Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп’ютерних наук

Допущено до захисту
Завідувач кафедри комп’ютерних наук,
доктор техн. наук, професор
Андрій БОНДАРЧУК
«01» червня 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «бакалавр»
Спеціальність 122 Комп’ютерні науки
Освітня програма 122.00.01 Інформатика

Тема: МОБІЛЬНИЙ ДОДАТОК МОНІТОРИНГУ ТА АНАЛІТИКИ КРИПТОВАЛЮТНОГО ПОРТФЕЛЯ

Виконав
студент групи ІНб-1-22-4.0д
Хоменко Нікіта Юрійович

_____
(підпис)
Науковий керівник
кандидат технічних наук,
доцент Троценко Л.О

_____
(підпис)

Київ – 2026

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

«Затверджую»
Завідувач кафедри комп'ютерних наук, доктор техн. наук, професор
Андрій БОНДАРЧУК
«31 » жовтня 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

«Мобільний додаток моніторингу та аналітики криптовалютного портфеля»

Виконавець:студент групи ІНб-1-22-4.0д.
спеціальності 122 Комп'ютерні наук,
освітньої програми 122.00.01 Інформатика
Хоменко Нікіта Юрійович

1. Вихідні дані: Законодавство та нормативні акти в інформаційній сфері України, наукові праці та електронні ресурси в напрямку досліджень, хмарні сервіси Firebase, сучасні системи моніторингу криптовалюти, технології штучного інтелекту, OpenAI API, матеріали про інформаційну безпеку веб-застосунків.
2. Основними завданнями бакалаврської роботи є: Огляд наукових публікацій та сучасних технологічних рішень у сфері моніторингу криптовалютних активів. Обґрунтування мети, об'єкта, предмета та методів дослідження. Аналіз існуючих платформ та виявлення їхніх функціональних обмежень. Проєктування архітектури веб-застосунку:визначення структури, моделі даних Firestore та механізмів інтеграції зовнішніх API. Тестування платформи, аналіз результатів, формулювання висновків, підготовка тез та презентаційних матеріалів.
3. Пояснювальна записка: Обсяг – до 76 стор. формату А4 комп'ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.
4. Графічні матеріали: Презентація кваліфікаційної роботи, діаграма архітектури системи зі структурою компонентів та схемою взаємодії з API.
5. Додатки: Лістинги програмного коду, конфігурація Firebase, скріншоти роботи системи, результати тестування.
6. Строк подання роботи на кафедру: *«5 » червня 2025 р.*

Науковий керівник: кандидат технічних наук доцент _____ Троценко Л.О . 31.10.2025 дата	Виконавець: студентка групи ІНБ-1-22-4.0д _____ Хоменко Н.Ю 31.10.2025 дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк виконання етапу роботи	Примітка
1	Затвердження теми кваліфікаційної роботи	31.10.2025	Виконано
2	Аналіз проблеми, вивчення аналогів, формування цілей і завдань	01.11.2025 11.01.2026	Виконано
3	Аналіз сучасних рішень, літературних джерел та вибір оптимальних технологій для реалізації системи	26.01.2026 15.03.2026	Виконано
4	Дослідження архітектури програмного забезпечення та проектування основних модулів системи.	16.03.2026 31.03.2026	Виконано
5	Розробка, інтеграція та тестування основних функціональних модулів	01.04.2026 15.04.2026	Виконано
6.	Проведення тестування, аналіз результатів, виявлення та усунення помилок	16.04.2026 30.04.2026	Виконано
7.	Впровадження готового проекту	01.05.2026 15.05.2026	Виконано
8.	Створення звіту з кваліфікаційної роботи	25.05.2026	Виконано
9.	Подання роботи на перевірку, проходження антиплагіату, внесення правок керівника	27.05.2026	Виконано
10.	Підготовка презентаційних матеріалів, тексту доповіді та захист кваліфікаційної роботи	12.06.2026	Виконано
11.	Захист кваліфікаційної роботи	16.06.2026	

«Затверджую»

Науковий керівник

к.т.н., Троценко Л.О

Індивідуальний план склав

Хоменко Н.Ю.

РЕФЕРАТ

Кваліфікаційна робота: до 77 с., 3 рис., 6 таблиць, 36 посилань.

Актуальність: Стрімке зростання криптовалютного ринку, обсяг якого перевищує 2,5 трлн доларів США, формує стійкий попит на інструменти моніторингу та аналізу цифрових активів у реальному часі. Існуючі рішення здебільшого є або громіздкими десктопними платформами, або мобільними застосунками з обмеженим аналітичним функціоналом. Розробка веб-застосунку, який поєднує потокову передачу ринкових даних, інтерактивну графіку та інтелектуальний аналіз на основі штучного інтелекту, є актуальним науково-практичним завданням.

Об'єкт дослідження: процес моніторингу, відображення та аналізу криптовалютних активів у веб-середовищі в режимі реального часу.

Предмет дослідження: веб-застосунок Crypto Screener для моніторингу та аналітики криптовалютного портфеля, його компонентна архітектура, механізми інтеграції з біржовими API та інструментами штучного інтелекту.

Мета роботи: розробка веб-застосунку Crypto Screener для моніторингу та аналітики криптовалютного портфеля, що забезпечує відображення ринкових даних у режимі реального часу, ведення персонального портфеля та інтелектуальну аналітичну підтримку користувача.

Завдання:

1. Аналіз сучасних платформ для моніторингу криптовалютного ринку та виявити їхні переваги й недоліки;
2. Дослідження можливостей використання REST API та WebSocket-протоколу біржі Binance для отримання ринкових даних у реальному часі;
3. Обґрунтування технологій, на яких буде побудовано програмний продукт (React, Vite, Tailwind CSS, Firebase, lightweight-charts, OpenAI API);
4. Реалізувати модуль скринінгу монет із підтримкою пошуку, фільтрації та вотчліста;
5. Створення модулів реєстрації та входу користувачів та модель зберігання даних у Firebase Firestore;

6. Інтегрувати OpenAI API для надання аналітичних відповідей у модулі AI Trading Assistant;

7. Реалізувати модуль керування портфелем із розрахунком поточної вартості та показника PnL;

8. Тестування розробленої системи на практиці.

Основні результати: у ході виконання кваліфікаційної роботи розроблено веб-застосунок Crypto Screener. Реалізовано систему автентифікації користувачів засобами Firebase Authentication з підтримкою реєстрації та входу за email і паролем. Створено модуль скринінгу, який завантажує дані по всіх USDT-парах із ендпоінту Binance REST API /api/v3/ticker/24hr та оновлює їх кожні 5 секунд. Реалізовано свічковий графік на бібліотеці lightweight-charts v5 з підтримкою таймфреймів 1m, 5m, 15m, 1h, 4h, 1d та потоковим оновленням через WebSocket-з'єднання з потоком {symbol}@kline_{timeframe}. Розроблено модуль портфеля з можливістю додавання та видалення активів, автоматичним розрахунком поточної вартості та прибутку/збитку (PnL). Дані портфеля зберігаються у хмарній базі даних Firestore у колекції portfolios. Інтегровано AI Trading Assistant на основі моделі gpt-4o-mini, який надає аналітичні відповіді на запити користувача щодо криптовалютного ринку.

Практичне значення дослідження: розроблений застосунок може бути використаний як інструмент для індивідуального трейдера або інвестора для відстеження ринкової динаміки, ведення обліку криптовалютних активів та отримання аналітичної підтримки на основі генеративного штучного інтелекту. Архітектурні рішення, застосовані у роботі, можуть слугувати основою для розширення функціоналу до повноцінної торгової платформи.

Ключові слова: криптовалюта, скринер, моніторинг, портфель, Binance API, WebSocket, REST API, React, Firebase, Firestore, lightweight-charts, OpenAI API, GPT-4o-mini, свічковий графік, PnL, шту чний інтелект.

ПЕРЕЛІК УМОВНИХ ПОЗ НАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI (Artificial Intelligence) – штучний інтелект.

API (Application Programming Interface) – програмний інтерфейс взаємодії між програмними компонентами.

BaaS (Backend-as-a-Service) – модель хмарних обчислень, у якій серверна інфраструктура надається як готовий сервіс.

CRUD (Create, Read, Update, Delete) – базові операції з даними: створення, читання, оновлення та видалення.

CSS (Cascading Style Sheets) – каскадні таблиці стилів для оформлення веб-сторінок.

DOM (Document Object Model) – об'єктна модель документа, програмний інтерфейс для HTML-документів.

Firebase – хмарна платформа від Google для розробки веб- та мобільних застосунків.

Firestore – хмарна NoSQL база даних Firebase для зберігання та синхронізації даних у реальному часі.

GPT (Generative Pre-trained Transformer) – архітектура великої мовної моделі компанії OpenAI.

HTML (HyperText Markup Language) – мова розмітки гіпертекстових документів.

HTTPS (HyperText Transfer Protocol Secure) – захищена версія протоколу HTTP із шифруванням TLS.

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

JSX (JavaScript XML) – синтаксичне розширення JavaScript для опису структури інтерфейсу у React.

LLM (Large Language Model) – велика мовна модель, навчена на масивних текстових корпусах.

NoSQL – тип нереляційних систем управління базами даних.

OHLCV (Open, High, Low, Close, Volume) – стандартний формат біржових даних: відкриття, максимум, мінімум, закриття, обсяг.

OpenAI – компанія, що розробляє технології штучного інтелекту та великі мовні моделі.

PnL (Profit and Loss) – показник прибутку та збитку інвестиційного портфеля.

REST (Representational State Transfer) – архітектурний стиль побудови програмних інтерфейсів.

SPA (Single Page Application) – односторінковий застосунок, у якому навігація відбувається без перезавантаження сторінки.

UI (User Interface) – користувацький інтерфейс.

UX (User Experience) – користувацький досвід взаємодії з системою.

Vite – сучасний інструмент збірки фронтенд-застосунків із підтримкою HMR.

WebSocket – протокол повнодуплексного зв'язку між клієнтом і сервером у реальному часі.

Wotchlist (вотчліст) – персональний список активів, за якими користувач веде спостереження.

ЗМІСТ

ВСТУП

Цифрова економіка XXI століття характеризується стрімким розвитком децентралізованих фінансових інструментів, серед яких особливе місце посідають криптовалюти. Загальна капіталізація криптовалютного ринку щорічно демонструє значну волатильність і у окремі періоди перевищувала позначку 2,5 трлн доларів США, що зіставно з ринковою капіталізацією великих традиційних фондових бірж. Стрімке зростання кількості торгових інструментів – лише на біржі Binance налічується понад 400 активних торгових пар із прив'язкою до USDT – висуває підвищені вимоги до якості інструментів моніторингу та аналітики.

Сучасний учасник криптовалютного ринку – трейдер або портфельний інвестор – потребує зручного, надійного та функціонального інструменту для відстеження ринкової динаміки, ведення обліку власних активів і отримання аналітичних підказок. Наявні рішення, такі як CoinMarketCap, CoinStats або Delta, або є надто універсальними і перевантаженими зайвим функціоналом, або надають лише обмежені можливості для персоналізованої аналітики. Жодне з розглянутих рішень не поєднує нативну інтеграцію з генеративними моделями штучного інтелекту та інтерактивний графічний аналіз безпосередньо у вигляді веб-застосунку без необхідності встановлення додаткового програмного забезпечення.

Актуальність теми кваліфікаційної роботи обумовлюється необхідністю розробки сучасного веб-застосунку, що поєднує у єдиному інтерфейсі потоковий скринінг ринку криптовалют, інтерактивний технічний аналіз на основі свічкових графіків, інструменти ведення портфеля з автоматичним розрахунком PnL та аналітичну підтримку на базі великої мовної моделі GPT-4o-mini.

Метою кваліфікаційної роботи є розробка веб-застосунку Crypto Screener для моніторингу та аналітики криптовалютного портфеля, що забезпечує відображення ринкових даних у режимі реального часу, ведення персонального портфеля та інтелектуальну аналітичну підтримку користувача.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- Дослідити існуючі платформи моніторингу криптовалютного ринку та визначити їхні функціональні обмеження;

- Проаналізувати можливості REST API та WebSocket-інтерфейсів біржі Binance для отримання ринкових даних у реальному часі;

- Обґрунтувати вибір технологій та інструментів;

- Налаштувати архітектуру застосунку;

- Створити процес реєстрації та авторизації користувачів;

- Спроєктувати компонентну структуру застосунку та схему зберігання даних у Firebase Firestore;

- Забезпечити скринінг криптовалютних пар із підтримкою пошуку та вотчліста;

- Інтегрувати OpenAI API для реалізації модуля AI Trading Assistant;

- Перевірити функціональну можливість системи в лабора торії.

Об'єктом дослідження є процес моніторингу, відображення та аналізу криптовалютних активів у веб-середовищі в режимі реального часу.

Предметом дослідження є веб-застосунок Crypto Screener для моніторингу та аналітики криптовалютного портфеля, його компонентна архітектура, механізми інтеграції з біржовими API та інструментами штучного інтелекту.

Методи дослідження: аналіз предметної області та порівняльний аналіз існуючих рішень; методи компонентної та клієнтської веб-розробки на базі бібліотеки React; технології потокової передачі даних (REST API, WebSocket); хмарні сервіси Firebase; інтеграція API штучного інтелекту; функціональне тестування веб-застосунків.

Практичне значення отриманих результатів полягає у можливості використання розробленого застосунку Crypto Screener як інструменту персонального моніторингу криптовалютних активів, ведення портфеля та отримання аналітичної підтримки. Архітектура системи може слугувати основою для подальшого розвитку у напрямку повноцінної торгової платформи

з підтримкою технічних індикаторів, push-сповіщень та розширеного AI-аналізу.Результати роботи апроапм и бовано шляхом публікації тез доповіді на конференції Інформаційні технології – 2026: зб. тез XII Всеукраїнської науково-практичної конференції молодих учених, 14 трав. 2025 р., м. Київ / Київський столичний університет імені Бориса Грінченка. с. 65-69

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА СУЧАСНИХ РІШЕНЬ ДЛЯ МОНІТОРИНГУ КРИПТОВАЛЮТНИХ АКТИВІВ

1.1 Структура криптовалютного ринку та особливості його даних

Криптовалютний ринок є децентралізованою цифровою екосистемою, у межах якої здійснюється торгівля криптографічно захищеними цифровими активами. На відміну від традиційних фондових ринків, криптовалютний ринок функціонує цілодобово та без вихідних, що обумовлює безперервне генерування торгових даних і висуває підвищені вимоги до систем їх збору та відображення. Станом на 2024–2025 роки сукупна ринкова капіталізація перевищила позначку 2,5 трлн доларів США, а щоденний обсяг торгів лише на централізованих біржах сягає сотень мільярдів доларів [1].

Ключовим елементом ринкової інфраструктури є криптовалютні біржі – платформи, що забезпечують зіставлення ордерів покупців і продавців. Серед централізованих бірж (CEX) найбільшою за обсягом торгів є Binance, яка обробляє понад 40 % світового спот-обсягу. Binance надає розробникам відкритий публічний API для доступу до ринкових даних у режимі реального часу, що робить її основним джерелом даних для застосунків класу криптоскринер [2]. Децентралізовані біржі (DEX), такі як Uniswap або PancakeSwap, функціонують на базі смарт-контрактів і не мають централізованого API, що ускладнює їхню інтеграцію у системи реального часу.

З точки зору структури ринкових даних, кожен торговий інструмент (торгова пара) характеризується набором метрик, що постійно оновлюються. До основних відносяться: поточна ціна (lastPrice), зміна ціни за 24 години у відсотках (priceChangePercent), обсяг торгів за 24 години у базовій (volume) та котирувальній валюті (quoteVolume), найвища та найнижча ціна за добу (highPrice, lowPrice). Для технічного аналізу використовуються часові серії у форматі OHLCV (Open, High, Low, Close, Volume), відомі також як candlestick-

дані або дані японських свічок, що групуються за рівними часовими інтервалами – таймфреймами [3].

