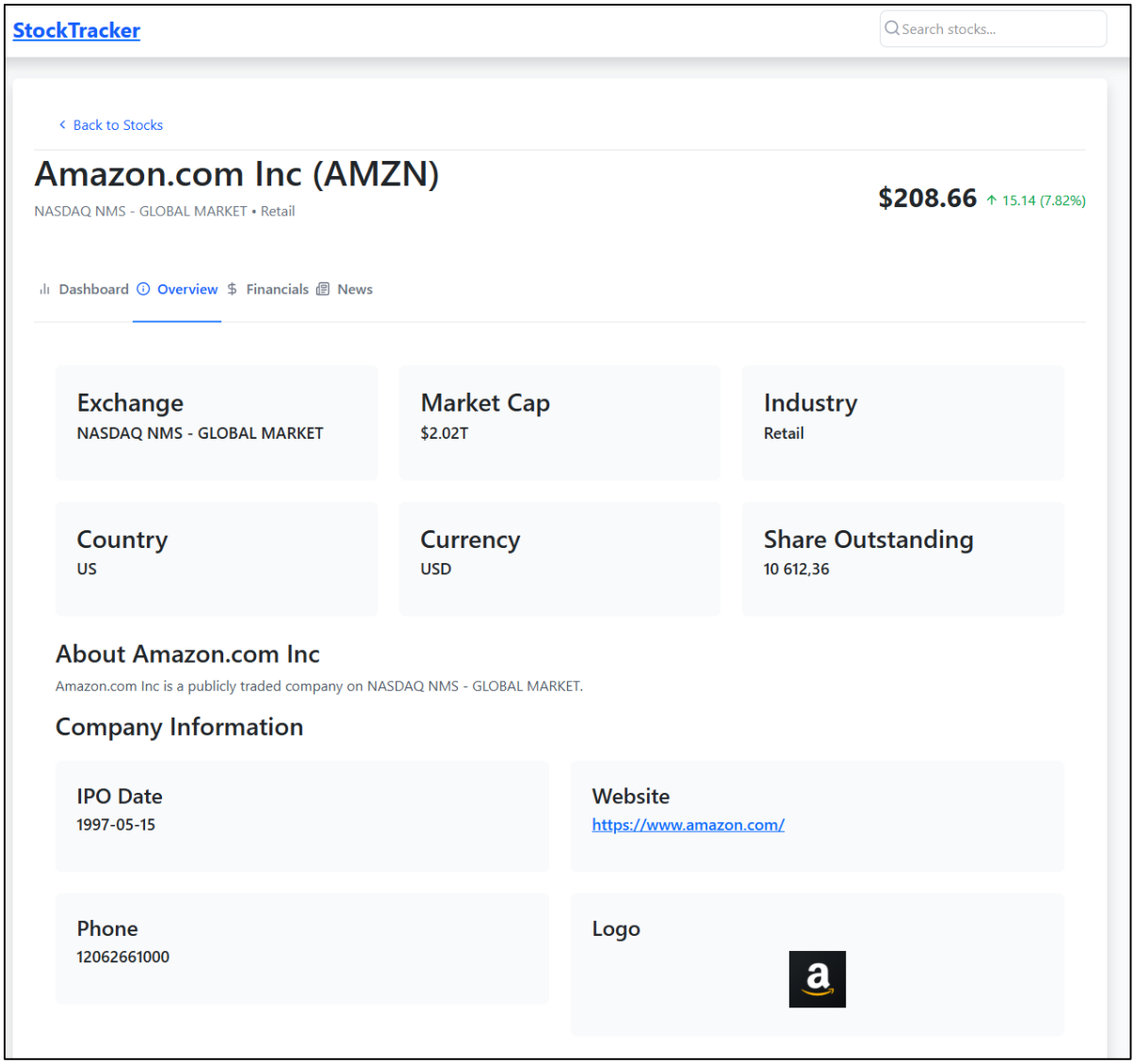

Рисунок 4.8 – Сторінка інформації про компанію

У другому ряду інформаційних блоків вказана країна походження компанії, валюта, в якій торгуються акції, та кількість акцій в обігу.

