ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ І ТЕХНОЛОГІЙ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

бакалавра
(освітньо-кваліфікаційний рівень)

на тему Розробка інтелектуального Telegram-бота 'CryptoRadar' для моніторингу та аналізу криптовалютного ринку в реальному часі

Виконав: здобувач *4* року навчання
групи *4ПР1*
напряму підготовки (спеціальності)
121-Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)

Осьмухін А.Г.
(підпис, прізвище та ініціали)

Керівник *Огнєва О.Є.*
(підпис, прізвище та ініціали)

Рецензент *к.т.н., доцент Корніловська Н.В*
(прізвище та ініціали)

Нормоконтролер *Комісаров О. С.*
(підпис, прізвище та ініціали)

Хмельницький – 2026

ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Інститут, факультет, відділення *Інформаційних технологій та дизайну*
Кафедра, циклова комісія *Програмних засобів і технологій*
Освітньо-кваліфікаційний рівень *бакалавр*
Освітньо-професійна підготовка *Програмна інженерія*
(шифр і назва)
Спеціальність *121-Інженерія програмного забезпечення*
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри *програмних засобів і технологій*
_____ О.Є. Огнєва
«___» _____ 2026 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Осьмухіну Антону Геннадійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) *Розробка інтелектуального Telegram-бота 'CryptoRadar' для моніторингу та аналізу криптовалютного ринку в реальному часі*

керівник проєкту (роботи) к.т.н. доцент Огнєва О.Є.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «16» січня 2026 року №82-с

2. Строк подання здобувачем проєкту(роботи) 10.06.2026

3. Вихідні дані до проєкту (роботи) *джерела літератури та періодики, а також матеріали переддипломної практики.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження та аналіз інтелектуальних систем моніторингу ринку криптовалют
Аналіз вимог та розробка проєктних специфікацій
Проєктування інтелектуального програмного комплексу
Розробка, тестування та робота з інтелектуальним програмним комплексом

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
2 таблиці, 18 малюнків

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *16.01.2026*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів проєкту (роботи)	Примітка
1	Отримання завдання	16.01.2026	Виконано
2	Підбір літератури	16.01.2026-10.02.2026	Виконано
3	Аналіз предметної області	10.02.2026-27.03.2026	Виконано
4	Формулювання вимог та техніко-економічне обґрунтування розробки	03.02.2026-29.03.2026	Виконано
5	Побудова концептуальної моделі та логічної схеми системи	20.03.2026-10.04.2026	Виконано
6	Архітектурне моделювання	21.03.2026-03.04.2026	Виконано
7	Програмна реалізація Telegram-бота та інтерактивного веб-інтерфейсу	05.03.2026-10.04.2026	Виконано
8	Верифікація, функціональне тестування та налагодження компонентів	07.04.2026-20.04.2026	Виконано
9	Оформлення пояснювальної записки	28.04.2026-01.06.2026	Виконано
10	Захист кваліфікаційної роботи	24.06.2026	Виконано

Здобувач _____ *Осьмухін А.Г.*
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ *О.Є.Огнєва*
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна робота бакалавра: 73 сторінки, містить 18 рисунків, 2 таблиц , 15 джерел та один додаток.

Мета роботи – розробка інтелектуального програмного комплексу «CryptoRadar» на базі Telegram Mini App та технологій штучного інтелекту, основним завданням якого є забезпечення оперативного моніторингу котирувань криптовалют, динамічна візуалізація інтерактивних фінансових графіків та генерація лаконічних текстових звітів ШІ українською мовою в реальному часі.

Об’єкт дослідження – методи проєктування, програмні архітектури та технології створення інтелектуальних мобільних клієнт-серверних веб-додатків.

Предмет дослідження – розробка інтелектуального Telegram-бота та інтегрованого веб-додатка (Mini App) для моніторингу криптовалютного ринку з інтелектуальним ШІ-аналізом.

Методи дослідження – аналіз та узагальнення теоретичних відомостей про сучасні децентралізовані фінансові ринки, технології інтеграції сторонніх прикладних інтерфейсів (API) та обробки великих обсягів даних; об'єктно-орієнтоване моделювання архітектури програмного забезпечення з використанням UML-нотацій; проєктування інтерактивних користувацьких SPA-інтерфейсів та розробка системних інструкцій (Prompt Engineering) для інтеграції великих мовних моделей.

Результат роботи:

Запропоновано та обґрунтовано клієнт-серверу архітектуру інтелектуального програмного комплексу на основі асинхронного бекенду та легкого кросплатформного клієнтського Mini App;

Реалізовано асинхронний інтеграційний контур з джерелом ринкових даних CoinGecko API та хмарним сервісом штучного інтелекту Groq AI (модель LLaMA 3.1 8B) для генерації аналітичних інсайтів;

Створено інтерактивний користувацький інтерфейс з відображенням свічкових графіків цін TradingView Lightweight Charts, вбудований безпосередньо в месенджер Telegram за допомогою Telegram Web App SDK.

Новизна роботи:

розроблено інтелектуальний програмний комплекс, який об'єднує оперативне отримання котирувань, динамічну візуалізацію фінансових графіків та генерацію автоматичних текстових інсайтів штучного інтелекту українською мовою в єдиному нативному вікні чату месенджера Telegram;

доведено практичну ефективність застосування технології Telegram Mini Apps спільно з хмарним викликом LLM для суттєвого зниження апаратних вимог на клієнтські пристрої користувачів та зменшення їхнього когнітивного навантаження під час аналізу високодеталізованих фінансових інструментів.

Ключові слова: ВЕБ-ДОДАТОК, TELEGRAM MINI APP, ШТУЧНИЙ ІНТЕЛЕКТ, LLAMA 3.1, GROQ API, COINGECKO, TRADINGVIEW LIGHTWEIGHT CHARTS, ПРОЄКТУВАННЯ, РОЗРОБКА, ТЕСТУВАННЯ.

АНОТАЦІЯ

Кваліфікаційна робота бакалавра складається з таких структурних елементів: вступ, чотири розділи, загальні висновки, список використаних джерел та додатки.