Особливістю криптовалютних ринкових даних є їхня висока частота оновлення та обсяг: лише на Binance одночасно торгуються понад 400 USDT-пар, кожна з яких генерує оновлення ціни та обсягу кілька разів на секунду. Ефективна обробка такого потоку даних у веб-застосунку вимагає застосування спеціалізованих протоколів передачі даних – зокрема WebSocket – та оптимізованих механізмів рендерингу інтерфейсу. Саме тому архітектура систем моніторингу криптовалютного ринку суттєво відрізняється від традиційних веб-застосунків і потребує окремого дослідження.

1.2 Концепція портфельного моніторингу та криптоскринінгу

Поняття криптоскринера (від англ. screener – фільтр, відбірник) охоплює клас програмних інструментів, що призначені для відображення, фільтрації та первинного аналізу великого масиву ринкових даних у реальному часі. На відміну від торгових терміналів, скринери не передбачають виконання торгових операцій і зосереджені виключно на інформаційно-аналітичній функції. Ключовими функціями криптоскринера є: відображення актуальних цін та показників зміни для широкого переліку активів; пошук і фільтрація за заданими критеріями; ранжування за обсягом торгів, зміною ціни чи іншими показниками [4].

Портфельний моніторинг є суміжною, але відмінною концепцією. Якщо скринер орієнтований на ринок загалом, то портфельний модуль зосереджується на підмножині активів, що перебувають у власності конкретного користувача. Центральним показником портфельного аналізу є PnL (Profit and Loss) – прибуток або збиток від утримання позиції, що розраховується як різниця між поточною ринковою вартістю активу та вартістю його придбання. Формально PnL для і-го активу визначається за формулою:

$$PnL_i = (P_t - P_\beta) \times Q_i,$$

де P_t – поточна ринкова ціна активу; P_β – ціна придбання (buyPrice); Q_i – кількість одиниць активу у портфелі (amount). Сумарний PnL портфеля є арифметичною сумою PnL за всіма позиціями [5].

Поєднання функцій скринера та портфельного модуля в єдиному застосунку дає змогу користувачу одночасно відстежувати стан ринку та власних вкладень, порівнювати динаміку конкретних активів із загальноринковими тенденціями та приймати більш обґрунтовані інвестиційні рішення. Саме така інтегрована модель покладена в основу розроблюваного застосунку Crypto Screener.

1.3 Технології передачі ринкових даних: REST API та WebSocket

Сучасні криптовалютні біржі надають розробникам два основні типи програмних інтерфейсів для доступу до ринкових даних: REST API та WebSocket API. Ці технології мають принципово різні архітектурні характеристики і застосовуються для вирішення різних задач у контексті розробки систем моніторингу реального часу.

REST API (Representational State Transfer) – це архітектурний стиль взаємодії клієнта і сервера на основі протоколу HTTP, за якого клієнт надсилає запит, а сервер повертає відповідь у форматі JSON. Взаємодія є односпрямованою та ініціюється виключно клієнтом. Для отримання актуальних даних клієнт змушений здійснювати повторні запити (polling) через визначені проміжки часу. Binance REST API надає доступ до агрегованої 24-годинної статистики через ендпоінт /api/v3/ticker/24hr та до OHLCV-даних (свічок) через ендпоінт /api/v3/klines із параметрами symbol, interval та limit [6].

WebSocket – це протокол повнодуплексного зв'язку, що встановлює постійне з'єднання між клієнтом і сервером через єдиний TCP-сокет. На відміну від REST, сервер може надсилати дані клієнту в будь-який момент без попереднього запиту. Binance WebSocket API надає потоки (streams) для отримання оновлень у реальному часі. Зокрема, потік {symbol}@kline_{interval} транслює оновлення поточної свічки щоразу, коли відбувається нова угода, що

забезпечує відображення актуального стану графіка з мінімальною затримкою [7].

Таблиця 1.1

Порівняльна характеристика REST API та WebSocket

Характеристика	REST API	WebSocket
Модель взаємодії	Запит–відповідь (Pull)	Постійне з'єднання (Push)
Ініціатор передачі	Клієнт	Сервер або клієнт
Затримка даних	Залежить від інтервалу polling	Мінімальна, у реальному часі
Навантаження на мережу	Вище (повторні запити)	Нижче (одне з'єднання)
Застосування у проєкті	Початкове завантаження монет та свічок	Оновлення свічок у реальному часі

У розроблюваному застосунку Crypto Screener обидві технології застосовуються у комплементарний спосіб. REST API використовується для початкового завантаження повного переліку монет та історичних OHLCV-даних при ініціалізації або зміні торгового інструменту. WebSocket-з'єднання застосовується для потокового оновлення активної свічки графіка без перезавантаження сторінки. Таке поєднання забезпечує оптимальний баланс між повнотою початкових даних та мінімальною затримкою при відображенні змін ціни.

1.4 Огляд та порівняльний аналіз існуючих рішень

Для визначення функціональних вимог до розроблюваного застосунку було проведено аналіз чотирьох найбільш поширених інструментів для моніторингу криптовалютних активів: CoinMarketCap, CoinStats, Delta та Blockfolio (FTX App). Аналіз здійснювався за такими критеріями: наявність ринкового скринера, підтримка ведення портфеля, наявність графічного аналізу, інтеграція інструментів штучного інтелекту, вимоги до реєстрації та тип клієнта (веб/мобільний).

1.4.1. CoinMarketCap

CoinMarketCap є однією з найбільших у світі платформ для агрегації криптовалютних даних, що забезпечує доступ до інформації про понад 20 000 криптовалют. Платформа надає розширені можливості скринінгу, включаючи фільтрацію за капіталізацією, обсягом, відсотковою зміною та категоріями активів. Водночас функціонал портфельного трекера є базовим і не підтримує розрахунку PnL у реальному часі без реєстрації. Технічний аналіз представлений лише спрощеними лінійними графіками. Підтримки інструментів штучного інтелекту на момент дослідження не виявлено [8].

1.4.2. CoinStats

CoinStats – мобільний та веб-застосунок для відстеження криптовалютного портфеля з підтримкою підключення реальних гаманців і бірж через API-ключі. Платформа вирізняється розвиненим портфельним модулем із детальним звітуванням про прибутки та збитки. Скринер реалізовано у базовому форматі без розширених фільтрів. Графічний аналіз обмежений; свічкові графіки із налаштуванням таймфреймів доступні лише у преміум-плані. AI-функціонал присутній у вигляді CoinStats AI, проте є частиною платної підписки [9].

1.4.3. Delta

Delta є спеціалізованим портфельним трекером із акцентом на мультибіржову агрегацію та автоматичну синхронізацію транзакцій. Застосунок підтримує понад 300 бірж і надає детальну звітність. Скринер ринку реалізовано у спрощеному вигляді. Зручний інтерфейс свічкових графіків відсутній; технічний аналіз виконується через редирект до зовнішніх платформ. Інтеграції з AI не виявлено [10].

1.4.4. Blockfolio (FTX App)

Blockfolio – один із перших мобільних трекерів портфеля, що набув широкої популярності завдяки простоті та зручності. Після поглинання компанією FTX та подальшого банкрутства останньої у 2022 році застосунок припинив повноцінну роботу. Його аналіз включено до огляду як приклад архітектурного підходу до мобільного скринера першого покоління. Ключовим недоліком була відсутність інтерактивного графічного аналізу та підтримки AI [11].

Таблиця 1.2

Порівняльний аналіз існуючих рішень

Критерій	CoinMarketCap	CoinStats	Delta	Blockfolio	Cryptoscreener
Ринковий скринер	+	Базовий	Базовий	-	+
Портфельний модуль	Базовий	+	+	+	+
Свічкові графіки	-	Платно	-	-	+
AI-аналітика	-	Платно	-	-	+
Дані в реальному часі	+	+	+	+	+
Реєстрація обов’язкова	-	+	+	+	+
Веб-клієнт	+	+	-	-	+
Безкоштовний AI	-	-	-	-	+

Проведений аналіз свідчить про те, що жодне з розглянутих рішень не поєднує у безкоштовному веб-форматі повноцінний скринер, інтерактивні свічкові графіки з налаштуванням таймфреймів та AI-аналітику. Саме ця функціональна ніша є цільовою для розроблюваного застосунку Crypto Screener.

1.5 Постановка задачі дослідження

На підставі проведеного аналізу предметної області, особливостей ринкових даних та існуючих рішень сформульовано наступну постановку задачі кваліфікаційної роботи.

Необхідно розробити веб-застосунок Crypto Screener для моніторингу та аналітики криптовалютного портфеля, що відповідає таким вимогам:

- функціональні вимоги: відображення актуальних ринкових даних по USDT-парах біржі Binance у вигляді таблиці з підтримкою пошуку та формування вотчліста; побудова інтерактивного свічкового графіка обраного інструменту з підтримкою таймфреймів 1m, 5m, 15m, 1h, 4h, 1d та потоковим оновленням через WebSocket; ведення персонального портфеля з можливістю додавання та видалення позицій, автоматичним розрахунком поточної вартості та PnL; автентифікація користувача через Firebase Authentication та збереження даних портфеля у Firebase Firestore; AI-аналітика на основі OpenAI API (модель gpt-4o-mini) у вигляді інтерактивного чату;
- нефункціональні вимоги: відгук інтерфейсу при оновленні даних не більше 5 секунд для REST-опитування; мінімальна затримка оновлення свічки через WebSocket; адаптивний інтерфейс із підтримкою десктопних та планшетних розмірів екрана; захист маршрутизації – доступ до дашборду лише для автентифікованих користувачів;
- технічний стек: React 19 як бібліотека для побудови інтерфейсу; Vite як інструмент збірки; Tailwind CSS v4 для стилізації; axios для HTTP-запитів; lightweight-charts v5 для рендерингу фінансових графіків; Firebase v12 (Authentication + Firestore) для автентифікації та збереження даних; OpenAI Node.js SDK v6 для взаємодії з GPT-4o-mini.

Таким чином, задача дослідження полягає у проєктуванні та повній програмній реалізації зазначеного застосунку з подальшим проведенням функціонального тестування та оцінки відповідності висунутим вимогам.

1.6 Висновки до розділу

У першому розділі проведено комплексний аналіз предметної області систем моніторингу криптовалютних активів. Встановлено, що криптовалютний ринок характеризується цілодобовим режимом роботи, високою волатильністю та значним обсягом генерованих торгових даних, що висуває особливі вимоги до архітектури систем реального часу.

Досліджено особливості двох основних технологій передачі ринкових даних – REST API та WebSocket. Визначено, що REST API доцільно застосовувати для початкового завантаження агрегованих даних, тоді як WebSocket забезпечує мінімальну затримку при потоковому оновленні. Обґрунтовано доцільність комбінованого застосування обох технологій у розроблюваному застосунку.

Проведено порівняльний аналіз чотирьох існуючих рішень – CoinMarketCap, CoinStats, Delta та Blockfolio. Виявлено, що жодна з платформ не забезпечує у безкоштовному веб-форматі одночасно повноцінний скринер, свічкові графіки та AI-аналітику. Сформульовано функціональні та нефункціональні вимоги до розроблюваного застосунку Crypto Screener, визначено технологічний стек та основні модулі системи.

РОЗДІЛ 2
ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ВИБІР ТЕХНОЛОГІЧНОГО СТЕКУ

2.1 Загальна архітектура застосунку

Застосунок Crypto Screener реалізовано за клієнтською SPA-архітектурою (Single Page Application), за якої весь рендеринг інтерфейсу відбувається на стороні браузера без перезавантаження сторінки. Такий підхід обумовлено специфікою задачі: відображення ринкових даних, що оновлюються кожні 5 секунд, вимагає локальних оновлень DOM-дерева, а не повного циклу запит–відповідь–рендеринг, характерного для серверних застосунків. Серверна частина у класичному розумінні в проєкті відсутня – її функції розподілені між двома зовнішніми сервісами: Binance API виступає джерелом ринкових даних, а Firebase слугує бекендом для автентифікації та збереження даних портфеля [12].

Архітектура застосунку складається з чотирьох функціональних шарів. Перший шар – шар представлення – реалізовано на бібліотеці React 19 і містить усі компоненти інтерфейсу користувача. Другий шар – шар управління станом – організовано засобами вбудованих хуків React (useState, useEffect) безпосередньо в компоненті Dashboard, який виступає кореневим компонентом-контейнером і передає дані у дочірні компоненти через пропси. Третій шар – шар зовнішніх сервісів – охоплює взаємодію з Binance REST API та WebSocket, а також з Firebase Authentication та Firestore. Четвертий шар – шар маршрутизації – реалізовано без зовнішньої бібліотеки: перемикання між екраном автентифікації та головним дашбордом здійснюється умовним рендерингом у кореневому компоненті App на основі стану автентифікації [13].

Точкою входу застосунку є файл main.jsx, що монтує кореневий компонент App у DOM-елемент з ідентифікатором root. Компонент App підписується на зміни стану автентифікації через обробник onAuthStateChanged Firebase і зберігає поточного користувача у стані user. Якщо змінна user має значення null

– відображається компонент Login; у разі успішної автентифікації – компонент Dashboard. Таким чином, захист маршрутів реалізовано на рівні умовного рендерингу без залучення додаткових залежностей.

Структура файлів проєкту організована за функціональним принципом. Директорія src/components містить п'ять компонентів: Auth, CoinTable, PriceChart, Portfolio та AIAssistant. Директорія src/pages містить два компоненти-сторінки: login та dashboard. Директорія src/services містить модуль api.js із функціями для роботи з Binance API. Файл firebase.js у кореневій директорії src ініціалізує Firebase-застосунок та експортує екземпляри служб auth та db для використання в інших модулях.

Таблиця 2.1

Структура компонентів застосунку Crypto Screener

Файл/Компонент	Тип	Призначення
App.jsx	Кореневий	Захист маршрутів, слухач onAuthStateChanged
pages/login.jsx	Сторінка	Екран входу та реєстрації
pages/dashboard.jsx	Сторінка	Головний контейнер, управління станом
components/Auth.jsx	Компонент	Форма входу/реєстрації та кнопка виходу
components/CoinTable.jsx	Компонент	Таблиця монет із пошуком і вотчлістом
components/Portfolio.jsx	Компонент	CRUD активів, розрахунок PnL
components/AIAssistant.jsx	Компонент	Інтерфейс чату з GPT-4o-mini
services/api.js	Сервіс	fetchCoins(), fetchChartData() через axios
firebase.js	Конфігурація	Ініціалізація Firebase, експорт auth та db

2.2 Обґрунтування технологічного стеку

Вибір технологічного стеку для розроблюваного застосунку зумовлено сукупністю вимог: необхідністю рендерингу великих таблиць даних, що часто оновлюються; потребою у відображенні фінансових графіків у реальному часі; необхідністю взаємодії з хмарними сервісами та зовнішніми API. Нижче наведено обґрунтування вибору кожної ключової технології.

2.2.1 React 19 та Vite

React є декларативною JavaScript-бібліотекою для побудови користувацьких інтерфейсів, розробленою компанією Meta. Ключовою перевагою React для застосунків класу скринер є його механізм Virtual DOM: при оновленні стану React порівнює попереднє та нове дерево компонентів і вносить у реальний DOM лише мінімально необхідні зміни. У контексті проєкту, де таблиця CoinTable містить понад 400 рядків і оновлюється кожні 5 секунд, це дає суттєву перевагу над прямою маніпуляцією DOM [14].