Нижче розташований розділ “About”, який містить короткий опис компанії. Цей розділ надає базову інформацію про правовий статус компанії. Далі йде розділ “Company Information” з детальнішими даними про компанію, представленими у чотирьох блоках. Тут користувач може дізнатися дату первинного публічного розміщення акцій, офіційний веб-сайт компанії, контактний телефон та побачити логотип компанії, який розміщений у правому нижньому кутку сторінки.

На вкладці “Financials”, зображеній на рисунку 4.9, представлено детальний огляд фінансових показників компанії, розділених на кілька тематичних блоків.

На початку розділу “Key Financials” представлені основні фінансові коефіцієнти компанії. Вони відображають загальну фінансову ефективність та прибутковість за останній період.

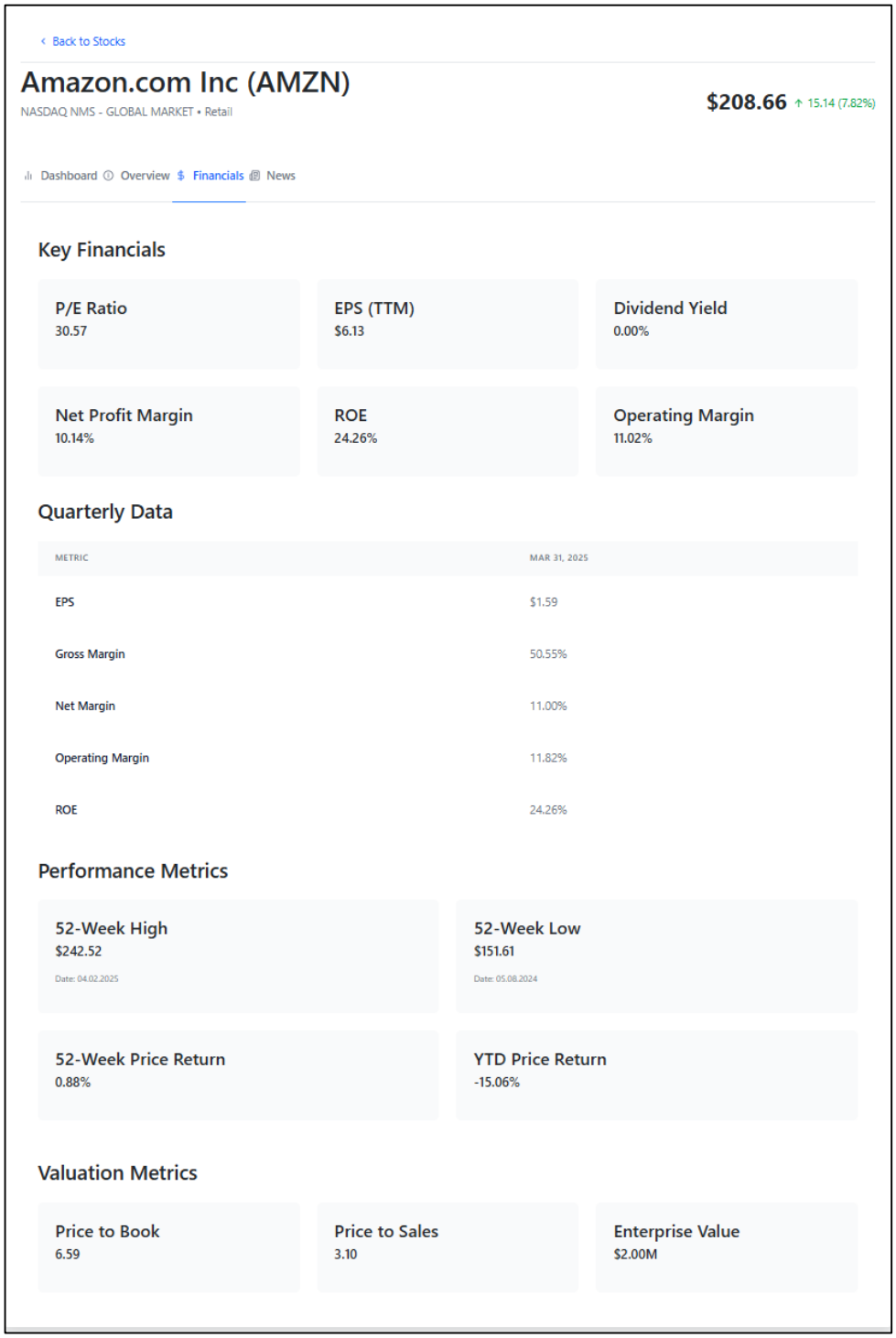

Рисунок 4.9 – Сторінка фінансової інформації про компанію

Нижче розташований розділ “Quarterly Data”, який містить таблицю з фінансовими показниками за останній звітний квартал. Ці дані допомагають інвесторам зрозуміти останні фінансові результати компанії.