Перший розділ «Дослідження та аналіз інтелектуальних систем моніторингу ринку криптовалют» присвячений огляду сучасних джерел інформації та вивченню специфіки динамічних ринкових даних криптовалютного сегменту. У розділі детально проаналізовано принципи побудови мобільних аналітичних рішень, розглянуто архітектурні особливості систем збору та візуалізації котирувань в реальному часі. Також здійснено порівняльний аналіз існуючих комерційних та відкритих програмних продуктів для криптовалютного моніторингу і обґрунтовано необхідність створення швидкого інтелектуального помічника у вигляді мікрододатка.

Другий розділ «Аналіз вимог та розробка проєктних специфікацій інформаційної системи» містить підрозділи, в яких сформульовано ключові вимоги до інтелектуального Telegram-бота та вбудованого веб-додатка, проведено формалізацію функціональних та нефункціональних вимог, а також розроблено проєктні специфікації. У даному розділі здійснено детальне проєктування користувацьких сценаріїв та побудовано діаграми прецедентів використання. Змодельовано поведінку системи та процеси передачі інформації між клієнтською частиною Mini App, проксі-сервером та зовнішніми інтеграційними сервісами за допомогою UML-діаграм послідовності. Описано загальну логіку взаємодії користувача з інтерфейсом та розроблено специфікації інтерфейсу.

Третій розділ «проєктування інтелектуального програмного комплексу» містить такі підрозділи: «Постановка задачі проєктування», «Обґрунтування актуальності та вибір методів розв'язання», «проєктування клієнт-серверної архітектури системи». У цьому розділі детально формалізовано інженерні завдання розробки та обґрунтовано вибір асинхронних архітектурних рішень.

Проведений порівняльний аналіз алгоритмів обробки даних та інтеграції великих мовних моделей доводить доцільність обраного стеку технологій, який орієнтований на високу швидкість роботи та мобільність інтерфейсу: асинхронне серверне ядро на базі фреймворку FastAPI (мова Python), модуль асинхронних запитів aiohttp, хмарний сервіс інференції великих мовних моделей Groq API (модель LLaMA 3.1 8B Instant), а також легкий кросплатформний веб-клієнт на базі технологій HTML5, CSS3, Vanilla JavaScript та Telegram Web App SDK. Засобом інтерактивної візуалізації графіків обрано високопродуктивну відкриту бібліотеку TradingView Lightweight Charts.

Четвертий розділ «Програмна реалізація, тестування та розгортання інформаційного комплексу» складається з підрозділів, присвячених обґрунтуванню інструментальних засобів розробки, опису процесів написання та конфігурації коду, тестуванню системи, інсталяції та розгортанню на сервері, а також оцінці перспектив модернізації. Розроблений інтелектуальний комплекс пройшов серію тестувань на виявлення помилок маршрутизації, коректність відображення графіків та точність відповідей моделі ШІ. Результати верифікації підтвердили повну відповідність системи критеріям функціональності, надійності, швидкодії та мобільності використання. Окреслено ключові напрямки майбутнього розширення функціоналу системи.

ABSTRACT

The bachelor's qualification thesis consists of the following structural parts: an introduction, four chapters, general conclusions, a list of references, and appendices.

The first chapter "Research and Analysis of Intelligent Cryptocurrency Market Monitoring Systems" is devoted to reviewing modern information sources and studying the specifics of dynamic market data in the cryptocurrency segment. The chapter analyzes in detail the principles of building mobile analytical solutions and considers the architectural features of real-time quote collection and visualization systems. A comparative analysis of existing commercial and open-source software products for cryptocurrency monitoring is also conducted, justifying the need to create a fast intelligent assistant in the form of a micro-application.

The second chapter "Requirements Analysis and Development of Information System Design Specifications" contains subsections that formulate the key requirements for the intelligent Telegram bot and the embedded web application, formalize functional and non-functional requirements, and develop design specifications. This chapter provides a detailed design of user scenarios and constructs use case diagrams. The system behavior and data transmission processes between the Mini App client, the proxy server, and external integration services are modeled using UML sequence diagrams. The general logic of user interaction with the interface is described, and interface specifications are developed.

The third chapter "Design of the Intelligent Software Suite" contains the following subsections: "Formulation of the Design Task", "Justification of Urgency and Choice of Solving Methods", "Design of the System's Client-Server Architecture". In this chapter, the engineering tasks of development are detailed, and the choice of asynchronous architectural solutions is justified. A comparative analysis of data processing algorithms and integration of large language models proves the feasibility of the chosen technology stack, which is focused on high performance and interface mobility: an asynchronous server core based on the

FastAPI framework (Python language), an aiohttp asynchronous request module, a cloud inference service for large language models Groq API (LLaMA 3.1 8B Instant model), as well as a lightweight cross-platform web client based on HTML5, CSS3, Vanilla JavaScript, and Telegram Web App SDK. The high-performance open-source TradingView Lightweight Charts library was chosen as the tool for interactive visualization of charts.

The fourth chapter "Software Implementation, Testing, and Deployment of the Information Suite" consists of subsections dedicated to the justification of software development tools, the description of code writing and configuration processes, system testing, installation, and deployment on the server, as well as the assessment of modernization prospects. The developed intelligent suite underwent a series of tests to identify routing errors, verify correct chart rendering, and ensure the accuracy of AI model responses. The verification results confirmed full compliance with the criteria of functionality, reliability, performance, and mobility of use. The key directions for future expansion of the system's functionality are outlined.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЗ – програмне забезпечення.

ШІ – штучний інтелект.

API (Application Programming Interface) – прикладний програмний інтерфейс, який забезпечує взаємодію та обмін даними між різними програмними продуктами.

ASGI (Asynchronous Server Gateway Interface) – стандарт асинхронного інтерфейсу серверного шлюзу для мови програмування Python, призначений для побудови високопродуктивних мережевих додатків.

CORS (Cross-Origin Resource Sharing) – технологія спільного використання ресурсів між різними джерелами, яка регулює безпеку запитів веб-браузера до сторонніх доменів.

CSS (Cascading Style Sheets) – каскадні таблиці стилів, мова розмітки стилів для опису візуального оформлення веб-документів.

DOM (Document Object Model) – об'єктна модель документа, програмний інтерфейс для представлення та взаємодії з елементами HTML-сторінок.