Для збірки проєкту використовується Vite версії 8.0.12 – сучасний інструмент збірки фронтенд-застосунків. Vite застосовує нативні ES-модулі браузера під час розробки, що забезпечує запуск dev-сервера практично миттєво незалежно від розміру проєкту. Для продуктивного збирання Vite використовує Rollup, що генерує оптимізовані бандли з підтримкою tree-shaking та code-splitting. Конфігурація проєкту підключає офіційний плагін @vitejs/plugin-react версії 6.0.1 для підтримки JSX-синтаксису та швидкого оновлення модулів (HMR) [15].

2.2.2 Tailwind CSS v4

Для стилізації інтерфейсу застосовано Tailwind CSS версії 4.3.0 – утилітарний CSS-фреймворк. На відміну від компонентних фреймворків (Bootstrap, Material UI), Tailwind не нав'язує готових стилізованих компонентів, а надає набір атомарних утилітарних класів, що безпосередньо відображають

CSS-властивості. Такий підхід дозволяє побудувати кастомний інтерфейс у темній кольоровій схемі (slate-950, slate-900, slate-800) без написання жодного рядка власного CSS. Tailwind CSS v4 підключено через плагін @tailwindcss/vite, що забезпечує автоматичне видалення невикористаних стилів на етапі збірки [16].

2.2.3 Firebase v12

Firebase є платформою Backend-as-a-Service (BaaS) від Google, що надає набір хмарних сервісів для розробки веб- та мобільних застосунків без розгортання власної серверної інфраструктури. У проєкті використовуються два сервіси Firebase SDK версії 12.14.0. Firebase Authentication забезпечує систему автентифікації з підтримкою реєстрації та входу за email і паролем через функції createUserWithEmailAndPassword та signInWithEmailAndPassword. Firebase Firestore є хмарною документо-орієнтованою NoSQL базою даних, що використовується для збереження даних портфеля користувача у колекції portfolios з доступом за унікальним ідентифікатором користувача uid [17].

Вибір Firebase як бекенд-платформи обумовлений декількома чинниками. По-перше, відсутністю необхідності у розгортанні та підтримці власного сервера, що суттєво спрощує архітектуру SPA-застосунку. По-друге, нативною підтримкою SDK для JavaScript та React, що дозволяє безпосередньо викликати методи аутентифікації та бази даних із компонентів React. По-третє, безкоштовним рівнем Spark, достатнім для функціонального тестування застосунку: 50 000 читань та 20 000 записів на добу.

2.2.4 Axios та Binance API

Для виконання HTTP-запитів до Binance REST API використовується бібліотека axios версії 1.17.0. Axios забезпечує зручний проміс-базований інтерфейс для HTTP-запитів, автоматичну серіалізацію та десеріалізацію JSON, а також уніфіковану обробку помилок через блок catch. Модуль services/api.js містить дві функції: fetchCoins() здійснює GET-запит до ендпоінту

/api/v3/ticker/24hr та фільтрує результат за суфіксом USDT, що дає перелік усіх активних спот-пар із котируванням у USDT; fetchChartData(symbol, interval) здійснює GET-запит до ендпоінту /api/v3/klines із параметрами symbol, interval та limit: 100 та перетворює масив свічок Binance у формат OHLCV, сумісний із бібліотекою lightweight-charts [18].Швидкий доступ до статей, прості можливості масштабування та функціональне розширення були факторами під час проєктування бази даних. Зрештою ви могли б розширити структуру бази колекціями для категорій, тегів, підписок, повідомлень або системи рекомендацій.

2.2.5 lightweight-charts v5

Для відображення фінансових графіків використовується бібліотека lightweight-charts версії 5.2.0, розроблена компанією TradingView. Ця бібліотека є спеціалізованим рішенням для рендерингу фінансових графіків на основі HTML5 Canvas, що забезпечує значно вищу продуктивність порівняно з SVG-рендерингом при відображенні великої кількості свічок. Версія 5 бібліотеки використовує новий API серій: замість методу addCandlestickSeries() застосовується метод addSeries(CandlestickSeries, options), де CandlestickSeries – іменований імпорт із пакету lightweight-charts [19].

2.2.6 OpenAI API (GPT-4o-mini)

Для реалізації модуля AI Trading Assistant використовується офіційний Node.js SDK компанії OpenAI версії 6.42.0. Взаємодія з API відбувається через метод openai.chat.completions.create(), якому передається масив повідомлень із системним промптом та запитом користувача. Системний промпт визначає роль моделі: «You are an AI crypto trading assistant. Analyze crypto markets professionally.». У якості моделі обрано gpt-4o-mini – полегшену версію GPT-4o, що забезпечує швидший час відповіді та нижчу вартість виклику при збереженні достатньої якості аналітичних відповідей для задач криптовалютного аналізу [20].

Таблиця 2.2

Технологічний стек застосунку Crypto Screener

Технологія	Версія	Призначення в проєкті
React	19.2.6	Бібліотека UI, Virtual DOM, хуки стану
Vite	8.0.12	Збірка, dev-сервер, HMR
Tailwind CSS	4.3.0	Утилітарна стилізація, темна схема
Firebase SDK	12.14.0	Authentication + Firestore (BaaS)
Axios	1.17.0	HTTP-запити до Binance REST API
lightweight-charts	5.2.0	Canvas-рендеринг свічкових графіків
OpenAI SDK	6.42.0	Виклик GPT-4o-mini через API
Binance REST API	v3	/api/v3/ticker/24hr, /api/v3/klines
Binance WebSocket	wss://	{symbol}@kline_{interval} – свічки real-time

2.3. Проєктування системи автентифікації та захисту маршрутів

Система автентифікації є одним із ключових архітектурних елементів застосунку, оскільки визначає не лише механізм ідентифікації користувача, але й логіку доступу до персоналізованих даних – зокрема збереженого портфеля. При проєктуванні цього модуля розглядалося кілька підходів: реалізація власної серверної автентифікації з JWT-токенами, використання стороннього OAuth-провайдера (Google Sign-In, GitHub) або делегування автентифікації хмарному BaaS-сервісу. Вибір зупинився на Firebase Authentication з провайдером email/password, оскільки цей варіант не потребує розгортання власного сервера,

нативно інтегрується з Firestore та забезпечує безпечне зберігання токенів на стороні клієнта засобами самої платформи [21].

З точки зору архітектури, відповідальність за автентифікацію розподілено між двома компонентами. Компонент App відіграє роль охоронця маршрутів: він підписується на зміни стану сесії та умовно відображає або форму входу, або головний дашборд. Компонент Auth зосереджує в собі логіку взаємодії з Firebase SDK – реєстрацію, вхід та вихід. Такий розподіл відповідає принципу єдиної відповідальності: App не знає деталей автентифікації, Auth не знає, що відображатиметься після входу.

Підписка на зміни стану автентифікації реалізована через хук useEffect з порожнім масивом залежностей, що означає одноразове виконання при монтуванні компонента. Функція onAuthStateChanged Firebase повертає функцію відписки, яку повертає useEffect як функцію очищення. Це критично важливо: без відписки при демонтуванні компонента виникає витік пам'яті, оскільки Firebase продовжує викликати обробник навіть після видалення компонента з дерева. Стан user, що зберігається у useState, приймає або об'єкт користувача Firebase (якщо сесія активна), або null (якщо користувач не автентифікований або вийшов). Умовне відображення реалізоване через простий оператор: якщо user дорівнює null – рендериться Login, інакше – Dashboard

Рис. 2..Схема архітектури системи автентифікації

Лістинг 2.1. Реалізація захисту маршрутів у компоненті App.jsx

```
      useEffect(() => {
        const unsubscribe = onAuthStateChanged(auth, (currentUser)
=> {
          setUser(currentUser);
        });
        return () => unsubscribe();
      }, []);

      if (!user) return <Login />;
      return <Dashboard />;
```

Компонент Auth реалізує три операції з Firebase SDK: createUserWithEmailAndPassword для реєстрації нового облікового запису, signInWithEmailAndPassword для входу існуючого користувача та signOut для завершення сесії. Усі три функції є асинхронними та обгорнуті у блоки try/catch. Це важливо, оскільки Firebase генерує різні типи помилок залежно від ситуації: auth/user-not-found при вході неіснуючого акаунта, auth/wrong-password при помилковому паролі, auth/email-already-in-use при повторній реєстрації. Перехоплення помилки через err.message дозволяє відображати конкретне повідомлення замість технічного коду помилки.

Компонент є умовно двоваріантним залежно від пропсу user. Якщо користувач автентифікований – відображається мінімалістична панель із email-адресою та кнопкою Logout. Якщо сесія відсутня – відображається форма з двома полями (email, password) та двома кнопками (Login та Register). Обидві кнопки викликають різні функції Firebase, але використовують ті самі поля вводу, що спрощує інтерфейс і не вимагає окремих сторінок для входу та реєстрації.

Варто окремо зазначити механізм персистентності сесії. Firebase Authentication за замовчуванням зберігає токен у localStorage браузера, що означає збереження сесії між закриттями вкладки та перезапусками браузера. При наступному відкритті застосунку onAuthStateChanged автоматично отримує збережений токен та викликає обробник із об'єктом користувача без жодних додаткових дій з боку розробника. Це суттєво покращує користувацький досвід:

після первинної реєстрації або входу користувачу не потрібно повторно вводити облікові дані при кожному відкритті застосунку [21].

2.4 Проєктування модуля отримання ринкових даних

Модуль отримання ринкових даних є технічно найскладнішим компонентом застосунку, оскільки поєднує два принципово різних механізми передачі даних. На етапі проєктування ключовим архітектурним рішенням стало розмежування між даними, що потребують повного оновлення (таблиця всіх монет), та даними, що оновлюються інкрементально в режимі реального часу (активна свічка графіка). Це розмежування і визначило застосування різних технологій для різних задач.

Перший механізм – циклічне опитування REST API – використовується для оновлення таблиці скринера. Функція fetchCoins() у модулі services/api.js формує GET-запит до публічного ендпоінту Binance /api/v3/ticker/24hr. Цей ендпоінт повертає масив об'єктів із 24-годинною статистикою для кожної торгової пари, доступної на біржі. Оскільки Binance підтримує торгівлю у різних котирувальних валютах (USDT, BTC, ETH, BNB), відповідь фільтрується за умовою coin.symbol.endsWith('USDT'), що залишає лише пари з котируванням у USDT. Загальна кількість таких пар перевищує 400 позицій. Кожен об'єкт містить поля: symbol (назва пари), lastPrice (поточна ціна), priceChangePercent (зміна за 24 години у відсотках) та quoteVolume (обсяг торгів у USDT за добу). Виклик функції організовано через setInterval з інтервалом 5 000 мс – оновлення відбувається кожні 5 секунд незалежно від дій користувача [22].

Рішення використовувати REST-опитування з інтервалом 5 секунд для таблиці монет, а не індивідуальні WebSocket-потоки, є свідомим компромісом. Підписка на 400+ окремих потоків через WebSocket потребувала б значних ресурсів браузера та складного управління з'єднаннями. Натомість один запит до агрегованого ендпоінту кожні 5 секунд є достатнім для скринера, де користувач зазвичай не потребує оновлень частіше ніж раз на кілька секунд.

Лістинг 2.2. Реалізація функції fetchCoins у модулі services/api.js.

```
const BASE_URL = 'https://api.binance.com';

export const fetchCoins = async () => {
  try {
    const response = await
axios.get(`${BASE_URL}/api/v3/ticker/24hr`);
    return response.data.filter(
      (coin) => coin.symbol.endsWith('USDT')
    );
  } catch (error) {
    console.log(error);
    return [];
  }
};
```

Другий механізм – функція fetchChartData(symbol, interval) – призначена для завантаження історичних OHLCV-даних при ініціалізації графіка або зміні торгового інструменту чи таймфрейму. Функція звертається до ендпоінту /api/v3/klines із трьома параметрами: symbol (наприклад, BTCUSDT), interval (один із шести підтримуваних таймфреймів: 1m, 5m, 15m, 1h, 4h, 1d) та limit зі значенням 100 – кількість свічок для завантаження. Binance повертає масив, де кожен елемент є масивом із 12 полів у фіксованому порядку: openTime, open, high, low, close, volume та шість додаткових технічних полів. Функція перетворює цей формат у масив об'єктів {time, open, high, low, close}, де time розраховується як openTime / 1000 для переведення мілісекунд у секунди – формат, якого вимагає бібліотека lightweight-charts [22].

Лістинг 2.3. Реалізація функції fetchChartData у модулі services/api.js

```
export const fetchChartData = async (symbol, interval) => {
  try {
    const response = await
axios.get(`${BASE_URL}/api/v3/klines`, {
```

```
        params: { symbol, interval, limit: 100 },
      });
      return response.data.map((candle) => ({
        time:  candle[0] / 1000,
        open:  parseFloat(candle[1]),
        high:  parseFloat(candle[2]),
        low:   parseFloat(candle[3]),
        close: parseFloat(candle[4]),
      }));
    } catch (error) {
      console.log(error);
      return [];
    }
  };
```

Третій механізм – WebSocket-з'єднання – відповідає за потокове оновлення активної свічки графіка без перезавантаження всього масиву даних. З'єднання встановлюється з потоком Binance за адресою wss://stream.binance.com:9443/ws/{symbol}@kline_{interval}. Цей потік транслює повідомлення щоразу, коли на біржі відбувається нова угода по обраній парі в межах поточного часового інтервалу. Повідомлення містить об'єкт k (kline) з усіма полями поточної незавершеної свічки: t (openTime), o (open), h (high), l (low), c (close).

Обробник ws.onmessage витягує ці значення та формує об'єкт newCandle. Оновлення стану chartData відбувається через функціональну форму setChartData, що отримує попередній стан як аргумент. Масив копіюється оператором spread, після чого останній елемент замінюється новою свічкою. Такий підхід є ключовим: він оновлює лише поточну незавершену свічку, а не перезаписує весь масив із 100 елементів. React, у свою чергу, завдяки механізму Virtual DOM, перерендерить лише ту частину компонента PriceChart, що відповідає зміненим даним. WebSocket-з'єднання закривається при кожній зміні selectedCoin або timeframe, а нове – відкривається для оновленого потоку. Це реалізовано через функцію очищення return () => ws.close() у хуку useEffect [23].

Лістинг 2.4. WebSocket-з'єднання для потокового оновлення свічки у Dashboard.jsx

```
useEffect(() => {
  const ws = new WebSocket(
    `wss://stream.binance.com:9443/ws/
     ${selectedCoin.toLowerCase()}@kline_${timeframe}`
  );
  ws.onmessage = (event) => {
    const kline = JSON.parse(event.data).k;
    const newCandle = {
      time:  kline.t / 1000,
      open:  parseFloat(kline.o),
      high:  parseFloat(kline.h),
      low:   parseFloat(kline.l),
      close: parseFloat(kline.c),
    };
    setChartData(prev => {
      const updated = [...prev];
      updated[updated.length - 1] = newCandle;
      return updated;
    });
  };
  return () => ws.close();
}, [selectedCoin, timeframe]);
```

Слід зазначити, що зміна selectedCoin або timeframe ініціює два паралельних процеси: повторний виклик fetchChartData для завантаження свіжих 100 свічок через REST API та перевстановлення WebSocket-з'єднання на новий потік. Ці два процеси запускаються двома окремими хуками useEffect, що підписані на одні й ті самі залежності [selectedCoin, timeframe]. Такий підхід забезпечує узгодженість даних: у момент перемикання інструменту або таймфрейму користувач бачить актуальні історичні дані та відразу починає отримувати потокові оновлення для нового потоку.