Далі представлений розділ “Performance Metrics”, який включає інформацію про цінові показники акцій протягом останніх 52 тижнів.

В кінці сторінки розміщений розділ “Valuation Metrics”, який містить додаткові оцінні коефіцієнти. Вони допомагають інвесторам оцінити справедливу вартість компанії відносно різних фінансових показників.

На вкладці “News”, зображеній на рисунку 4.10, відображаються останні новини, пов’язані з обраною компанією.

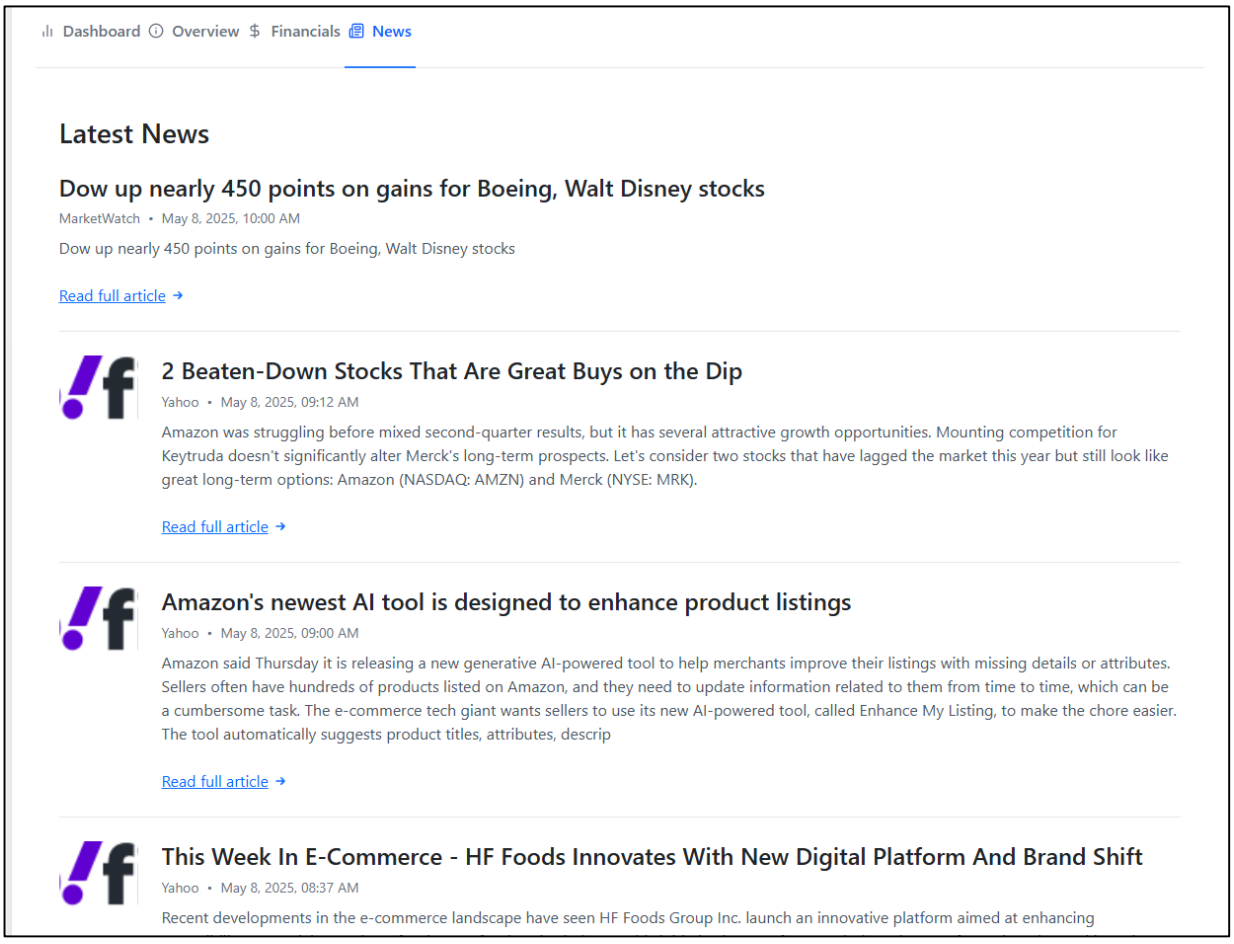

Рисунок 4.10 – Сторінка новин обраної акції

Кожна новина супроводжується датою публікації, часом та посиланням “Read full article” для переходу до повного тексту матеріалу.

ВИСНОВКИ

На основі аналізу підходів, методів та алгоритмів розв'язання поставленої задачі обґрунтовано використання мікросервісної архітектури, реактивного програмування, потокової обробки даних у режимі реального часу. Мікросервісний підхід забезпечує високу масштабованість системи, гнучкість у модифікації та нарощуванні функціоналу, незалежність компонентів та можливість їх незалежного розгортання й оновлення. Реактивне програмування дозволяє ефективно обробляти асинхронні потоки біржових даних, а потокова обробка забезпечує оперативне оновлення інформації про фінансові інструменти в режимі реального часу.

На основі аналізу програмних засобів обґрунтовано використання Spring Boot для розробки серверної частини, React для створення інтерактивного користувацького інтерфейсу, Apache Kafka для організації потокової обробки даних, PostgreSQL для зберігання структурованих даних, Redis для кешування швидкозмінної інформації. Обраний технологічний стек забезпечує високу продуктивність системи, надійність обробки великих обсягів біржової інформації та зручність супроводження програмного забезпечення.