ENV (Environment) – конфігураційні змінні оточення або файли налаштувань програмного забезпечення.

FNG (Fear and Greed) – індекс настроїв ринку («страху та жадібності»), який відображає психологічний стан інвесторів.

HTML (HyperText arkup Language) – стандартизована мова розмітки гіпертексту, що використовується для структурування веб-сторінок.

HTTP / HTTPS (HyMperText Transfer Protocol / Secure) – протокол передачі гіпертексту (та його захищена версія), що є основним стандартом обміну даними в мережі Інтернет.

JS (JavaScript) – мультипарадигмальна мова програмування, яка застосовується для створення інтерактивних клієнтських веб-інтерфейсів.

JSON (JavaScript Object Notation) – текстовий формат представлення структурованих даних, призначений для легкого обміну інформацією між клієнтом та сервером.

LLM (Large Language Model) – велика мовна модель, глибока нейромережа для розпізнавання, обробки та генерації природного тексту.

MVP (Minimum Viable Product) – мінімально життєздатний продукт, версія програмного забезпечення з базовим набором функцій для первинного тестування користувачами.

OHLC (Open, High, Low, Close) – формат відображення фінансових даних на графіках (ціна відкриття, максимум, мінімум та ціна закриття за певний таймфрейм).

REST (Representational State Transfer) – архітектурний стиль побудови розподілених систем, заснований на передачі представлень стану за допомогою стандартних HTTP-методів.

SDK (Software Development Kit) – спеціалізований набір засобів, інструментів та бібліотек для розробки програмного забезпечення під конкретну платформу.

SPA (Single Page Application) – односторінковий веб-додаток, технологія побудови інтерфейсів, яка завантажує єдиний HTML-документ і динамічно оновлює контент без перезавантаження сторінки.

TMA (Telegram Mini App) – вбудований кросплатформний веб-додаток, який запускається безпосередньо всередині екосистеми месенджера Telegram.

UI (User Interface) – користувацький інтерфейс, сукупність графічних елементів для безпосередньої взаємодії людини з програмою.

UX (User Experience) – досвід користувача, сукупність суб'єктивних вражень та зручності користування програмним інтерфейсом.

UML (Unified Modeling Language) – уніфікована мова графічного моделювання для проєктування та документування архітектури програмних систем.

URL (Uniform Resource Locator) – уніфікований покажчик ресурсу, адреса певного веб-документа чи файлу в мережі Інтернет.

ВСТУП

Актуальність теми. Сучасний розвиток глобального фінансового простору характеризується активною інтеграцією цифрових активів та технологій децентралізованих фінансів. Ринок криптовалют є одним із найбільш динамічних секторів економіки, що відрізняється цілодобовим циклом торгівлі та надвисоким рівнем коливання цін. У таких умовах успішна діяльність інвесторів та трейдерів напряму залежить від оперативності отримання точних котирувань та вміння швидко аналізувати поточні тренди.

Проте використання класичних торгових терміналів на мобільних пристроях часто є незручним через велике навантаження на систему та складність інтерфейсів. Це створює потребу в розробці легких, швидких мобільних помічників. Технологія Telegram Mini Apps дозволяє розгортати повноцінні інтерактивні веб-додатки безпосередньо в месенджері, що забезпечує користувачам миттєвий доступ до інформації без встановлення додаткового програмного забезпечення.

Окрім цього, інтеграція хмарних мовних моделей штучного інтелекту відкриває нові можливості для автоматичної обробки сирих фінансових даних. ШІ здатний миттєво інтерпретувати складні числові показники та надавати користувачеві лаконічні висновки природною мовою. Таким чином, створення інтелектуального програмного комплексу для моніторингу криптовалютного ринку з інтегрованими графіками та автоматичним ШІ-аналізом є актуальним інженерним завданням.

Об’єкт дослідження – процеси збору, обробки, візуалізації та інтелектуального аналізу потокових ринкових даних у реальному часі.

Предмет дослідження – асинхронні програмні архітектури, кросплатформні мобільні інтерфейси типу Telegram Mini App та методи Prompt Engineering для інтеграції великих мовних моделей штучного інтелекту.

Методи дослідження. Для розв’язання поставлених завдань використано методи системного аналізу предметної області, об’єктно-орієнтоване моделювання (UML) для проєктування архітектури програмного забезпечення, методи асинхронного мережевого програмування, технології створення Single Page Applications (SPA), а також методи інженерії підказок (Prompt Engineering) для взаємодії з великими мовними моделями (LLM).

Мета і завдання дослідження. Метою кваліфікаційної роботи є проєктування та програмна розробка інтелектуального комплексу «CryptoRadar» на базі Telegram Mini App та технологій штучного інтелекту для надання користувачам оперативних котирувань, динамічних фінансових графіків та автоматичних звітів ШІ українською мовою.

Для досягнення цієї мети поставлено такі завдання:

проаналізувати ринок криптовалютних аналітичних інструментів та обґрунтувати вибір стеків програмування;

спроєктувати загальну клієнт-серверну архітектуру комплексу та створити UML-моделі взаємодії компонентів;

реалізувати асинхронне збирання даних з CoinGecko API з використанням алгоритмів кешування для оптимізації запитів;

розробити модуль інтеграції з Groq API (модель LLaMA 3.1 8B) для генерації торгових висновків та логіку їхнього автоматичного перекладу на українську мову;

реалізувати SPA-інтерфейс Mini App з інтегрованими свічковими графіками TradingView Lightweight Charts та Telegram Web App SDK;

виконати комплексне тестування розробленого програмного забезпечення на предмет швидкодії та стабільності роботи.

Наукова новизна одержаних результатів полягає в оптимізації архітектурної взаємодії між легкими інтерфейсами мобільних месенджерів та хмарними сервісами штучного інтелекту. Це дозволяє генерувати точні текстові фінансові висновки на основі динамічного ринкового контексту без перевантаження апаратних ресурсів клієнтського пристрою.

Практичне значення одержаних результатів полягає у розробці готового до використання програмного комплексу «CryptoRadar». Створений інструмент дозволяє інвесторам та трейдерам суттєво оптимізувати час на аналіз ринку завдяки поєднанню точних котирувань, інтерактивних графіків та автоматичних ШІ-інсайтів українською мовою в одному зручному інтерфейсі.