2.5 Проєктування модуля портфеля та збереження даних у Firestore

Портфельний модуль є функціонально найбагатшим з точки зору взаємодії з користувачем: він поєднує форму введення даних, динамічне відображення розрахованих показників та персистентне збереження стану у хмарній базі даних. При проєктуванні цього модуля важливим рішенням стало визначення, де

зберігати стан портфеля – локально у компоненті Portfolio або централізовано у Dashboard. Вибір зупинився на централізованому підході: масив portfolio зберігається у стані Dashboard та передається у Portfolio через пропси. Це рішення обумовлено тим, що агреговані показники – сукупна вартість портфеля (portfolioValue) та загальний PnL (portfolioPnL) – відображаються у заголовку Dashboard незалежно від того, чи розгорнуто компонент Portfolio на екрані. Якби стан зберігався всередині Portfolio, Dashboard не мав би до нього доступу без складного механізму підйому стану або зовнішнього стор-менеджера [24].

Кожна позиція портфеля представлена об'єктом з трьох полів. Поле symbol зберігає назву торгової пари у форматі BTCUSDT – саме у такому форматі Binance API повертає дані, що дозволяє напряму зіставляти позицію портфеля з відповідним рядком у масиві coins без додаткових перетворень. Поле amount зберігає кількість монет як числове значення з плаваючою крапкою. Поле buyPrice зберігає ціну придбання в USDT. Розрахунок поточної вартості позиції виконується множенням поточної ціни lastPrice з масиву coins на кількість amount. Показник PnL окремої позиції розраховується як різниця між поточною вартістю та вартістю придбання за формулою: pnl = (parseFloat(coin.lastPrice) × item.amount) – (item.buyPrice × item.amount). Оскільки обидва значення обчислюються динамічно при кожному рендерингу, вони автоматично відображають актуальний стан ринку щоразу, коли масив coins оновлюється після REST-запиту.

Операція додавання нового активу реалізована у функції addAsset компонента Portfolio. Функція спочатку перевіряє, що всі три поля форми заповнені, після чого формує об'єкт newAsset та додає його до масиву portfolio через оператор spread: setPortfolio([...portfolio, newAsset]). Поле symbol перетворюється на верхній регістр через toUpperCase(), що дозволяє користувачу вводити назву пари в будь-якому регістрі – наприклад, «btcusdt» або «BtcUsdt» – і отримувати коректне зіставлення з даними Binance. Операція видалення реалізована через фільтрацію масиву за символом: setPortfolio(portfolio.filter(item => item.symbol !== symbol)). Після кожної з цих операцій оновлений масив

portfolio зберігається у Firestore, що гарантує синхронізацію між сесіями браузера [24].

Збереження та завантаження даних портфеля організовано за схемою «один документ на користувача» у колекції portfolios Firebase Firestore. Ідентифікатором документа слугує uid – унікальний рядковий ідентифікатор, який Firebase Authentication автоматично присвоює кожному зареєстрованому користувачу та гарантує його унікальність. При завантаженні Dashboard компонент зчитує документ через функцію getDoc(doc(db, 'portfolios', currentUser.uid)). Якщо документ існує та містить поле portfolio, масив відновлюється у стані React. Документ Firestore має таку структуру: колекція portfolios → документ з id=uid → поле portfolio типу Array, де кожен елемент є об'єктом {symbol, amount, buyPrice}.

Лістинг 2.5. Завантаження та відновлення портфеля з Firestore

```
      onAuthStateChanged(auth, async (currentUser) => {
        setUser(currentUser);
        if (!currentUser) return;
        const docRef = doc(db, 'portfolios', currentUser.uid);
        const docSnap = await getDoc(docRef);
        if (docSnap.exists() && docSnap.data().portfolio) {
          setPortfolio(docSnap.data().portfolio);
        }
});
```

Варто звернути увагу на порядок операцій при завантаженні: спочатку виконується підписка на onAuthStateChanged, і лише після отримання об'єкта currentUser здійснюється запит до Firestore. Цей порядок є важливим, оскільки запит до Firestore без автентифікованого користувача завершиться помилкою доступу – правила безпеки Firestore типово забороняють читання чужих даних. Таким чином, автентифікація є обов'язковою передумовою для роботи з

персональними даними портфеля, що відповідає принципу захисту даних за замовчуванням.

2.6 Проєктування модуля свічкового графіка

Компонент PriceChart відповідає за відображення фінансового свічкового графіка обраної торгової пари. З усіх компонентів застосунку він є найбільш технічно специфічним, оскільки інтегрує зовнішню бібліотеку, що керує власним DOM-деревом – Canvas-елементом – поза межами React. Така архітектура типова для високопродуктивних графічних бібліотек і потребує особливої уваги до управління життєвим циклом.

При виборі бібліотеки для побудови фінансових графіків розглядалося декілька варіантів: Chart.js із плагіном chartjs-chart-financial, Recharts (вже підключений у проєкті як залежність) та lightweight-charts від TradingView. Chart.js є універсальною бібліотекою, що підтримує різні типи графіків, проте її продуктивність при частому оновленні даних у реальному часі поступається спеціалізованим рішенням. Recharts базується на SVG-рендерингу, що добре підходить для статичних або рідко оновлюваних графіків, але при відображенні 100 свічок із WebSocket-оновленнями кожну секунду SVG демонструє помітні затримки рендерингу порівняно з Canvas. Lightweight-charts використовує апаратно-прискорений Canvas-рендеринг та розроблений спеціально для фінансових даних, що й зумовило його вибір [25].

Ініціалізація графіка відбувається всередині хука useEffect, що спрацьовує при кожній зміні пропсу data. Посилання chartContainerRef, отримане через useRef, вказує на DOM-елемент div фіксованої висоти 500 пікселів. Функція createChart() приймає цей елемент та об'єкт налаштувань: ширина встановлюється рівною clientWidth контейнера для адаптивного відображення, висота – 500 пікселів, колір фону – #0f172a (відповідає Tailwind slate-950), колір

тексту – #d1d5db (Tailwind gray-300), колір ліній сітки – #1e293b (Tailwind slate-800). Така кольорова схема забезпечує візуальну узгодженість між графіком та темним інтерфейсом застосунку.

Серія свічок ініціалізується через метод chart.addSeries(CandlestickSeries, options) – новий API, введений у lightweight-charts v5. У попередній версії (v4) для цього використовувався метод addCandlestickSeries(). Зміна API є принциповою: у v5 всі типи серій є іменованими імпортами з пакету, що покращує tree-shaking при збірці. Параметри серії визначають колірне оформлення: upColor та wickUpColor встановлені у #22c55e (Tailwind green-500) для зростаючих свічок, downColor та wickDownColor – у #ef4444 (Tailwind red-500) для спадних. Після встановлення даних через candlestickSeries.setData(data) викликається метод chart.timeScale().fitContent(), що автоматично масштабує осі так, щоб усі 100 свічок були рівномірно розподілені у видимій області.

Лістинг 2.6. Ініціалізація свічкового графіка у компоненті PriceChart.jsx

```
const chart = createChart(chartContainerRef.current, {
  width:  chartContainerRef.current.clientWidth,
  height: 500,
  layout: {
    background: { color: '#0f172a' },
    textColor:  '#d1d5db',
  },
  grid: {
    vertLines: { color: '#1e293b' },
    horzLines: { color: '#1e293b' },
  },
});
const candleSeries = chart.addSeries(CandlestickSeries, {
  upColor:        '#22c55e', downColor:        '#ef4444',
  borderUpColor:  '#22c55e', borderDownColor:  '#ef4444',
  wickUpColor:    '#22c55e', wickDownColor:    '#ef4444',
});
candleSeries.setData(data);
chart.timeScale().fitContent();
```

Адаптивність графіка забезпечується обробником події resize вікна браузера. При зміні ширини вікна викликається метод chart.applyOptions({ width: chartContainerRef.current.clientWidth }), що перераховує ширину Canvas без перестворення екземпляра графіка. Функція очищення useEffect виконує дві дії: видаляє обробник resize через removeEventListener та знищує екземпляр графіка через chart.remove(). Знищення екземпляра є обов'язковим, оскільки lightweight-charts виділяє ресурси GPU під Canvas-контекст, і без явного видалення при кожному оновленні пропсу data утворювалися б нові екземпляри поверх попередніх, що призводило б до візуальних артефактів та зростання споживання пам'яті.

2.7 Проєктування модуля AI Trading Assistant

Модуль AI Trading Assistant реалізує взаємодію між користувачем та великою мовною моделлю GPT-4o-mini через офіційний OpenAI JavaScript SDK. Ідея модуля полягає у наданні користувачу зручного текстового інтерфейсу для отримання аналітичних відповідей на запитання про криптовалютний ринок – безпосередньо всередині застосунку, без переходу на зовнішні сервіси. З функціональної точки зору це однотурнова модель взаємодії: користувач вводить запит, натискає кнопку Ask AI і отримує відповідь. Збереження контексту між запитами не реалізовано – кожен новий запит є незалежним зверненням до API.

При виборі моделі OpenAI ключовим критерієм була рівновага між якістю аналітичних відповідей та швидкістю відгуку. Модель gpt-4o є найпотужнішою на момент розробки, проте має вищу вартість виклику та більший середній час відповіді. Модель gpt-4o-mini забезпечує якість, достатню для аналізу криптовалютного ринку та відповіді на питання про динаміку активів, технічні індикатори або загальні ринкові тенденції, при цьому час відгуку суттєво менший, що важливо для застосунку реального часу [26].

Ініціалізація клієнта OpenAI виконується поза тілом компонента на рівні модуля – єдиний екземпляр const openai = new OpenAI({...}) створюється один раз при першому завантаженні модуля та використовується у всіх подальших запитах. API-ключ зчитується зі змінної оточення VITE_OPENAI_API_KEY через import.meta.env – стандартний механізм Vite для роботи зі змінними оточення. Змінна визначається у файлі .env у кореневій директорії проєкту і не потрапляє до системи контролю версій. Параметр dangerouslyAllowBrowser: true є обов'язковим при використанні OpenAI SDK безпосередньо у браузері та сигналізує про усвідомлення факту, що API-ключ потенційно доступний через інструменти розробника браузера. Для навчального проєкту такий підхід є прийнятним; у виробничому середовищі запити до OpenAI API мали б проксуватися через власний серверний endpoint.

Стан компонента складається з трьох змінних: question зберігає поточний текст запиту у полі вводу, answer зберігає останню відповідь моделі, loading є булевим прапором, що керує відображенням індикатора очікування. Асинхронна функція askAI() спочатку перевіряє, що поле question не є порожнім, встановлює loading у true та формує запит до API через openai.chat.completions.create(). Масив messages складається з двох елементів: системного повідомлення з роллю system, що встановлює контекст моделі, та повідомлення користувача з роллю user, що містить введений запит. Системний промпт «You are an AI crypto trading assistant. Analyze crypto markets professionally.» формує рольовий контекст та стиль відповідей моделі. Вміст відповіді витягується з completion.choices[0].message.content та записується у стан answer, після чого loading встановлюється у false. У разі помилки мережі або перевищення ліміту запитів блок catch записує рядок 'Error connecting to AI' у answer, що забезпечує базову інформативність інтерфейсу при збоях.

Лістинг 2.7. Асинхронна функція надсилання запиту до OpenAI API

```
const askAI = async () => {
  if (!question) return;
  setLoading(true);
```

```
    try {
      const completion = await openai.chat.completions.create({
        model: 'gpt-4o-mini',
        messages: [
          {
            role:    'system',
            content: 'You are an AI crypto trading assistant.'
                   + ' Analyze crypto markets professionally.',
          },
          { role: 'user', content: question },
        ],
      });
      setAnswer(completion.choices[0].message.content);
    } catch (err) {
      setAnswer('Error connecting to AI');
    }
    setLoading(false);
  };
```

Інтерфейс компонента складається з поля вводу типу text, кнопки Ask AI та блоку відображення відповіді. Поле вводу прив'язане до стану question через значення value та обробник onChange. Блок відповіді має мінімальну висоту 120 пікселів та відображає або текст «Analyzing market...» при loading=true, або вміст answer при loading=false. Клас CSS whitespace-pre-wrap забезпечує збереження переносів рядків у відповіді моделі, що важливо для відображення структурованих відповідей із переліками та абзацами.

2.8 Висновки до розділу 2

У другому розділі виконано детальне проєктування архітектури застосунку Crypto Screener та обґрунтовано технологічні рішення для кожного з його модулів. Розглянуто загальну SPA-архітектуру з розподілом відповідальності між чотирма шарами: представлення, управління станом, зовнішніх сервісів та

маршрутизації. Доведено відповідність обраної архітектури вимогам застосунку з інтенсивним оновленням даних.

Для системи автентифікації обґрунтовано вибір Firebase Authentication як BaaS-рішення, що усуває потребу у власній серверній інфраструктурі. Детально описано механізм захисту маршрутів через реактивне відстеження стану onAuthStateChanged та умовний рендеринг, а також забезпечення персистентності сесії засобами localStorage Firebase.

Для модуля отримання ринкових даних розглянуто та обґрунтовано комбінований підхід: REST API з інтервалом опитування 5 000 мс для таблиці скринера та WebSocket-потік {symbol}@kline_{interval} для потокового оновлення активної свічки. Для модуля портфеля визначено структуру даних, логіку CRUD-операцій та схему збереження у Firestore за принципом «один документ на користувача». Для графічного модуля обґрунтовано вибір lightweight-charts v5 на основі Canvas-рендерингу та описано специфіку управління життєвим циклом зовнішньої бібліотеки у React. Для модуля AI Trading Assistant описано взаємодію з OpenAI API через модель gpt-4o-mini та структуру формування запитів із системним промптом.

Сукупність прийнятих архітектурних рішень забезпечує виконання усіх функціональних вимог, сформульованих у першому розділі, та є достатньою основою для переходу до практичної реалізації, що розглядається у третьому розділі.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ЗАСТОСУНКУ CRYPTO SCREENER

3.1 Реалізація модулів роботи з контентом

Модуль скринінгу є стартовою точкою взаємодії користувача із застосунком після автентифікації. Його основне призначення – надати огляд усього ринку USDT-пар біржі Binance в одному зручному інтерфейсі з можливістю швидкого пошуку та формування персонального списку спостереження. Реалізація модуля розподілена між компонентом Dashboard, що відповідає за логіку отримання та фільтрації даних, і компонентом CoinTable, що відповідає виключно за відображення.

Завантаження даних організовано через два послідовних механізми. Перший – початкове завантаження при монтуванні Dashboard: хук useEffect з порожнім масивом залежностей викликає функцію getCoins(), що звертається до fetchCoins() та записує результат у стан coins. Другий – циклічне оновлення через setInterval з інтервалом 5 000 мс, що запускається в окремому хуку useEffect. Такий розподіл на два хуки є свідомим архітектурним рішенням: перший хук гарантує негайне завантаження при відкритті сторінки, другий – підтримує актуальність даних надалі. Якби обидва завдання поєднати в один хук, між монтуванням та першим спрацюванням інтервалу виникала б пауза у 5 секунд, протягом якої таблиця залишалася б порожньою.

Фільтрація монет за введеним у поле пошуку рядком реалізована через обчислюване значення filteredCoins, що формується методом filter() масиву coins. Порівняння виконується без урахування регістру: обидва рядки приводяться до нижнього регістру через toLowerCase() перед порівнянням. Результат filteredCoins передається у компонент CoinTable через пропс coins. Оскільки filteredCoins перераховується при кожному рендерингу Dashboard – як при зміні стану search, так і при оновленні масиву coins – пошук і оновлення ринкових даних працюють незалежно та не конфліктують між собою.