Розроблено програмну систему "Веб-додаток для моніторингу та аналізу біржових акцій", яка включає систему моніторингу біржових акцій у режимі реального часу, інтерфейс для аналізу фінансових показників, механізми пошуку та відстеження акцій, графічну візуалізацію даних та оперативне оновлення біржової інформації. Система забезпечує високоефективну обробку фінансової інформації та надає користувачам зручні інструменти для прийняття обґрунтованих інвестиційних рішень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Річардсон К. Шаблони мікросервісів: з прикладами на Java / К. Річардсон. – Шелтер-Айленд, Нью-Йорк : Manning Publications, 2018. – 520 с.

2. Вілсон Р. Get your hands dirty on clean architecture: практичний посібник зі створення чистих вебзастосунків з використанням Java та Spring / Р. Вілсон : Leanpub, 2019. – 289 с.

3. WebSocket API. MDN Web Docs : веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 24.05.2025).

4. Server-sent events. MDN Web Docs : веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events (дата звернення: 24.05.2025).

5. What is Java and why do I need it? : веб-сайт. URL: https://www.java.com/en/download/help/whatis_java.html (дата звернення: 12.05.2025).

6. What is Java? Java Programming Language Explained : веб-сайт. URL: https://aws.amazon.com/what-is/java/ (дата звернення: 12.05.2025).

7. Spring Framework : веб-сайт. URL: https://spring.io/projects/spring-framework (дата звернення: 12.05.2025).

8. Spring boot: структура проекту та розробка веб-додатків : веб-сайт. URL: https://foxminded.ua/spring-boot/ (дата звернення: 12.05.2025).

9. Overview :: Spring Framework : веб-сайт. URL: https://docs.spring.io/spring-framework/reference/web/webflux/new-framework.html (дата звернення: 12.05.2025).

10. R2DBC : веб-сайт. URL: https://r2dbc.io/ (дата звернення: 12.05.2025).

11. About PostgreSQL : веб-сайт. URL: https://www.postgresql.org/about/ (дата звернення: 12.05.2025).

12. Вступ до JavaScript : веб-сайт. URL: https://uk.javascript.info/intro (дата звернення: 12.05.2025).

13. What is React JS? : веб-сайт. URL: https://codeinstitute.net/global/blog/what-is-react-js/ (дата звернення: 12.05.2025).

14. DOM vs Virtual DOM in React : веб-сайт. URL: https://pieces.app/blog/dom-virtual-dom-react (дата звернення: 12.05.2025).