Апробація та публікації. Результати дослідження та основні архітектурні рішення були представлені у формі доповіді на науково-практичній конференції молодих вчених та опубліковані у збірнику наукових праць.

Структура роботи. Пояснювальна записка складається зі вступу, чотирьох розділів, загальних висновків, списку використаних джерел та додатків. Робота викладена на 73 сторінках, містить 18 рисунків, 2 таблиці та 15 використаних джерел.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ТА АНАЛІЗ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ МОНІТОРИНГУ РИНКУ КРИПТОВАЛЮТ

1.1. Особливості криптовалютних ринкових даних та їх агрегація

Сучасна екосистема децентралізованих фінансових активів функціонує в унікальних інфраструктурних умовах, які кардинально відрізняють її від традиційних фондових чи валютних ринків. Головними характеристиками криптовалютного ринку є його транскордонність, повна децентралізація торгових майданчиків, функціонування в режимі 24/7/365 та надзвичайно висока волатильність цінових показників. Котирування цифрових активів формуються під впливом колосальної кількості факторів: від фундаментальних макроекономічних показників до настроїв у соціальних мережах та новинного фону, що викликає різкі та непередбачувані коливання цін протягом лічених хвилин.

Ринкові дані криптовалют (Cryptocurrency Market Data) є сукупністю числових та текстових показників, що відображають поточний стан торгівлі та капіталізації активів. До базових структурованих параметрів, які підлягають постійному моніторингу, належать:

Поточна ринкова ціна (Current Price): середньозважена вартість активу, виражена у фіатній валюті (найчастіше в доларах США - USD) або базовому активі (наприклад, Bitcoin - BTC).

Зміна ціни за 24 години (24h Price Change): відсоткове або абсолютне відхилення вартості активу порівняно з аналогічним часом попередньої доби.

Ринкова капіталізація (Market Capitalization): сумарна вартість усіх випущених в обіг монет активу, яка розраховується як добуток поточної ціни на обсяг циркулюючої пропозиції (Circulating Supply).

Обсяг торгів за 24 години (24h Trading Volume): загальна сума угод з купівлі-продажу активу на всіх інтегрованих біржах протягом останніх 24 годин, яка відображає рівень ліквідності та активності ринку.

Глобальні індикатори ринку: відсоток домінування біткоїна (BTC Dominance), сумарна ринкова капіталізація та індекси настроїв інвесторів, зокрема індекс страху та жадібності (Fear and Greed Index).

Оскільки криптовалютний ринок не має єдиного регульованого центру торгівлі, аналогічного класичним біржам NYSE чи NASDAQ, операції відбуваються на сотнях централізованих (CEX) та децентралізованих (DEX) майданчиків одночасно. Кожна біржа має власну склянку ордерів (Order Book) та індивідуальні показники попиту і пропозиції, через що ціна на один і той самий актив на різних платформах може відрізнятися.

Це створює гостру потребу в агрегації даних (Data Aggregation). Агрегація у фінансовому моніторингу - це процес автоматичного збору розрізненої інформації з первинних джерел (бірж), її фільтрації, математичної обробки (найчастіше розрахунку об'ємно-зваженої середньої ціни - VWAP) та консолідації в єдиний інформаційний потік.

Процес агрегації криптовалютних даних на концептуальному рівні складається з кількох послідовних технологічних кроків, які схематично зображено на рисунку 1.1.

Рисунок 1.1. Концептуальна схема процесу агрегації та передачі криптовалютних даних

Основним джерелом агрегованих ринкових даних для сучасних розробників виступають спеціалізовані аналітичні платформи (API-провайдери), серед яких одним із лідерів є незалежний сервіс CoinGecko. Використання прикладного програмного інтерфейсу CoinGecko API дозволяє отримувати попередньо оброблені та зважені дані про тисячі цифрових активів.

Однак інтеграція сторонніх API-сервісів у реальних програмних комплексах супроводжується низкою технічних викликів. По-перше, публічні безкоштовні версії API мають жорсткі ліміти на кількість запитів (Rate Limits). Наприклад, безкоштовний тарифний план CoinGecko обмежує частоту звернень до 30 запитів на хвилину. У випадку перевищення цього ліміту сервер повертає помилку HTTP 429 (Too Many Requests), що може повністю заблокувати роботу додатка для користувачів.

По-друге, постійні мережеві запити до зовнішніх серверів створюють значні затримки (Latency) у відгуку інтерфейсу, що негативно впливає на UX користувача.

Для подолання цих обмежень у розробці інтелектуальних систем моніторингу застосовують методи асинхронного програмування та кешування даних на проміжному проксі-сервері (FastAPI backend). Замість того, щоб під час кожного кліку користувача звертатися безпосередньо до CoinGecko, серверна частина системи періодично опитує API в асинхронному режимі, зберігає актуальні структури даних у локальній оперативній пам'яті (клієнтському кеші) та миттєво віддає їх за запитом користувача. Це дозволяє забезпечити стабільну роботу системи за будь-якої інтенсивності запитів та гарантує швидкий відгук користувацького Mini App.

1.2. Принципи побудови інтелектуальних систем моніторингу фінансових активів

Традиційні інформаційні системи фінансового моніторингу орієнтовані на збір, первинну обробку та пасивне відображення кількісних ринкових показників. Проте у сучасних реаліях, коли швидкість зміни інформації вимірюється секундами, а обсяги вхідних даних стрімко зростають, перед інвесторами постає проблема «інформаційного перевантаження». Користувач змушений самостійно відслідковувати десятки графіків, порівнювати ціни на різних таймфреймах та інтерпретувати технічні індикатори. Для вирішення цієї проблеми створюються інтелектуальні системи моніторингу, які здатні не лише пасивно транслювати котирування, а й здійснювати їх інтелектуальний аналіз та надавати готові висновки.

Інтелектуалізація систем моніторингу фінансових активів базується на переході від «даних» (data-centric) до «знань» (knowledge-centric). Головною метою таких систем є мінімізація когнітивного навантаження на людину шляхом автоматизації аналітичних процесів.