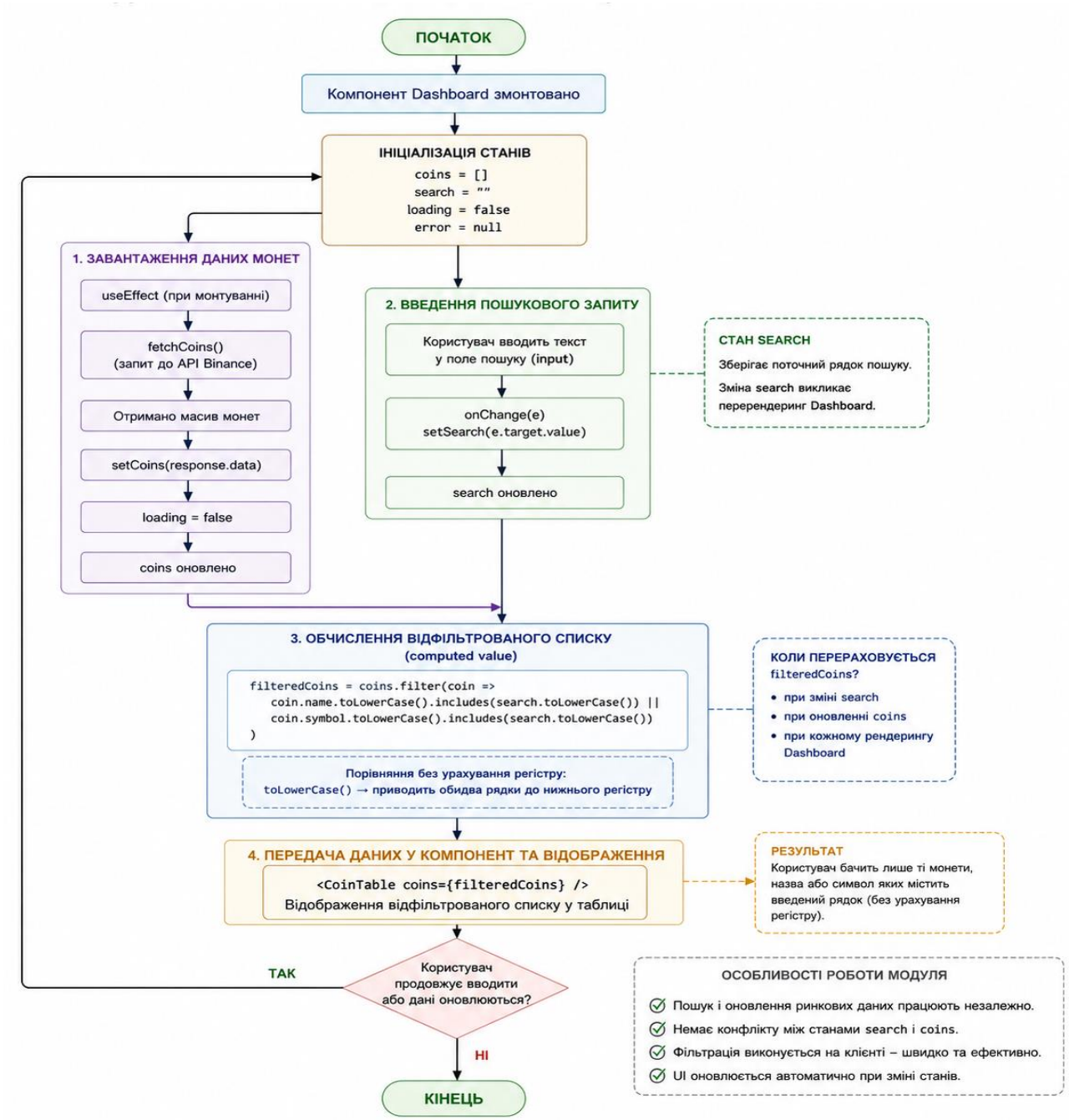


Рис. 3.1. Схема-фільтрації пошуку монет

Лістинг 3.1. Фільтрація монет за пошуковим запитом у Dashboard.jsx

```
      const filteredCoins = coins.filter((coin) =>
        coin.symbol.toLowerCase().includes(search.toLowerCase())
);
```

Компонент CoinTable приймає чотири пропси: coins (масив монет для відображення), onSelect (callback при кліку на рядок), watchlist (масив символів у списку спостереження) та toggleWatchlist (функція додавання/видалення з

вотчліста). Таблиця відображає п'ять стовпців: зірочка вотчліста, символ пари, поточна ціна, зміна за 24 години та обсяг торгів. Контейнер таблиці має фіксовану висоту 800 пікселів та вертикальне прокручування через CSS-клас overflow-y-auto, заголовок таблиці закріплено (sticky top-0) для зручності навігації при прокручуванні довгого списку.

Стовпець ціни форматується через parseFloat та toLocaleString(), що автоматично додає розділювачі тисяч залежно від локалі браузера. Стовпець зміни за 24 години відображається з двома знаками після коми та умовним кольором: зелений (#22c55e, клас text-green-400) для позитивних значень та червоний (#ef4444, клас text-red-400) для негативних. Стовпець обсягу торгів ділиться на 1 000 000 та округлюється до цілого, після чого відображається з суфіксом «M» (мільйони USDT), що робить великі числа зручними для сприйняття. Вотчліст реалізовано як масив рядків у стані Dashboard: кнопка зірочки відображає заповнену зірку (☆), якщо символ присутній у масиві watchlist, або контурну (☆) – якщо ні. Клік по зірочці викликає toggleWatchlist, що або видаляє символ із масиву через filter, або додає його через spread-оператор.

3.2 Реалізація модуля свічкового графіка з підтримкою таймфреймів

Модуль графіка реалізовано у компоненті PriceChart та керується зовні через два стани Dashboard: selectedCoin (обрана торгова пара) та timeframe (обраний таймфрейм). Вибір монети здійснюється кліком на рядок у CoinTable через callback onSelect, що встановлює новий selectedCoin. Вибір таймфрейму реалізовано через групу кнопок, що відображають масив ['1m', '5m', '15m', '1h', '4h', '1d']. Активний таймфрейм виділяється зеленим фоном (bg-green-500 text-black), неактивні – темно-сірим (bg-slate-800). При кліку по кнопці викликається setTimeframe(tf), що ініціює оновлення залежних хуків useEffect.

Компонент PriceChart отримує масив свічок data через пропс та ініціалізує або оновлює графік при кожній зміні цього масиву. Особливість реалізації полягає в тому, що хук useEffect повністю перестворює екземпляр графіка при

кожній зміні data. З першого погляду це може здатися неоптимальним рішенням – адже lightweight-charts підтримує метод setData для оновлення існуючої серії – однак у поєднанні з WebSocket-потоком, що оновлює окремі свічки інкрементально, повне перестворення при завантаженні нових 100 свічок є логічно коректним і достатньо швидким. Перестворення займає менше одного кадру рендерингу браузера та не помітне користувачу.

Взаємодія між модулем графіка та WebSocket організована таким чином, що компонент PriceChart не знає про існування WebSocket – він лише отримує пропс data та відображає його. Вся логіка WebSocket зосереджена у Dashboard: при надходженні нового повідомлення стан chartData оновлюється, що викликає перерендеринг PriceChart з новим значенням data, що своєю чергою ініціює перестворення графіка з оновленою останньою свічкою. Такий однонаправлений потік даних відповідає архітектурному принципу React і спрощує налагодження: стан завжди відображає те, що бачить користувач.

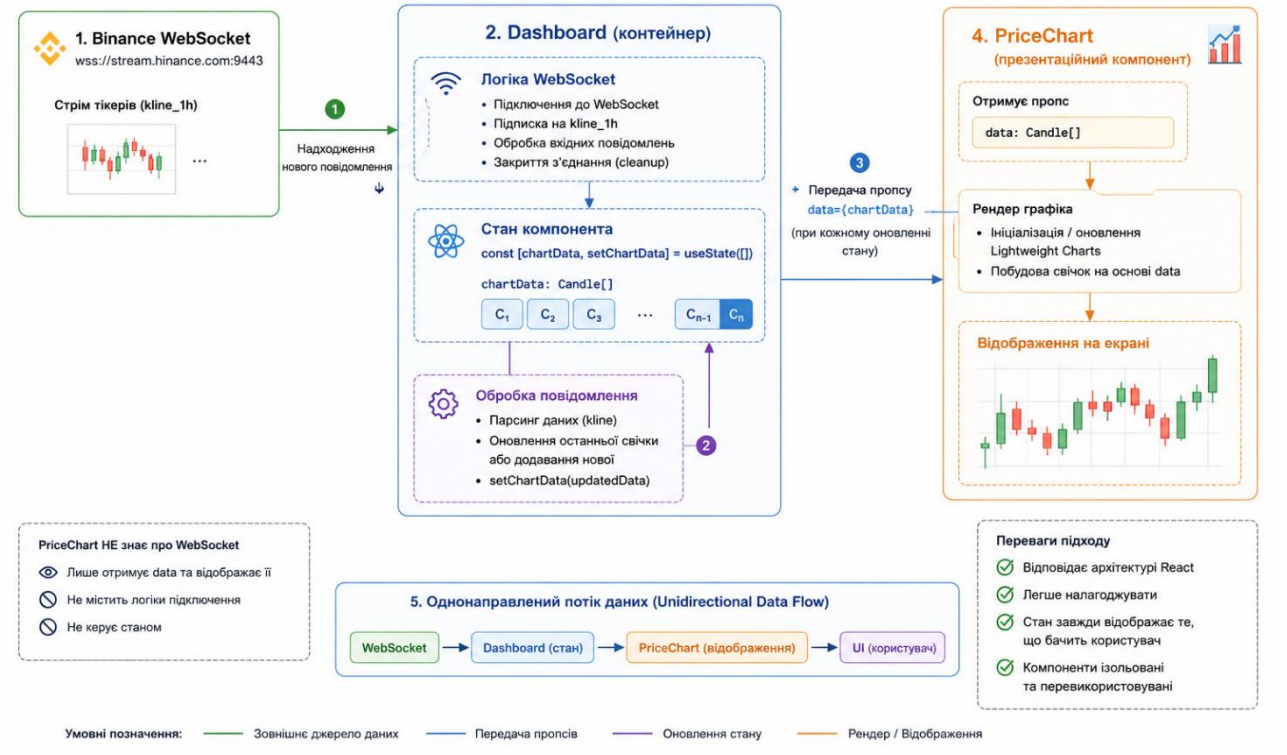


Рис. 3.2. Схема взаємодії компонентів графіка та WebSocket

Лістинг 3.2. Ініціалізація та оновлення графіка у PriceChart.jsx

```
      useEffect(() => {
        if (!chartContainerRef.current) return;
        const chart = createChart(chartContainerRef.current, {
          width:  chartContainerRef.current.clientWidth,
          height: 500,
          layout: { background: { color: '#0f172a' }, textColor:
'#d1d5db' },
          grid:   { vertLines: { color: '#1e293b' }, horzLines: {
color: '#1e293b' } },
        });
        const series = chart.addSeries(CandlestickSeries, {
          upColor: '#22c55e', downColor: '#ef4444',
          borderUpColor: '#22c55e', borderDownColor: '#ef4444',
          wickUpColor:   '#22c55e', wickDownColor:   '#ef4444',
        });
        if (data?.length) { series.setData(data); }
        chart.timeScale().fitContent();
        const handleResize = () => chart.applyOptions({
          width: chartContainerRef.current.clientWidth
        });
        window.addEventListener('resize', handleResize);
        return () => {
          window.removeEventListener('resize', handleResize);
          chart.remove();
        };
}, [data]);.
```

Зміна selectedCoin або timeframe у Dashboard ініціює два незалежних процеси. Перший – повторний виклик loadChart(), що через fetchChartData завантажує нові 100 свічок для нової пари або таймфрейму та записує їх у chartData. Другий – перевстановлення WebSocket-з'єднання: попереднє закривається через функцію очищення useEffect, нове відкривається для оновленого потоку {symbol}@kline_{interval}. Обидва процеси запускаються паралельно, оскільки підписані на ті самі залежності. На практиці REST-запит займає близько 200–400 мс, тоді як WebSocket-з'єднання встановлюється дещо швидше. Проте це не спричиняє конфліктів: REST завантажує масив із 100 свічок, WebSocket починає оновлювати лише останню, вже наявну свічку.

3.3 Реалізація модуля портфеля з розрахунком PnL

У сучасних вебзастосунках важливу роль відіграє не лише Компонент Portfolio реалізує повний цикл управління інвестиційними позиціями: введення нового активу, відображення поточного стану кожної позиції з розрахованими показниками та видалення позиції. Усі три операції взаємодіють зі станом portfolio, що зберігається у Dashboard, через пропси setPortfolio – це забезпечує централізованість даних та їхню доступність для розрахунку агрегованих показників у заголовку застосунку.

Форма додавання нового активу містить три поля вводу: symbol (торгова пара, наприклад BTCUSDT), amount (кількість монет) та buyPrice (ціна придбання). Поля типу number для amount та buyPrice використовують рядковий стан, що конвертується у число через parseFloat лише при формуванні об'єкта newAsset. Таке рішення дозволяє уникнути проблем із введенням десяткових чисел через браузерні обмеження полів type=number. Функція addAsset() перед додаванням перевіряє заповненість усіх трьох полів через умову !symbol || !amount || !buyPrice – якщо хоча б одне порожнє, операція скасовується без виведення помилки. Після успішного додавання всі три поля очищуються через відповідні setState.

Відображення кожної позиції у списку активів відбувається через метод map масиву portfolio. Для кожного елемента виконується пошук відповідного об'єкта у масиві coins через find за символом. Якщо монета не знайдена (наприклад, при першому рендерингу до завантаження coins), компонент повертає null, що React просто ігнорує. Коли монета знайдена, розраховуються: поточна вартість позиції (value = parseFloat(coin.lastPrice) × item.amount), сума вкладень (invested = item.buyPrice × item.amount) та показник PnL (pnl = value – invested). Позитивний PnL відображається зеленим кольором із префіксом «+», від'ємний – червоним без префікса. Кнопка Delete викликає

removeAsset(item.symbol), що фільтрує масив portfolio та зберігає оновлену версію у Firestore.

Лістинг 3.3. Розрахунок показників позиції портфеля у компоненті Portfolio.jsx

```
      const coin = coins.find((c) => c.symbol === item.symbol);
      if (!coin) return null;

      const currentPrice = parseFloat(coin.lastPrice);
      const value    = currentPrice * item.amount;
      const invested = item.buyPrice * item.amount;
const pnl      = value - invested;
```

Агреговані показники портфеля – сукупна вартість та загальний PnL – розраховуються у Dashboard через метод reduce масиву portfolio. Функція-редуктор для portfolioValue знаходить відповідну монету у масиві coins та додає до акумулятора добуток поточної ціни та кількості монет. Функція-редуктор для portfolioPnL обчислює різницю між поточною вартістю та вкладеннями для кожної позиції та підсумовує ці різниці. Обидва показники відображаються у правій частині заголовка застосунку та оновлюються синхронно з кожним циклом оновлення coins, тобто кожні 5 секунд. Значення форматуються через toFixed(2) – два знаки після коми – для відповідності стандарту відображення фінансових сум.

Збереження змін портфеля у Firestore відбувається при кожній операції додавання або видалення активу. Після оновлення стану portfolio у Dashboard викликається функція setDoc(doc(db, 'portfolios', user.uid), { portfolio: updatedPortfolio }), що перезаписує документ користувача у Firestore оновленим масивом. Такий підхід забезпечує повну синхронізацію між різними сесіями браузера та пристроями: наступного разу при вході в систему функція getDoc завантажить актуальний стан портфеля.

3.4 Реалізація модуля AI Trading Assistant

Модуль AI Trading Assistant у застосунку вирішує практичне завдання: надати користувачу можливість отримати аналітичну відповідь на довільне питання щодо криптовалютного ринку без необхідності залишати застосунок. Компонент AIAssistant є самодостатнім та не отримує жодних пропсів від Dashboard – це свідоме рішення, оскільки на момент реалізації модуль не має доступу до поточних ринкових даних і не передає їх у контекст запиту до моделі. Запит формується виключно на основі тексту, введеного користувачем.