15. TypeScript : веб-сайт. URL: https://www.typescriptlang.org/ (дата звернення: 12.05.2025).

16. Apache Kafka : веб-сайт. URL: https://kafka.apache.org/ (дата звернення: 12.05.2025).

17. Schema Registry Overview : веб-сайт. URL: https://docs.confluent.io/platform/6.2/schema-registry/index.html (дата звернення: 12.05.2025).

18. Починаємо роботу з Apache Kafka. Частина I : веб-сайт. URL: https://dou.ua/forums/topic/39363/ (дата звернення: 12.05.2025).

19. Docker on AWS : веб-сайт. URL: https://aws.amazon.com/docker/?nc1=h_ls (дата звернення: 12.05.2025).

20. Amazon Web Services. “AWS Elastic Beanstalk” : веб-сайт. URL: https://aws.amazon.com/elasticbeanstalk/ (дата звернення: 12.05.2025).

21. Amazon Web Services. “AWS Identity and Access Management (IAM)” : веб-сайт. URL: https://aws.amazon.com/iam/ (дата звернення: 12.05.2025).

22. Amazon Web Services. “IAM User Guide” : веб-сайт. URL: https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html (дата звернення: 12.05.2025).

23. Amazon Web Services. “Amazon Virtual Private Cloud (Amazon VPC)” : веб-сайт. URL: https://aws.amazon.com/vpc/ (дата звернення: 12.05.2025).

24. Amazon Web Services. “Amazon VPC User Guide” : веб-сайт. URL: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html (дата звернення: 12.05.2025).

25. Amazon Web Services. “Security Groups for Your VPC” : веб-сайт. URL: https://docs.aws.amazon.com/vpc/latest/userguide/vpc-security-groups.html (дата звернення: 12.05.2025).

26. Amazon Web Services. “Security Best Practices for Your VPC” : веб-сайт. URL: https://docs.aws.amazon.com/vpc/latest/userguide/vpc-security-best-practices.html (дата звернення: 12.05.2025).

27. Amazon Web Services. “Amazon Managed Streaming for Apache Kafka (Amazon MSK)” : веб-сайт. URL: https://aws.amazon.com/msk/ (дата звернення: 12.05.2025).

28. Amazon Web Services. “Amazon MSK Developer Guide” : веб-сайт. URL: https://docs.aws.amazon.com/msk/latest/developerguide/what-is-msk.html (дата звернення: 12.05.2025).

29. Amazon Web Services. “AWS Amplify” : веб-сайт. URL: https://aws.amazon.com/amplify/ (дата звернення: 12.05.2025).

ДОДАТОК А

Реалізація веб-додатку відстеження біржових акцій з використанням мікросервісної архітектури та хмарних технологій

Програмні засоби:

- мова програмування – Java;
- фреймворк – Spring;
- бібліотека – WebFlux.

```
@RestController
@RequestMapping("/trade")
@RequiredArgsConstructor
@Validated
@Slf4j
public class TradeController {

  private final TradeService tradeService;

  @GetMapping(value = "/{tickerSymbol}", produces =
MediaType.APPLICATION_NDJSON_VALUE)
  public Flux<TradeDto> getTradesByTickerSymbol(@PathVariable("tickerSymbol")
@NotBlank String tickerSymbol,
    LocalDateTime from, LocalDateTime to) {
   log.info("Handling getTradesByTickerSymbol: {}, from: {}, to: {}", tickerSymbol,
from, to);
   if (from.isAfter(to)) {
    throw new IllegalArgumentException("From date must be before to date");
   }
   return tradeService.getTradesByTickerSymbol(tickerSymbol, from, to);
  }
```

```
  @GetMapping(value = "/{tickerSymbol}/stream", produces =
MediaType.TEXT_EVENT_STREAM_VALUE)
  public Flux<TradeDto> getTradesByTickerSymbol(@PathVariable("tickerSymbol")
@NotBlank String tickerSymbol) {
   log.info("Handling SSE getTradesByTickerSymbol: {}", tickerSymbol);

   return tradeService.subscribeLiveTradesByTickerSymbol(tickerSymbol)
    .doOnSubscribe(s -> log.info("Client subscribed to trade stream for {}",
tickerSymbol))
    .doOnCancel(() -> log.info("Client unsubscribed from trade stream for {}",
tickerSymbol)).log();
  }
}

@Service
@RequiredArgsConstructor
@Transactional(readOnly = true)
@Slf4j
public class TradeServiceImpl implements TradeService {

  private final TradeRepository tradeRepository;

  @Transactional
  @Override
  public Mono<TradeDto> saveTrade(TradeDto tradeDto) {
   return tradeRepository
    .findByTickerSymbolAndAndTradeTimestamp(tradeDto.tickerSymbol(),
tradeDto.tradeTimestamp())
    .defaultIfEmpty(tradeDto).flatMap(trade -> {
```

```
      Mono<TradeDto> saveMono;
      if (Objects.equals(tradeDto, trade)) {
       saveMono = tradeRepository.saveTrade(tradeDto);
      } else {
       log.info("Trade existed: {} and trying saving: {}", trade, tradeDto);
       saveMono =
tradeRepository.updateTrade(TradeDto.builder().tickerSymbol(tradeDto.tickerSymbol()
)
         .tradeTimestamp(tradeDto.tradeTimestamp())
         .price((tradeDto.price() * tradeDto.volume() + trade.price() * trade.volume())
          / (tradeDto.volume() + trade.volume()))
         .volume(tradeDto.volume() + trade.volume()).build());
      }
      return saveMono.doOnNext(this::emitTradeUpdate);
     });
  }

  private void emitTradeUpdate(TradeDto trade) {
   Sinks.Many<TradeDto> sink = tradeStreams.get(trade.tickerSymbol());
   if (sink != null) {
    sink.tryEmitNext(trade);
    log.info("Emitted trade update for {}: price={}, volume={}", trade.tickerSymbol(),
trade.price(),
      trade.volume());
   } else {
    log.info("No subscribers for trade ticker: {}", trade.tickerSymbol());
    tradeStreams.put(trade.tickerSymbol(),
Sinks.many().multicast().onBackpressureBuffer());
   }
  }
```

```
  @Override
  public Flux<TradeDto> getTradesByTickerSymbol(String tickerSymbol,
LocalDateTime from, LocalDateTime to) {
   return tradeRepository.findTradesByTickerSymbol(tickerSymbol, from, to);
  }

  private final Map<String, Sinks.Many<TradeDto>> tradeStreams = new
ConcurrentHashMap<>();

  @Override
  public Flux<TradeDto> subscribeLiveTradesByTickerSymbol(String tickerSymbol) {
   return tradeStreams.computeIfAbsent(tickerSymbol, k ->
Sinks.many().multicast().onBackpressureBuffer()).asFlux()
     .doOnCancel(() -> {
      if (tradeStreams.containsKey(tickerSymbol)) {
       tradeStreams.remove(tickerSymbol);
       log.debug("Removed trade stream for ticker: {}", tickerSymbol);
      }
     }).doOnTerminate(() -> {
      if (tradeStreams.containsKey(tickerSymbol)) {
       tradeStreams.remove(tickerSymbol);
       log.debug("Removed trade stream for ticker: {}", tickerSymbol);
      }
     });
  }
}

@Service
@Slf4j
```

```
@RequiredArgsConstructor
public class TradeUpdateConsumer implements CommandLineRunner {
  private final TradeAvroMapper tradeAvroMapper;
  private final TradeService tradeService;
  private final ReactiveKafkaConsumerTemplate<String, Trade>
reactiveKafkaTradeConsumerTemplate;

  private Flux<TradeDto> consumeTradeUpdates() {
   return reactiveKafkaTradeConsumerTemplate.receiveAutoAck()
     .doOnNext(consumerRecord -> log.info("Consumed event key={}, value={}",
consumerRecord.key(),
       consumerRecord.value()))
     .map(ConsumerRecord::value).map(tradeAvroMapper::toDomain)
     .concatMap(tradeDto -> tradeService.saveTrade(tradeDto).onErrorResume(t -> {
      log.error("Error happened while saving trade: {}", t.getMessage());
      return Mono.empty();
     })).doOnError(throwable -> log.error("Error happened while consuming : {}",
throwable.getMessage(),
       throwable));
  }

  @Override
  public void run(String... args) {
   consumeTradeUpdates().subscribe();
  }
}
```