Побудова сучасних інтелектуальних систем фінансового моніторингу спирається на такі фундаментальні принципи:

1. **Принцип безперервності та асинхронності**: забезпечення цілодобового та безперебійного аналізу вхідного потоку котирувань в асинхронному режимі без блокування користувацького інтерфейсу.
2. **Принцип контекстуалізації даних:** об'єднання розрізнених показників (ціни окремого активу, глобальних індексів, домінування лідерів ринку) в єдиний інформаційний контекст для комплексного аналізу.
3. **Принцип автоматизованої семантизації:** перетворення сирих числових JSON-структур ринкових котирувань на якісні, легкі для розуміння людиною висновки природною мовою за допомогою генеративного штучного інтелекту.
4. **Принцип мультимодальності інтерфейсу**: комбінування візуального аналізу (свічкових графіків) та інтелектуальної текстової аналітики в межах єдиного візуального простору (Single Page Application).

Протягом тривалого часу інтелектуальний аналіз ринків обмежувався спробами математичного прогнозування котирувань за допомогою нейромереж типу LSTM або класичних статистичних моделей (ARIMA). Проте хаотична природа та висока волатильність криптовалютного ринку роблять точне чисельне прогнозування на великих проміжках часу практично неможливим.

Сучасний підхід до інтелектуалізації фінансових систем змістився в бік застосування великих мовних моделей (LLM). Замість ризикованих чисельних прогнозів, генеративний штучний інтелектуальний агент виконує роль професійного експерта-консультанта. Модель обробляє комплексний ринковий контекст та генерує текстовий інсайт: пояснює причини поточної динаміки цін, виявляє приховані технічні сигнали та надає короткий стратегічний опис можливих ринкових сценаріїв.

Для реалізації цього підходу на практиці розробляється трирівнева архітектура обробки ринкових знань, яку представлено на рисунку 1.2.

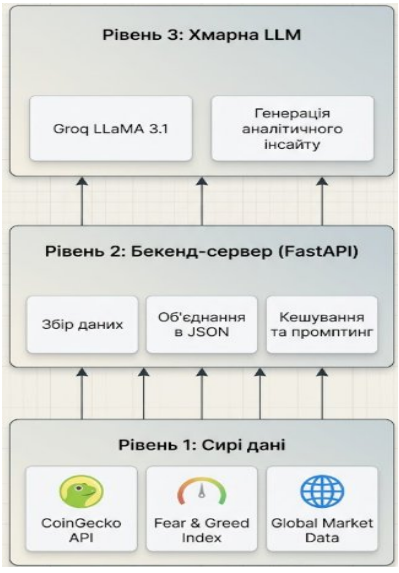

Рисунок 1.2 - Рівні обробки інформації в інтелектуальній системі фінансового моніторингу

Основним інструментом управління поведінкою штучного інтелекту в таких системах є методологія Prompt Engineering (інженерія підказок). Вона

передбачає створення системного промпту - детальних інструкцій, які налаштовують LLM на роль лаконічного та досвідченого аналітика, виключають довгі абстрактні вступи («вода»), забороняють стандартні юридичні дисклеймери («це не є фінансовою порадою») та обмежують формат відповіді (наприклад, до 40-50 слів у стилі професійного трейдерського звіту).

Завдяки цьому користувач системи моніторингу отримує унікальну перевагу: замість виснажливого порівняння таблиць та побудови складних ліній на графіках, він може за один клік отримати концентровану текстову аналітику ринку, адаптовану під його локальну мову.

1.3. Особливості програмних комплексів для візуалізації та ШІ-аналізу котирувань

Ефективність систем підтримки прийняття рішень на фінансових ринках безпосередньо залежить від ергономіки та способів представлення інформації. проєктування інтерфейсів для моніторингу волатильних активів базується на поєднанні двох взаємодоповнюючих каналів сприйняття інформації людським мозком: візуального та смислового (семантичного). Синергія цих каналів дозволяє значно прискорити аналіз ринкової ситуації та знижує ризик прийняття хибних рішень.

Візуальний канал сприйняття реалізується через графічне відображення котирувань. Найбільш інформативним стандартом аналізу фінансових активів у світі є японські свічки (Candlestick Chart) та формат відображення даних OHLC. Вони дозволяють трейдеру за частку секунди оцінити загальний напрямок тренду, локальні мінімуми й максимуми ціни, а також динаміку боротьби між покупцями та продавцями на певному часовому інтервалі (таймфреймі).

Проте графічний аналіз має суттєві обмеження: він вимагає від користувача спеціальної підготовки (знання графічного аналізу, вміння

визначати рівні підтримки та опору тощо) та не дає відповіді на питання «Чому саме ціна рухається у цьому напрямку?».

Для подолання цих обмежень інтегрується семантичний канал аналізу на основі генеративного штучного інтелекту. ШІ-агент обробляє текстову та числову інформацію (наприклад, глобальні ринкові індекси, Fear & Greed Index, динаміку суміжних активів) і надає стислий змістовний опис ситуації природною мовою.

Спільне функціонування цих двох каналів сприйняття в інтерфейсі програмного комплексу представлено на рисунку 1.3.

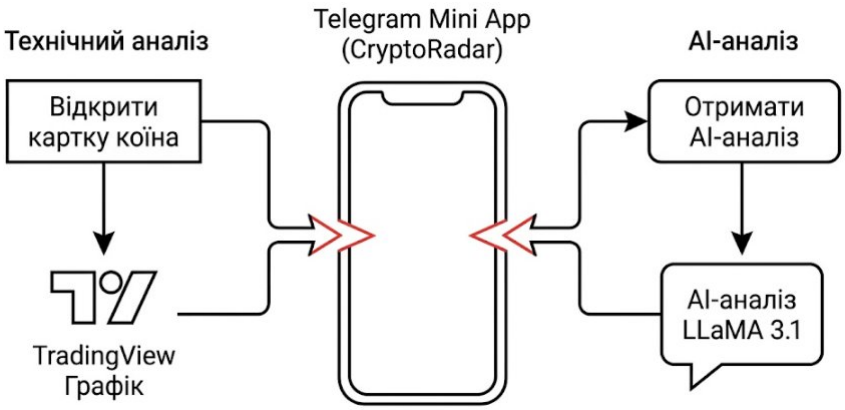

Рисунок 1.3 - Концепція синергії графічного відображення котирувань та ШІ-аналітики в єдиному інтерфейсі