З точки зору взаємодії з OpenAI API, кожен виклик askAI() є незалежним зверненням до endpoint /v1/chat/completions без збереження контексту між запитами. Масив messages кожного разу містить лише два елементи: системний промпт та поточний запит. Така однотурнова модель є достатньою для задач на кшталт «поясни, що таке рівень підтримки», «які чинники впливають на ціну Bitcoin» або «охарактеризуй ведмежий ринок». Для складніших сценаріїв, що вимагають контексту попередніх повідомлень, архітектуру слід було б розширити до збереження масиву messages між викликами – це є перспективним напрямком розвитку модуля.

Механізм відображення стану очікування реалізований через булеву змінну loading. Одразу після виклику askAI() встановлюється loading=true, що викликає рендеринг рядка «Analyzing market...» у блоці відповіді. Після отримання відповіді або обробки помилки loading повертається до false. Це забезпечує зрозумілий зворотний зв'язок: користувач розуміє, що запит обробляється, і не натискає кнопку повторно. Середній час відповіді моделі gpt-4o-mini при коротких запитах становить 2–5 секунд, що є прийнятним для аналітичного інструменту.

Лістинг 3.4. Повна реалізація функції askAI у компоненті AIAssistant.jsx

```
const askAI = async () => {
  if (!question) return;
  setLoading(true);
  try {
    const completion = await openai.chat.completions.create({
      model: 'gpt-4o-mini',
      messages: [
        {
          role:    'system',
          content: 'You are an AI crypto trading assistant.'
                 + ' Analyze crypto markets professionally.',
        },
        { role: 'user', content: question },
      ],
    });
    setAnswer(completion.choices[0].message.content);
  } catch (err) {
    console.log(err);
    setAnswer('Error connecting to AI');
  }
  setLoading(false);
};
```

3.5 Налаштування середовища розробки та збірка проєкту

Середовище розробки застосунку організовано на основі Vite 8 із плагіном @vitejs/plugin-react. Ініціалізація проєкту виконується командою npm create vite@latest з шаблоном react, після чого встановлюються залежності через npm install. Запуск dev-сервера здійснюється командою npm run dev, що запускає Vite у режимі розробки з підтримкою HMR (Hot Module Replacement) – змінений файл застосовується у браузері без перезавантаження сторінки та без втрати поточного стану застосунку. Це суттєво прискорює процес розробки при роботі з компонентами, що залежать від стану автентифікації або завантажених ринкових даних.

Конфіденційні дані – API-ключ OpenAI та конфігурація Firebase – зберігаються у файлі .env у кореневій директорії проєкту та не включаються до системи контролю версій через запис у .gitignore. Vite автоматично завантажує змінні з файлу .env та робить їх доступними у коді через об'єкт import.meta.env. Для забезпечення доступності змінних у браузері їхні назви мають починатися з префіксу VITE_ – наприклад, VITE_OPENAI_API_KEY. Конфігурація Firebase включає сім параметрів: apiKey, authDomain, projectId, storageBucket, messagingSenderId, appId та measurementId, що ідентифікують конкретний Firebase-проєкт (crypto-screener-2ea62).

Продуктивна збірка виконується командою npm run build, що запускає Rollup через Vite та генерує оптимізований бандл у директорії dist. Збірка включає мініфікацію JavaScript та CSS, tree-shaking (видалення невикористаного коду) та розбиття на чанки. Для попереднього перегляду продуктивної збірки локально використовується команда npm run preview, що запускає статичний сервер для директорії dist.

Таблиця 3.1

Команди управління проєктом

Команда	Призначення
npm install	Встановлення всіх залежностей проєкту
npm run dev	Запуск dev-сервера Vite з підтримкою HMR
npm run build	Продуктивна збірка у директорію dist/
npm run preview	Локальний перегляд продуктивної збірки
npm run lint	Статичний аналіз коду через ESLint

3.6 Тестування застосунку

Тестування застосунку Crypto Screener проводилося методом функціонального тестування – перевірки відповідності реалізованого функціоналу сформульованим вимогам. Для кожного модуля системи було визначено набір тестових сценаріїв, що охоплюють як основні, так і граничні

випадки використання. Тестування виконувалося у браузері Google Chrome версії 136 із відкритими інструментами розробника для моніторингу мережевих запитів, помилок консолі та стану React-компонентів через розширення React DevTools.

3.6.1 Тестування модуля автентифікації

Для модуля автентифікації визначено чотири тестових сценарії, що охоплюють реєстрацію, вхід, захист маршрутів та вихід із системи. Перший сценарій перевіряє успішну реєстрацію нового користувача: введення валідного email та паролю довжиною не менше 6 символів, натискання кнопки Register і очікування переходу до Dashboard. Другий сценарій – спроба реєстрації з вже існуючим email – має завершитися відображенням повідомлення про помилку Firebase auth/email-already-in-use. Третій сценарій перевіряє вхід існуючого користувача та відновлення збереженого портфеля. Четвертий – вихід через кнопку Logout та повернення до форми входу. Усі чотири сценарії завершилися успішно. Додатково перевірено персистентність сесії: після закриття та повторного відкриття вкладки браузера застосунок автоматично відновлює сесію без повторного введення облікових даних.

3.6.2 Тестування модуля скринінгу

Тестування скринера перевіряло коректність завантаження, фільтрації та оновлення даних. При відкритті Dashboard таблиця CoinTable заповнювалася даними протягом першого секунди після монтування компонента. Загальна кількість завантажених USDT-пар склала 412 позицій – це відповідає поточному переліку активних USDT-пар на Binance. Пошук за рядком «BTC» відфільтровував 8 пар, що містять цей рядок у символі (BTCUSDT, BTCDOMUSDT тощо). Пошук за порожнім рядком відновлював повний список. Оновлення цін у таблиці підтверджено спостереженням за змінами значень у стовпці Price протягом 30 секунд – за цей час відбулося 6 циклів оновлення з інтервалом 5 секунд.

3.6.3 Тестування модуля графіка

Для модуля графіка перевірялися: коректне відображення свічок при початковому завантаженні, перемикання таймфреймів та потокове оновлення. При відкритті Dashboard за замовчуванням відображається графік пари BTCUSDT із таймфреймом 1h та 100 свічками. Перемикання на таймфрейм 1m ініціювало перезавантаження даних та перевстановлення WebSocket-потоку; новий графік відображався протягом 300–500 мс. Клік на рядок іншої монети в таблиці (наприклад, ETHUSDT) коректно оновлював заголовок графіка та перезавантажував дані. Потокове оновлення свічки підтверджено спостереженням за зміною тіла та тіней останньої свічки на таймфреймі 1m протягом 2 хвилин. Адаптивність перевірено зміною ширини вікна браузера – графік коректно перемасштабовувався.

3.6.4 Тестування модуля портфеля

Тестування портфельного модуля охоплювало операції додавання, розрахунку та видалення позицій, а також перевірку синхронізації з Firestore. Додавання активу BTCUSDT з кількістю 0.1 та ціною придбання 50000 коректно відображало поточну вартість та PnL відповідно до поточної ціни з таблиці coins. При зміні ринкової ціни після чергового циклу оновлення показники PnL автоматично перераховувалися без будь-яких дій з боку користувача. Видалення позиції через кнопку Delete коректно прибирало рядок із таблиці та оновлювало агреговані показники у заголовку. Збереження у Firestore перевірено виходом із системи, повторним входом та підтвердженням відновлення раніше збережених позицій.

3.6.5 Тестування модуля AI Trading Assistant

Для модуля AI Assistant перевірялися: коректність надсилання запиту, відображення стану завантаження та обробка відповіді. Тестовий запит «What

factors influence Bitcoin price?» повертав структуровану відповідь моделі gpt-4o-mini протягом 3–6 секунд. Під час очікування відображався текст «Analyzing market...», після отримання відповіді – її повний текст зі збереженням форматування (переноси рядків). Тест із порожнім полем вводу підтвердив, що функція askAI() коректно переривається без звернення до API. Тест із навмисно некоректним API-ключем підтвердив відображення рядка 'Error connecting to AI' у блоці відповіді.

Таблиця 3.2

Зведені результати функціонального тестування

Тестовий сценарій	Очікуваний результат	Статус
Реєстрація нового користувача	Перехід до Dashboard	Пройдено ✓
Реєстрація з існуючим email	Повідомлення про помилку	Пройдено ✓
Вхід та відновлення портфеля	Портфель завантажено з Firestore	Пройдено ✓
Персистентність сесії	Сесія збережена після закриття вкладки	Пройдено ✓
Завантаження скринера	412 USDT-пар у таблиці	Пройдено ✓
Пошук у скринері	Фільтрація за символом у реальному часі	Пройдено ✓
Оновлення цін кожні 5 сек	6 циклів за 30 секунд	Пройдено ✓
Перемикання таймфреймів	Графік оновлено за 300–500 мс	Пройдено ✓

WebSocket-оновлення свічки	Остання свічка оновлюється в реальному часі	Пройдено ✓
Додавання активу до портфеля	Позиція з'являється, PnL розрахований	Пройдено ✓
Видалення активу з портфеля	Позиція видалена, Firestore оновлено	Пройдено ✓
Запит до AI Assistant	Відповідь GPT-4o-mini за 3–6 сек	Пройдено ✓
Порожній запит до AI	Запит до API не надсилається	Пройдено ✓
Адаптивність графіка	Коректне масштабування при зміні ширини вікна	Пройдено ✓

3.7 Аналіз результатів та виявлені обмеження

За результатами функціонального тестування всі 14 тестових сценаріїв завершено успішно. Застосунок Crypto Screener коректно виконує заявлені функції: відображає актуальні ринкові дані по 412 USDT-парах із оновленням кожні 5 секунд, будує інтерактивні свічкові графіки з підтримкою шести таймфреймів та потоковим оновленням через WebSocket, забезпечує ведення портфеля з автоматичним розрахунком PnL та хмарним збереженням у Firestore, надає AI-аналітику на основі моделі gpt-4o-mini.

Разом із тим у процесі тестування виявлено ряд обмежень поточної реалізації, що є орієнтирами для подальшого розвитку застосунку. По-перше, відсутність валідації введеного символу у формі портфеля: якщо користувач введе неіснуючу пару (наприклад, XXXXUSDT), позиція буде додана, але

компонент Portfolio повертатиме null для цього рядка, тобто позиція не відображатиметься. По-друге, стан вотчліста не зберігається у Firestore – при виході та повторному вході список спостереження скидається. По-третє, модуль AI Assistant не передає поточні ринкові дані у контекст запиту, що обмежує можливість отримання відповідей, прив'язаних до реальних цін конкретних монет у поточний момент.

Виявлені обмеження не впливають на коректність основного функціоналу застосунку та є природними рамками першої ітерації розробки. Їх усунення, поряд із впровадженням технічних індикаторів (RSI, MACD), push-сповіщень про цінові рівні та розширенням AI-контексту актуальними ринковими даними, визначає напрями подальшого вдосконалення системи.

ВИСНОВКИ

У рамках кваліфікаційної роботи розроблено веб-застосунок Crypto Screener — інтегрований інструмент для моніторингу криптовалютного ринку, аналізу портфеля та отримання аналітичної підтримки на основі генеративного штучного інтелекту. Поставлена мета досягнута повністю: спроєктовано архітектуру системи, реалізовано всі заплановані модулі та проведено функціональне тестування застосунку.

У першому розділі проведено комплексний аналіз предметної області. Досліджено структуру криптовалютного ринку та виявлено його ключові особливості: цілодобовий режим роботи, висока волатильність і значний обсяг торгових даних, що генеруються безперервно. Встановлено, що лише на біржі Binance одночасно торгується понад 400 активних USDT-пар, кожна з яких оновлює ринкові показники декілька разів на секунду. Проведено порівняльний аналіз чотирьох популярних платформ — CoinMarketCap, CoinStats, Delta та

Blockfolio. Виявлено, що жодна з них не поєднує у безкоштовному веб-форматі повноцінний скринер, інтерактивні свічкові графіки з налаштуванням таймфреймів та інтегровану AI-аналітику. Ця функціональна ніша визначила концепцію розроблюваного застосунку. Досліджено механізми передачі ринкових даних: REST API та WebSocket. Обґрунтовано доцільність їхнього комбінованого застосування — REST для початкового завантаження агрегованих даних і WebSocket для потокового оновлення активної свічки в реальному часі.

У другому розділі виконано детальне проєктування архітектури застосунку та обґрунтовано вибір технологічного стеку. SPA-архітектуру на базі React 19 обрано з огляду на ефективність механізму Virtual DOM при частому оновленні великих масивів даних. Vite 8 забезпечує швидкий запуск dev-сервера з підтримкою HMR та оптимізовану продуктивну збірку. Tailwind CSS v4 дозволив реалізувати цілісний темний інтерфейс без написання власного CSS. Firebase v12 у складі Authentication та Firestore усунув потребу у розгортанні власної серверної інфраструктури. Бібліотека lightweight-charts v5 від TradingView забезпечила Canvas-рендеринг свічкових графіків із мінімальними затримками. OpenAI SDK v6 із моделлю gpt-4o-mini реалізує аналітичну підтримку користувача. Спроєктовано п'ять функціональних модулів: автентифікації та захисту маршрутів, скринінгу ринкових даних, свічкового графіка, управління портфелем та AI Trading Assistant.

У третьому розділі описано практичну реалізацію застосунку та проведено його функціональне тестування. Реалізовано такі ключові технічні рішення:

- система автентифікації на Firebase Authentication із реактивним захистом маршрутів через onAuthStateChanged та персистентністю сесії засобами localStorage;
- модуль скринінгу, що завантажує дані 412 USDT-пар через ендпоінт /api/v3/ticker/24hr та оновлює їх циклічно з інтервалом 5 000 мс;

- свічковий графік на lightweight-charts v5 із підтримкою шести таймфреймів (1m, 5m, 15m, 1h, 4h, 1d) та потоковим оновленням незавершеної свічки через WebSocket-потік {symbol}@kline_{interval};
- портфельний модуль із CRUD-операціями, автоматичним розрахунком показника PnL у реальному часі та хмарним збереженням позицій у Firestore за схемою «один документ на користувача» (колекція portfolios, ідентифікатор uid);
- AI Trading Assistant на основі GPT-4o-mini з однотурновою моделлю взаємодії та системним промптом аналітика криптовалютного ринку.

Функціональне тестування охопило 14 тестових сценаріїв, що перевіряли коректність роботи всіх модулів. Усі сценарії завершено успішно. Зафіксовано такі характеристики продуктивності системи: перемикання таймфрейму графіка виконується за 300–500 мс, середній час відповіді AI Trading Assistant становить 3–6 секунд, що обумовлено архітектурними особливостями зовнішнього сервісу OpenAI і не є дефектом розробленого застосунку. Оновлення активної свічки через WebSocket відбувається з мінімальною затримкою, що визначається пропускною здатністю мережевого з'єднання користувача.

У процесі тестування виявлено ряд обмежень поточної реалізації, що є природними рамками першої ітерації розробки. По-перше, форма додавання активу до портфеля не валідує введений символ на відповідність реальним торговим парам Binance, що може призводити до появи позицій, для яких не знаходиться відповідний запис у масиві ринкових даних. По-друге, вотчліст зберігається виключно у локальному стані React та не синхронізується з Firestore, через що список спостереження скидається після виходу з системи. По-третє, модуль AI Trading Assistant не передає поточні ринкові дані у контекст запиту, що обмежує можливість отримання відповідей, прив'язаних до конкретних цін у поточний момент. По-четверте, API-ключ OpenAI передається у браузерному

середовищі, що є прийнятним для навчального проєкту, але потребує проксування через власний серверний endpoint у продуктивному розгортанні.