Реалізація такої синергії в умовах мобільних месенджерів (Telegram Mini App) висуває надзвичайно жорсткі вимоги до використовуваних програмних засобів:

1. Мінімальний обсяг клієнтського коду (Light Footprint): мобільні пристрої користувачів можуть мати обмежену оперативну пам'ять та повільне мережеве з'єднання. Важкі бібліотеки візуалізації (наприклад, D3.js чи повні комерційні версії графіків) перевантажують браузер і викликають затримки при рендерингу.
2. Адаптивність та сенсорне керування (Touch Responsiveness): графік повинен бездоганно масштабуватися на екранах смартфонів різної

роздільної здатності, підтримувати жести зближення/розсунення пальців (pinch-to-zoom) та прокручування пальцем.

3. Асинхронне завантаження компонентів: інтерфейс веб-додатка повинен завантажуватися миттєво (SPA-архітектура), а важкі обчислення (запити до ШІ та побудова історичних графіків) мають виконуватися у фоновому режимі (on-demand) без заморожування екрана.

Ідеальним рішенням для візуалізації в таких умовах є використання відкритої бібліотеки TradingView Lightweight Charts[8]. Завдяки розміру standalone-файлу менше ніж 50 КБ, апаратному прискоренню рендерингу (через HTML5 Canvas) та вбудованій оптимізації під сенсорні дисплеї, ця бібліотека забезпечує плавну анімацію та миттєве відображення котирувань навіть на бюджетних мобільних пристроях.

У поєднанні з інтелектуальним модальним вікном ШІ-аналізу (на основі LLaMA 3.1), яке користувач може викликати в один клік безпосередньо під графіком, розроблений комплекс створює цілісну, швидку та зручну інформаційну екосистему для оперативного прийняття рішень на криптовалютному ринку.

1.4. Засоби та технології для створення асинхронних веб-додатків обробки та аналізу даних

Сучасні веб-додатки, які спеціалізуються на обробці та аналізі потокових ринкових даних у реальному часі, вимагають високої продуктивності серверної інфраструктури. Основною технічною особливістю таких систем є те, що більшість їхніх операцій належать до класу I/O-bound (операції введення-виведення, що обмежені очікуванням мережі). Сервер витрачає понад 95% свого робочого часу не на складні математичні обчислення центральним процесором (CPU-bound), а на очікування відповідей від зовнішніх мережевих ресурсів: API-провайдера CoinGecko та хмарного сервісу інференції штучного інтелекту Groq[4].

У традиційних веб-фреймворках (наприклад, Django чи Flask), які використовують синхронний стандарт WSGI (Web Server Gateway Interface), кожен новий запит від користувача обробляється в окремому потоці (Thread). Коли сервер звертається до зовнішнього API, цей потік повністю блокується (очікує мережеву відповідь). У разі одночасного звернення сотень користувачів ресурси сервера швидко вичерпуються (Thread Depletion), що призводить до критичного зростання часу очікування та падіння системи.

Для подолання цієї проблеми застосовують асинхронну архітектуру на базі стандарту ASGI (Asynchronous Server Gateway Interface) та вбудованої бібліотеки Python asyncio[9]. Асинхронний сервер працює в межах одного потоку, використовуючи механізм циклу подій (Event Loop). Коли програма ініціює запит до стороннього API, вона добровільно передає керування циклу подій, який перемикається на обробку інших завдань (наприклад, запитів від інших користувачів). Після отримання мережевої відповіді перервана операція автоматично відновлюється[1].

Порівняльний аналіз обробки запитів у синхронній та асинхронній архітектурах представлено на рисунку 1.4.

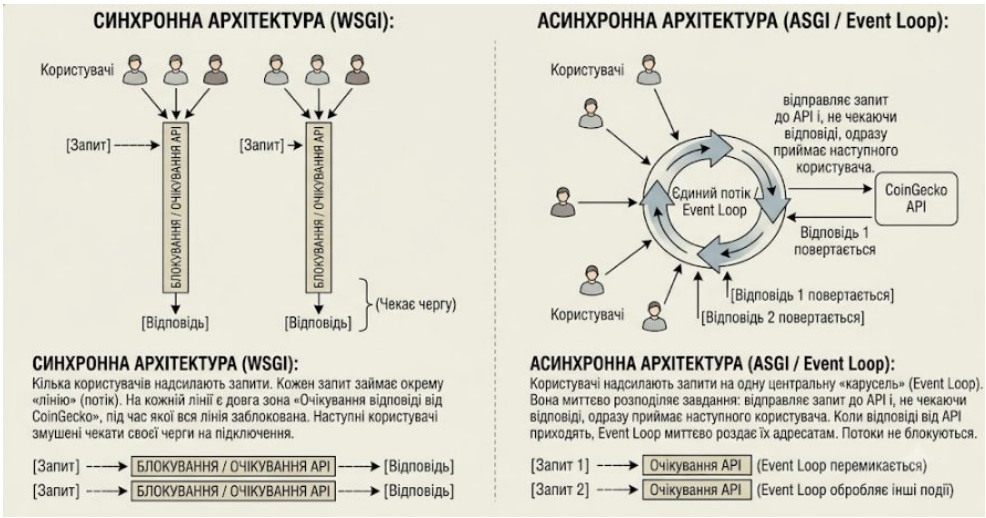

Рисунок 1.4 - Схема порівняння синхронної (WSGI) та асинхронної (ASGI) обробки запитів під навантаженням

Основним засобом побудови серверної частини нашого проєкту обрано сучасний асинхронний фреймворк FastAPI (мова Python 3.11). FastAPI побудований на базі двох високопродуктивних бібліотек:

Starlette: відповідає за швидку маршрутизацію веб-запитів та обробку асинхронних з'єднань;

Pydantic: забезпечує блискавичну валідацію, серіалізацію та структуризацію даних за допомогою строгої типізації Python (Type Hints)[11].

В якості ASGI-сервера використовується Uvicorn - надзвичайно швидка реалізація веб-сервера на базі бібліотеки uvloop, написаної на мові Cython, яка за швидкістю роботи наближається до рішень на мовах Go та Node.js[9].

Для реалізації асинхронного мережевого клієнта замість класичної синхронної бібліотеки requests використовується спеціалізована бібліотека aiohttp. Вона дозволяє створювати стійкі клієнтські сесії (aiohttp.ClientSession), які використовують пул з'єднань (Connection Pooling) для багаторазового використання одного TCP-порту. Це значно знижує окладні витрати на встановлення мережевих з'єднань.

Аналітичний блок системи інтегрований з хмарною інфраструктурою Groq Cloud API[4]. Вибір Groq зумовлений використанням революційних процесорів LPU (Language Processing Unit), які забезпечують швидкість генерації тексту моделлю LLaMA 3.1 8B на рівні понад 300 токенів за секунду. Це дозволяє отримувати готові аналітичні інсайти практично миттєво, усуваючи проблему довгих затримок ШІ-генерації, що є критично важливим для збереження плавності взаємодії користувача в межах Telegram Mini App.

1.5. Огляд існуючих програмних рішень для криптовалютного аналізу та відстеження портфеля

Швидкий розвиток індустрії цифрових активів стимулював появу численних програмних рішень для відстеження котирувань, технічного

аналізу та управління портфелем. На сьогоднішній день існуючі на ринку аналітичні інструменти можна умовно розділити на три основні категорії:

1. Глобальні інформаційно-аналітичні портали (наприклад, CoinGecko, CoinMarketCap): це потужні бази даних, які пропонують вичерпну інформацію про тисячі активів. Проте їхні мобільні додатки перевантажені рекламою, вимагають високої швидкості інтернету, споживають багато оперативної пам'яті та орієнтовані на пасивне споглядання цифр, не надаючи швидкого інтелектуального аналізу ринкових причин.
2. Професійні графічні платформи (наприклад, TradingView): світові лідери у сфері технічного аналізу. Вони пропонують бездоганний інструментарій для малювання та сотні індикаторів. Однак їхні інтерфейси є надзвичайно складними для пересічних інвесторів (високий поріг входження), платні функції коштують дорого, а на екранах смартфонів робота з дрібними елементами графіків стає вкрай незручною[8].
3. Спрощені інформаційні Telegram-боти: легкі та швидкі помічники в месенджерах. Їхнім головним недоліком є низька інформативність - вони надсилають лише статичні текстові повідомлення із поточною ціною, не дозволяючи користувачеві взаємодіяти з динамічними графіками або отримувати розгорнуту аналітику.

Для наочного й аргументованого обґрунтування технічної доцільності розробки нашої системи проведемо детальний порівняльний аналіз спроєктованого програмного комплексу «CryptoRadar» з типовими представниками кожної з вищеописаних категорій. Результати цього комплексного порівняння за ключовими технічними критеріями наочно представлено в таблиці 1.1.

Таблиця 1.1. Порівняльний аналіз розробленого комплексу «CryptoRadar» з існуючими аналогами

Критерій порівняння	Мобільний додаток CoinGecko	Мобільний додаток TradingView	Типовий Telegram Price Bot	Розроблений комплекс «CryptoRadar»
Необхідність встановлення на пристрій	Так (потребує завантаження з AppStore/GooglePlay)	Так (потребує завантаження з AppStore/GooglePlay)	Ні (запуск безпосередньо в месенджері)	**Ні (запуск в один клік у Telegram як Mini App)**
Вимоги до апаратних ресурсів смартфона	Високі (займає багато пам'яті, споживає батарею)	Дуже високі (вимагає продуктивного процесора)	Низькі (мінімальні вимоги месенджера)	**Дуже низькі (завдяки технології легкого SPA)**
Наявність інтерактивних графіків свічок (OHLC)	Обмежена (статичні лінійні графіки на мобільних)	Повна (надлишковий професійний інструментарій)	Відсутня (лише текстові цінові повідомлення)	**Оптимальна (інтерактивні графіки TradingView Lightweight Charts)**
Інтелектуальний аналіз ринку штучним інтелектом (LLM)	Відсутній (користувач аналізує дані самостійно)	Відсутній (користувач аналізує дані самостійно)	Відсутній (лише сирі цифрові показники)	**Наявний (миттєві інсайти від LLaMA 3.1 8B у реальному часі)**
Автоматична локалізація звітів (українська мова)	Часткова (лише елементи інтерфейсу)	Часткова (лише інтерфейс системи)	Відсутня або обмежена	**Повна (локалізація ринкових звітів ШІ українською мовою)**
Рівень когнітивного навантаження на користувача	Високий (велика кількість зайвих цифр та реклами)	Дуже високий (потребує фахових знань трейдингу)	Низький (але малоінформативний)	**Низький (інформація структурована та пояснена ШІ)**

Як видно з результатів порівняльного аналізу, розроблений інтелектуальний комплекс «CryptoRadar» займає унікальну нішу на ринку. Він виступає гібридним інтегрованим рішенням, яке поєднує мобільність та низькі системні вимоги Telegram-ботів із високою інформативністю професійних

графіків TradingView та аналітичними можливостями сучасного штучного інтелекту.

Завдяки безшовній інтеграції через Telegram Web App SDK, користувач отримує швидкий доступ до аналізу ринку прямо в процесі спілкування, що робить «CryptoRadar» ефективним та конкурентоспроможним інструментом підтримки прийняття рішень для сучасної аудиторії.

1.6. Висновки до розділу 1

У першому розділі було проведено комплексне дослідження предметної області, проаналізовано особливості криптовалютних ринкових даних та обґрунтовано технологічні засади проєктування інтелектуальних систем моніторингу.

За результатами аналізу зроблено такі ключові висновки:

Специфіка криптовалютного ринку (висока волатильність, робота в режимі 24/7 та фрагментація даних) вимагає автоматизованої агрегації котирувань для відображення середньозваженої ринкової ціни. Оптимальним безкоштовним джерелом таких даних визначено публічне API CoinGecko[3].