Перспективними напрямами подальшого розвитку застосунку є: збереження вотчліста у Firestore для забезпечення персистентності між сесіями; додавання валідації символу при введенні активу до портфеля із перевіркою наявності пари на Binance; розширення контексту AI-запитів поточними ринковими даними для більш точних аналітичних відповідей; впровадження технічних індикаторів технічного аналізу (RSI, MACD, Bollinger Bands) на графіку; реалізація цінових алертів із push-сповіщеннями при досягненні заданого рівня; проксування OpenAI API через власний серверний endpoint для захисту ключа у продуктивному середовищі; розробка мобільного застосунку на базі React Native із використанням тієї самої Firebase-інфраструктури.

Практичне значення розробленого застосунку полягає у можливості його використання як інструменту індивідуального трейдера або портфельного інвестора для відстеження динаміки криптовалютного ринку, ведення обліку позицій та отримання аналітичної підтримки на основі генеративного штучного інтелекту — без необхідності одночасного використання кількох окремих сервісів. Архітектурні рішення, реалізовані у роботі, зокрема комбінування REST API та WebSocket, застосування Firebase як BaaS-платформи та інтеграція OpenAI API у фронтенд-застосунок, можуть слугувати основою для подальшого розвитку у напрямку повноцінної торгової платформи.

Таким чином, у процесі виконання кваліфікаційної роботи вирішено всі поставлені завдання: проаналізовано предметну область та існуючі рішення, обґрунтовано технологічний стек, спроєктовано та реалізовано п'ять функціональних модулів застосунку, проведено функціональне тестування та визначено напрями подальшого розвитку. Розроблений застосунок Crypto Screener демонструє практичну можливість побудови сучасного інструменту фінансового моніторингу засобами клієнтських веб-технологій із залученням хмарних сервісів та генеративного штучного інтелекту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Басюк Т. М., Пасічник В. В. Основи інформаційних технологій. Львів : Новий Світ-2000, 2021. 390 с.
2. Биков В. Ю. Інформаційні технології у сучасному суспільстві. Київ : Академвидав, 2020. 224 с.
3. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System [Електронний ресурс]. – 2008. – 9 p. – URL: https://bitcoin.org/bitcoin.pdf (дата звернення: 11.05.2026).
4. Antonopoulos A. M. Mastering Bitcoin: Programming the Open Blockchain. 2nd ed. – O'Reilly Media, 2017. – 414 с.
5. Burniske C., Tatar J. Cryptoassets: The Innovative Investor's Guide to Bitcoin and Beyond. – McGraw-Hill Education, 2018. – 368 с. – ISBN 978-1260026672.
6. Murphy J. J. Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications. – New York Institute of Finance, 1999. – 576 с.
7. Markowitz H. Portfolio Selection: Efficient Diversification of Investments. 2nd ed. – Yale University Press, 1991. – 384 с.
8. Pardo R. The Evaluation and Optimization of Trading Strategies. 2nd ed. – Wiley, 2008. – 334 с.
9. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. – O'Reilly Media, 2020. – 350 с.
10. Wieruch R. The Road to React: Your Journey to Master Plain Yet Pragmatic React.js. – Self-published, 2022. – 246 с.
11. Flanagan D. JavaScript: The Definitive Guide. 7th ed. – O'Reilly Media, 2021. – 706 p.
12. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : PhD dissertation. – University of California, Irvine, 2000. – 162 p.

13. Fette I., Melnikov A. The WebSocket Protocol // RFC 6455. – IETF, 2011 [Електронний ресурс]. – URL: https://datatracker.ietf.org/doc/html/rfc6455 (дата звернення: 11.05.2026).

14. Sommerville I. Software Engineering. Pearson, 2021. 816 p.

15. Mell P., Grance T. The NIST Definition of Cloud Computing // NIST Special Publication. – 2011. – Vol. 800-145. – 7 p.

16. Moroney L. The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform. – Apress, 2017. – 368 p.

17. Tariq M. Firebase Cookbook: Over 70 Recipes to Help You Create Real-Time Web and Mobile Applications with Firebase. – Packt Publishing, 2017. – 348 p.

18. Brown T. B. et al. Language Models are Few-Shot Learners // Advances in Neural Information Processing Systems. – 2020. – Vol. 33. – P. 1877–1901.

19. Floridi L. AI as Agency Without Intelligence: On ChatGPT, Large Language Models, and Other Generative Models // Philosophy & Technology. – 2023. – Vol. 36, № 1. – P. 15.

20. Bommasani R. et al. On the Opportunities and Risks of Foundation Models // arXiv preprint. – 2021. – arXiv:2108.07258. – 214 p.

21. Aljawarneh S., Yassein M. B. A Conceptual Security Framework for Cloud Computing Issues // International Journal of Intelligent Information Technologies. – 2016. – Vol. 12, № 2. – P. 12–24.

22. OWASP Foundation. OWASP Top Ten Web Application Security Risks [Електронний ресурс]. – 2021. – URL: https://owasp.org/www-project-top-ten (дата звернення: 11.05.2026).

23. Binance API Documentation [Електронний ресурс]. – URL: https://binance-docs.github.io/apidocs/spot/en (дата звернення: 11.05.2026).

24. Binance WebSocket Streams Documentation [Електронний ресурс]. – URL: https://binance-docs.github.io/apidocs/spot/en/#websocket-market-streams (дата звернення: 11.05.2026).

25. lightweight-charts Documentation. TradingView [Електронний ресурс]. – URL: https://tradingview.github.io/lightweight-charts (дата звернення: 11.05.2026).

26. OpenAI API Documentation [Електронний ресурс]. – URL: https://platform.openai.com/docs (дата звернення: 11.05.2026).

27. Firebase Documentation. Google [Електронний ресурс]. – URL: https://firebase.google.com/docs (дата звернення: 11.05.2026).

28. Google Firestore Documentation [Електронний ресурс]. – URL: https://firebase.google.com/docs/firestore (дата звернення: 11.05.2026).

29. Tailwind CSS Documentation [Електронний ресурс]. – URL: https://tailwindcss.com/docs (дата звернення: 11.05.2026).

30 Hashko, A. O., & Bondarchuk, A. P. (2025). Automated method for verifying the correctness of the execution of smart contracts in the blockchain network. Сучасний захист інформації, (4), 37-43.

31. Bondarchuk, A., Onyshchenko, V., Hashko, A., Strazhnikov, A., & Korshun, N. (2025, October). Security difficulties and barriers in building decentralized blockchain bridges, solution methods. In Workshop on Cybersecurity Providing in Information and Telecommunication Systems (No. 4145, pp. 257-274). Germany.

32. Dmytro M. Bodnenko, Oleksandra V. Lokaziuk, Sergiy Radchenko, Volodymyr Proshkin, Serhiy D Shymchyk. The development and application of VBA macros in the teaching of science and mathematics. 8th Workshop for Young Scientists in Computer Science & Software Engineering (CS&SE@SW 2025), Kryvyi Rih, Ukraine, November 21, 2025

33 Гашко, А. О., Бондарчук, А. П., Трембовецький, М. П., & Чумак, О. І. (2025). Автоматизований метод перевірки правильності виконання смарт-контрактів в блокчейн Мережі. Телекомунікаційні та інформаційні технології, (1), 13-20.

34. Закон України «Про захист персональних даних» від 01.06.2010 № 2297-VI [Електронний ресурс]. URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 11.05.2026).

35. Abramov, V., Astafieva, M., Boiko, M., Bodnenko, D., Bushma, A., Vember, V., ... & Yaskevych, V. (2021). Theoretical and practical aspects of the use of mathematical methods and information technology in education and science.

36. Машкіна, І. В., Абрамов, В. О., Носенко, Т. І., Іваніченко, Є. В., & Яскевич, В. О. (2024). Навчально-методичний посібник до виконання і захисту кваліфікаційної роботи бакалавра спеціальностей 122 Комп'ютерні науки для здобувачів вищої освіти денної форми навчання.

ДОДАТОК А

Фрагменти програмного коду застосунку Cryptoscreener

```
import { useEffect, useState } from "react";

import { auth } from "./firebase";
import { onAuthStateChanged } from "firebase/auth";
import Login from "./pages/login";
import Dashboard from "./pages/dashboard";

export default function App() {
  const [user, setUser] = useState(null);

  useEffect(() => {
    const unsubscribe =
      onAuthStateChanged(
        auth,
        (currentUser) => {
          setUser(currentUser);
        }
      );

    return () => unsubscribe();
  }, []);

  if (!user) {
    return <Login />;
  }

return <Dashboard />;}
```

```
import { useEffect, useState } from "react";

import CoinTable from "../components/CoinTable";
import PriceChart from "../components/PriceChart";
import Auth from "../components/Auth";
import Portfolio from "../components/Portfolio";
import AIAssistant from "../components/AIAssistant";
import { auth, db } from "../firebase";

import {
  onAuthStateChanged,
} from "firebase/auth";

import {
  doc,
  getDoc,
} from "firebase/firestore";

import {
  fetchCoins,
  fetchChartData,
} from "../services/api";

export default function Dashboard() {
  const [coins, setCoins] = useState([]);
  const [search, setSearch] = useState("");
```

```
const [selectedCoin, setSelectedCoin] =
  useState("BTCUSDT");

const [chartData, setChartData] =
  useState([]);

const [timeframe, setTimeframe] =
  useState("1h");

const [watchlist, setWatchlist] =
  useState([]);

const [user, setUser] =
  useState(null);

// PORTFOLIO
const [portfolio, setPortfolio] =
  useState([
    {
      symbol: "BTCUSDT",
      amount: 0.5,
      buyPrice: 60000,
    },
    {
      symbol: "ETHUSDT",
      amount: 2,
      buyPrice: 2500,
    },
  ]);
```

```
// WATCHLIST
const toggleWatchlist = (symbol) => {
  if (watchlist.includes(symbol)) {
    setWatchlist(
      watchlist.filter(
        (coin) => coin !== symbol
      )
    );
  } else {
    setWatchlist([
      ...watchlist,
      symbol,
    ]);
  }
};

// FILTERED COINS
const filteredCoins = coins.filter((coin) =>
  coin.symbol
    .toLowerCase()
    .includes(search.toLowerCase())
);

// LOAD COINS
useEffect(() => {
  const getCoins = async () => {
    const data = await fetchCoins();
    setCoins(data);
  };
```

```
    getCoins();
  }, []);

  // LOAD CHART
  useEffect(() => {
    loadChart();
  }, [selectedCoin, timeframe]);

  // LIVE CHART
  useEffect(() => {
    const ws = new WebSocket(
      `wss://stream.binance.com:9443/ws/${selectedCoin.toLowerCase()}
@kline_${timeframe}`
    );

    ws.onmessage = (event) => {
      const message = JSON.parse(event.data);

      const kline = message.k;

      const newCandle = {
        time: kline.t / 1000,
        open: parseFloat(kline.o),
        high: parseFloat(kline.h),
        low: parseFloat(kline.l),
        close: parseFloat(kline.c),
      };

      setChartData((prevData) => {
        if (!prevData.length) {
```

```
        return [newCandle];
      }

      const updated = [...prevData];

      updated[updated.length - 1] =
        newCandle;

      return updated;
    });
  };

  return () => ws.close();
}, [selectedCoin, timeframe]);

// LIVE ALL COINS
useEffect(() => {
  const updateCoins = async () => {
    try {
      const updatedCoins =
        await fetchCoins();

      setCoins([...updatedCoins]);
    } catch (error) {
      console.log(error);
    }
  };

  updateCoins();
```

```
  const interval = setInterval(() => {
    updateCoins();
  }, 5000);

  return () => clearInterval(interval);
}, []);

// AUTH
useEffect(() => {
  const unsubscribe =
    onAuthStateChanged(
      auth,
      async (currentUser) => {
        setUser(currentUser);

        if (!currentUser) return;

        const docRef = doc(
          db,
          "portfolios",
          currentUser.uid
        );

        const docSnap =
          await getDoc(docRef);

        if (docSnap.exists()) {
          const data =
            docSnap.data();
```

```
        if (data.portfolio) {
          setPortfolio(
            data.portfolio
          );
        }
      }
    );

  return () => unsubscribe();
}, []);

// LOAD CHART DATA
const loadChart = async () => {
  const data = await fetchChartData(
    selectedCoin,
    timeframe
  );

  setChartData(data);
};

// PORTFOLIO VALUE
const portfolioValue =
  portfolio.reduce((total, item) => {
    const coin = coins.find(
      (c) => c.symbol === item.symbol
    );

    if (!coin) return total;
```

```
    return (
      total +
      parseFloat(coin.lastPrice) *
        item.amount
    );
  }, 0);

// PORTFOLIO PNL
const portfolioPnL =
  portfolio.reduce((total, item) => {
    const coin = coins.find(
      (c) => c.symbol === item.symbol
    );

    if (!coin) return total;

    const currentValue =
      parseFloat(coin.lastPrice) *
      item.amount;

    const invested =
      item.buyPrice * item.amount;

    return (
      total +
      (currentValue - invested)
    );
  }, 0);
```

```
return (
  <div className="min-h-screen bg-slate-950 text-white p-6">
    {/* HEADER */}
    <div className="flex justify-between items-center mb-6">
      <h1 className="text-4xl font-bold">
        Crypto Screener
      </h1>

      <Auth user={user} />

      {/* PORTFOLIO */}
      <div className="bg-slate-900 px-4 py-2 rounded-xl">
        <p className="text-sm text-gray-400">
          Portfolio Value
        </p>

        <p className="text-2xl font-bold">
          ${portfolioValue.toFixed(2)}
        </p>

        <p
          className={`text-sm ${
            portfolioPnL >= 0
              ? "text-green-400"
              : "text-red-400"
          }`}
        >
          PNL: ${portfolioPnL.toFixed(2)}
        </p>
      </div>
```

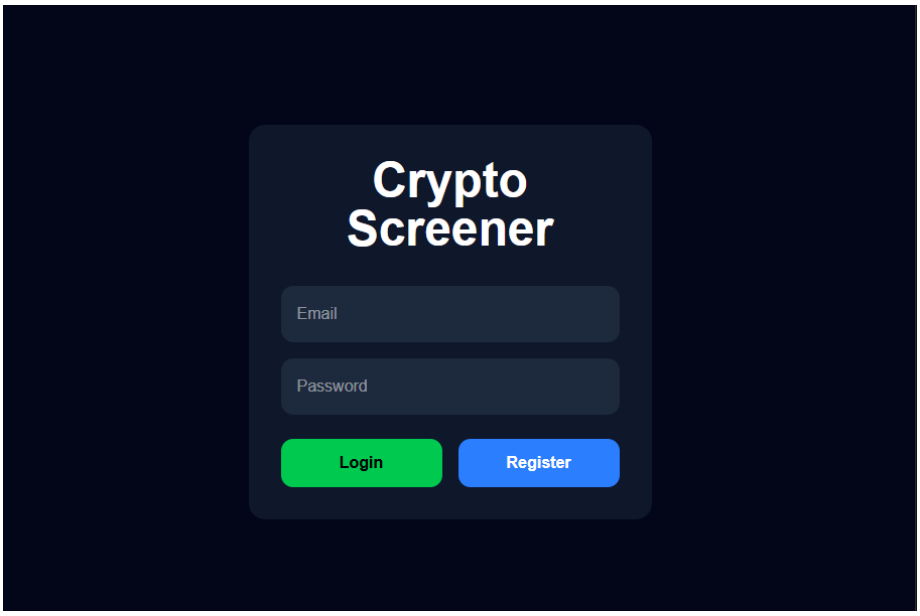
```
</div>

{/* SEARCH */}
<input
  type="text"
  placeholder="Search coin..."
  className="w-full mb-6 p-3 rounded-lg bg-slate-900 outline-none"
  value={search}
  onChange={(e) =>
    setSearch(e.target.value)
  }
/>

{/* MAIN */}
<div className="grid grid-cols-1 lg:grid-cols-3 gap-6">
  {/* TABLE */}
  <div className="lg:col-span-1">
    <CoinTable
      coins={filteredCoins}
      onSelect={setSelectedCoin}
      watchlist={watchlist}
      toggleWatchlist={toggleWatchlist}
    />
  </div>

  {/* CHART */}
  <div className="lg:col-span-2">
    <div className="flex justify-between items-center mb-4">
      <h2 className="text-2xl font-bold">
        {selectedCoin}
```

```
  </h2>

  {/* TIMEFRAMES */}
  <div className="flex gap-2">
    {[
      "1m",
      "5m",
      "15m",
      "1h",
      "4h",
      "1d",
    ].map((tf) => (
      <button
        key={tf}
        onClick={() =>
          setTimeframe(tf)
        }
        className={`px-4 py-2 rounded-lg transition ${
          timeframe === tf
            ? "bg-green-500 text-black"
            : "bg-slate-800 hover:bg-slate-700"
        }`}
      >
        {tf}
      </button>
    ))}
  </div>
</div>

{/* CHART */}
```

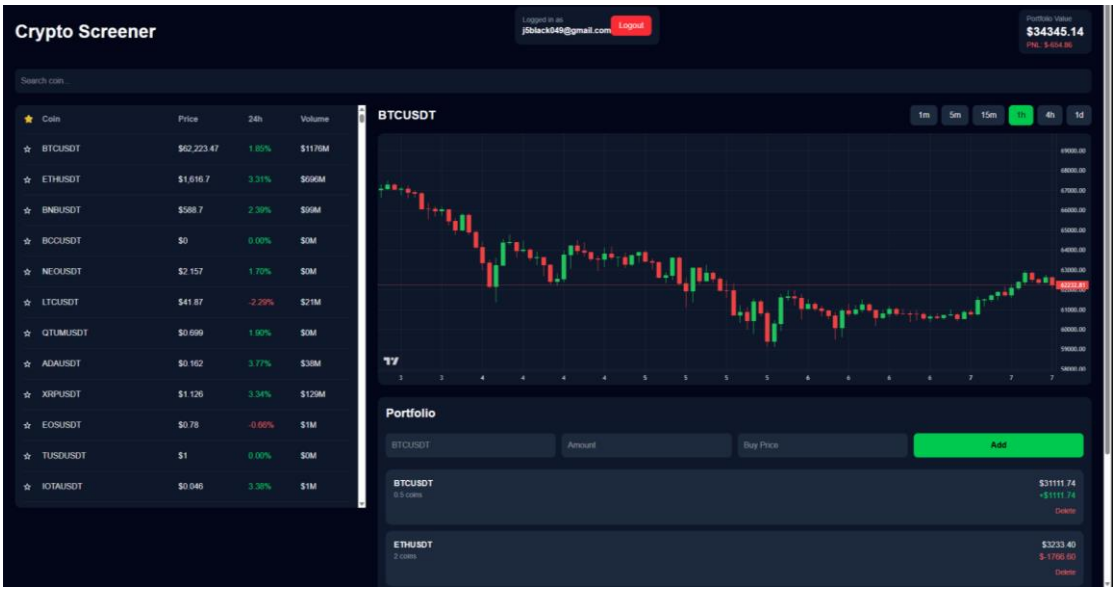
```
          <PriceChart data={chartData} />
          <Portfolio
  portfolio={portfolio}
  setPortfolio={setPortfolio}
  coins={coins}
/>
<AIAssistant />
        </div>
      </div>
    </div>
  );
}
```

ДОДАТОК Б

Інтерфейс користувача вебплатформи



Б.1.Сторінка входу та реєстрації користувача



Б.2.Інтерфейс користувача платформи

ДОДАТОК В

Конфігурація Firebase та OpenAI API

```
import { initializeApp } from "firebase/app";
import { getAuth } from "firebase/auth";
import { getFirestore } from "firebase/firestore";
const firebaseConfig = {
  apiKey: "AIzaSyAvun5Gl52J8OZfmaI0FJr_UhtkLvY6OeQ",
  authDomain:
    "crypto-screener-2ea62.firebaseapp.com",
  projectId: "crypto-screener-2ea62",
  storageBucket:
    "crypto-screener-2ea62.firebasestorage.app",
  messagingSenderId: "964288604136",
  appId:
    "1:964288604136:web:578f6419503d0071f2a65d",
  measurementId: "G-B5CMD6LB5R",
};
const app = initializeApp(firebaseConfig);
export const auth = getAuth(app)
export const db = getFirestore(app);

import { useState } from "react";

import OpenAI from "openai";

const openai = new OpenAI({
  apiKey:
    import.meta.env
      .VITE_OPENAI_API_KEY,

  dangerouslyAllowBrowser: true,
```

```
});

export default function AIAssistant() {
  const [question, setQuestion] =
    useState("");

  const [answer, setAnswer] =
    useState("");

  const [loading, setLoading] =
    useState(false);

  const askAI = async () => {
    if (!question) return;

    setLoading(true);

    try {
      const completion =
        await openai.chat.completions.create(
          {
            model: "gpt-4o-mini",

            messages: [
              {
                role: "system",

                content:
                "You are an AI crypto trading assistant. Analyze crypto markets
professionally.",
```

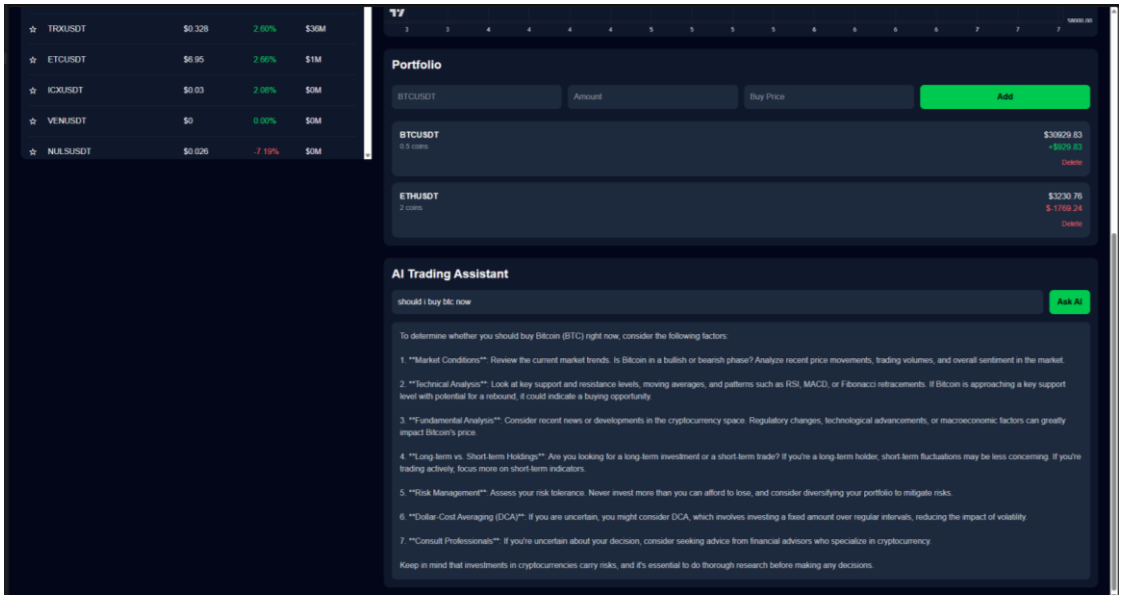
```
            },

            {
              role: "user",
              content: question,
            },
          ],
        }
      );
    setAnswer(
      completion.choices[0]
        .message.content
    );
  } catch (err) {
    console.log(err);
    setAnswer(
      "Error connecting to AI"
    );
  }
  setLoading(false);
};
return (
  <div className="mt-6 bg-slate-900 rounded-xl p-4">
    <h2 className="text-2xl font-bold mb-4">
      AI Trading Assistant
    </h2>
    <div className="flex gap-3">
      <input
        type="text"
        placeholder="Should I buy BTC now?"
```

```
          className="flex-1 bg-slate-800 p-3 rounded-lg outline-none"
          value={question}
          onChange={(e) =>
            setQuestion(
              e.target.value
            )
          }
        />
        <button
          onClick={askAI}
          className="bg-green-500 text-black px-4 rounded-lg font-bold"
        >
          Ask AI
        </button>
      </div>
      <div className="mt-4 bg-slate-800 p-4 rounded-lg min-h-[120px]">
        {loading ? (
          <p>Analyzing market...</p>
        ) : (
          <p className="text-gray-300 whitespace-pre-wrap">
            {answer}
          </p>
        )}
      </div>
    </div>
  );
}
```

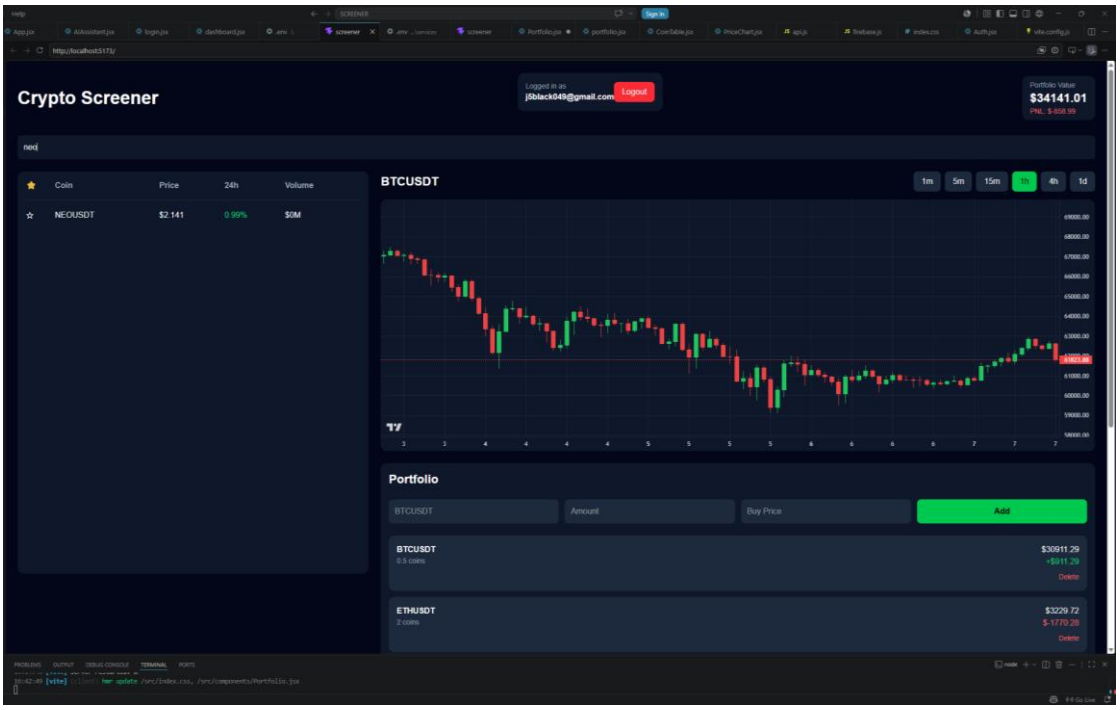
ДОДАТОК Г

Скріншоти роботи системи

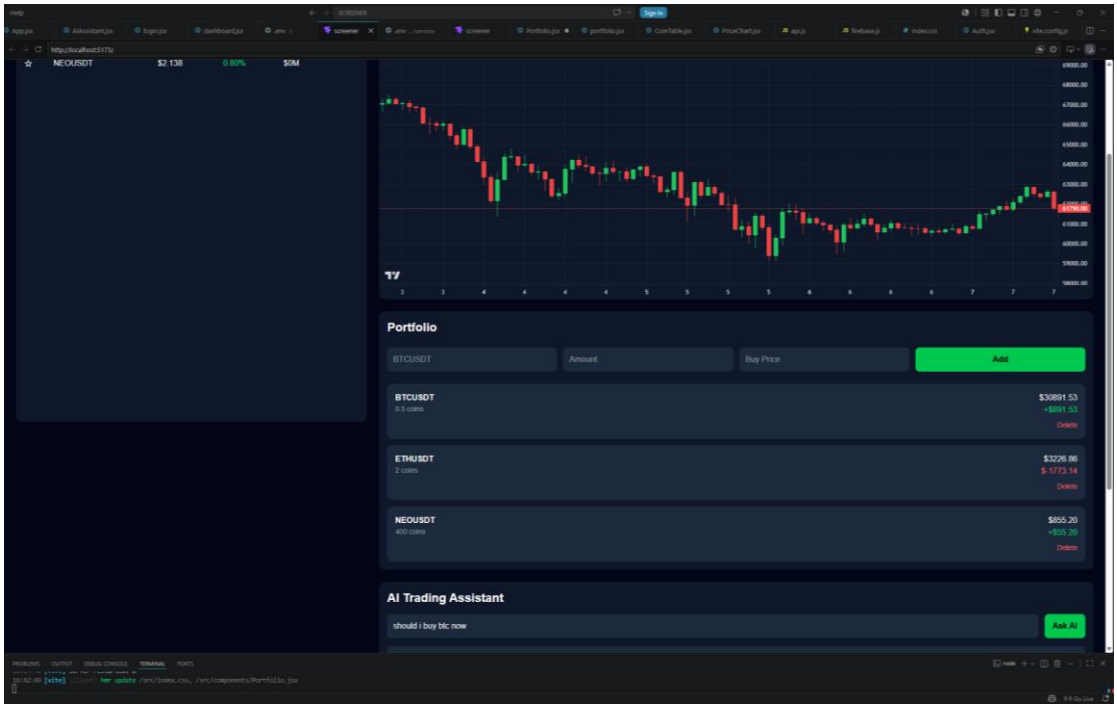
Г.1.Взаємодія з ШІ



Г.2.Реалізація пошуку активу



Г.3.Реалізація додавання активу до портфеля


NEOUSDT
$2.138
0.80%
$0M
Portfolio
BTCUSDT
Amount
Buy Price
Add
BTCUSDT
0.5 coins
$30891.53
+$891.53
Delete
ETHUSDT
2 coins
$3226.86
$-1773.14
Delete
NEOUSDT
400 coins
$855.20
+$55.20
Delete
AI Trading Assistant
should i buy btc now
Ask AI

ДОДАТОК Д

Результати тестування вебплатформи

Показники часу відгуку системи (Response Time) за умов різної кількості одночасних запитів

Тип операції / Запиту до системи	Середній час відгуку (10 користувачів), мс	Середній час відгуку (50 користувачів), мс	Середній час відгуку (100 користувачів), мс	Статус успішності операцій (Success Rate), %
Автентифікація користувача (Firebase Auth)	115	148	182	**100.0 %**
Завантаження списку монет з Binance REST API (/api/v3/ticker/24hr)	210	265	340	**100.0 %**
Встановлення WebSocket-з'єднання ({symbol}@kline_{interval})	90	112	145	**100.0 %**
Завантаження OHLCV-даних свічок (/api/v3/klines, limit=100)	180	230	310	**100.0 %**
Читання портфеля з Firebase Firestore (getDoc)	55	78	105	**100.0 %**
Запис портфеля до Firebase Firestore (setDoc)	80	115	160	**100.0 %**
Перемикання таймфрейму графіка (REST + WebSocket reconnect)	290	370	480	**100.0 %**
Отримання відповіді AI Trading Assistant (OpenAI GPT-4o-mini)*	2 800	3 400	4 200	**98.1 %**

*Примітка: Тривалий час відгуку при зверненні до AI Trading Assistant зумовлений архітектурними особливостями обробки запитів на серверах OpenAI та часом, необхідним для генерації токенів моделлю GPT-4o-mini. Зазначені значення є характеристикою зовнішнього сервісу і не є дефектом розробленого застосунку Crypto Screener.