Обґрунтовано потребу у поєднанні візуального каналу сприйняття ринку (свічкові графіки) з семантичним каналом (текстовий аналіз штучного інтелекту). Такий підхід дозволяє суттєво знизити когнітивне навантаження на кінцевого користувача порівняно зі складними професійними платформами.

Показано, що для мобільних пристроїв та роботи всередині месенджерів критично важливим є використання легких, асинхронних архітектурних рішень. Обраний технологічний стек, що включає мову Python, асинхронний фреймворк FastAPI на сервері та мобільну технологію Telegram Mini App на клієнті, забезпечує високу швидкість роботи системи без блокування інтерфейсу.

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА РОЗРОБКА ПРОЄКТНИХ СПЕЦИФІКАЦІЙ

2.1. Основні вимоги до інтелектуального програмного комплексу для моніторингу та аналізу криптовалют

Проєктування будь-якої складної програмної системи починається зі збору, систематизації та детального аналізу вимог. Визначення точних вимог дозволяє окреслити межі проєкту, спланувати його архітектуру та розробити критерії успішності майбутнього програмного продукту. Для інтелектуального програмного комплексу «CryptoRadar» технічні вимоги поділяються на дві основні групи: функціональні вимоги, що описують безпосередню поведінку системи та функції, які вона виконує для користувача, а також нефункциональні вимоги, які визначають якісні характеристики системи, такі як швидкодія, надійність, мобільність та інтерфейсні обмеження.

Функціональний аспект архітектури системи першочергово передбачає створення зручної та швидкої точки входу для користувача, роль якої виконує Telegram-бот. Взаємодія з ним починається з обробки стандартної команди /start, у відповідь на яку сервіс надсилає інтерактивне повідомлення із вбудованою веб-кнопкою для миттєвого розгортання інтерфейсу додатка (Telegram Mini App).

При ініціалізації клієнтської частини програмний комплекс в автоматичному режимі звертається до серверної архітектури для агрегації актуальних ринкових даних. Результатом цього процесу є рендеринг списку десяти найпопулярніших криптовалютних активів, відсортованих за ринковою капіталізацією, із відображенням їхньої символіки, назв, поточних цін та динаміки зміни вартості за останні 24 години. Поряд із цим важливе місце посідає функція інтелектуального пошуку довільних активів у базі даних, яка

дозволяє користувачеві вводити назву або тікер монети у пошуковому рядку та миттєво отримувати релевантні результати.

Центральним елементом інтерфейсу виступає детальна картка обраного активу, яка інтегрує розширену статистику ринку (обсяги торгів, капіталізацію) та динамічний свічковий графік котирувань. Для забезпечення якісного графічного аналізу передбачено можливість гнучкого перемикання часових інтервалів (таймфреймів), зокрема відображення даних за одну добу, тиждень, місяць або поточний рік. Математичний супровід операцій забезпечується вбудованим калькулятором-конвертером валют, який реалізує механізм швидкого двостороннього перерахунку кількості цифрових активів у базові фіатні гроші (USD, EUR, UAH) чи інші криптовалюти.

Особливістю та головною інтелектуальною компонентою комплексу є генерація семантичних ШІ-інсайтів. За запитом користувача система формує стислий аналітичний звіт, що включає опис поточної технічної картини, виявлення трендів та потенційних причин коливання ціни. Цей процес завершується автоматичною локалізацією звітів штучного інтелекту українською мовою та наданням інструменту для швидкого копіювання генерованого тексту в буфер обміну.

Нефункціональні критерії якості системи висувають жорсткі вимоги до її продуктивності та стабільності під навантаженням. З точки зору швидкодії, час обробки та повернення ринкових даних проксі-сервером на базі FastAPI не повинен перевищувати ліміт у 200 мс. Водночас складніший процес, який включає звернення до великої мовної моделі ШІ та переклад фінального інсайту, має повністю завершуватися в межах 4 секунд[2].

Для забезпечення стабільної експлуатації та запобігання блокуванням з боку зовнішніх сервісів, архітектура бекенду реалізує механізми суворого контролю лімітів (Rate-Limit Handling). Це досягається завдяки інтеграції шару кешування запитів до CoinGecko API, що нівелює ризик виникнення помилок HTTP 429 при одночасному зверненні великої кількості активних клієнтів[3].

Візуальна компонента та користувацький інтерфейс (UI) проєктованого Mini App підпорядковуються вимогам адаптивності та мобільної чуйності. Програмний код фронтенду забезпечує коректний рендеринг елементів на екранах смартфонів у діапазоні ширини від 320px до 600px із обов'язковою підтримкою темної палітри оформлення для комфортного сприйняття інформації.

Критерій мобільності та нульової інсталяції (Zero-Install) гарантує, що кінцевому користувачеві не потрібно завантажувати сторонні файли чи додатки з маркетів ПЗ, оскільки запуск екосистеми відбувається всередині месенджера Telegram на будь-яких операційних системах, включаючи iOS, Android, Windows та macOS. Насамкінець, архітектура має високі показники надійності та відмовостійкості, що виражається у перехопленні та безпечній обробці мережевих помилок (відсутність зв'язку з API чи втрата інтернет-з'єднання), замість яких користувач бачить інформативні повідомлення, а додаток продовжує стабільну роботу.

2.2. Формалізація вимог додатка

Для детального моделювання бізнес-процесів та формалізації взаємодії користувача з проєктованим програмним комплексом застосовується апарат діаграм прецедентів (Use Case Diagrams) уніфікованої мови моделювання UML. Це дозволяє чітко розмежувати ролі в системі, визначити межі програмного продукту та описати повний спектр функціональних сценаріїв.
У межах розробки комплексу «CryptoRadar» виділено два ключові типи суб'єктів взаємодії (акторів):

- Головний актор (Користувач системи) - ініціює взаємодію з інтерфейсом, переглядає дані, використовує конвертер та робить запити на аналіз.

- Другорядні актори (Зовнішні системи) - забезпечують життєдіяльність комплексу. До них належать серверна частина проєкту (FastAPI Backend), інформаційне криптографічне API (CoinGecko) та хмарний сервіс великої мовної моделі штучного інтелекту (LLaMA 3.1)[2].

Візуальну структуру зв'язків між акторами та основними прецедентами використання системи відображено на рисунку 2.1.

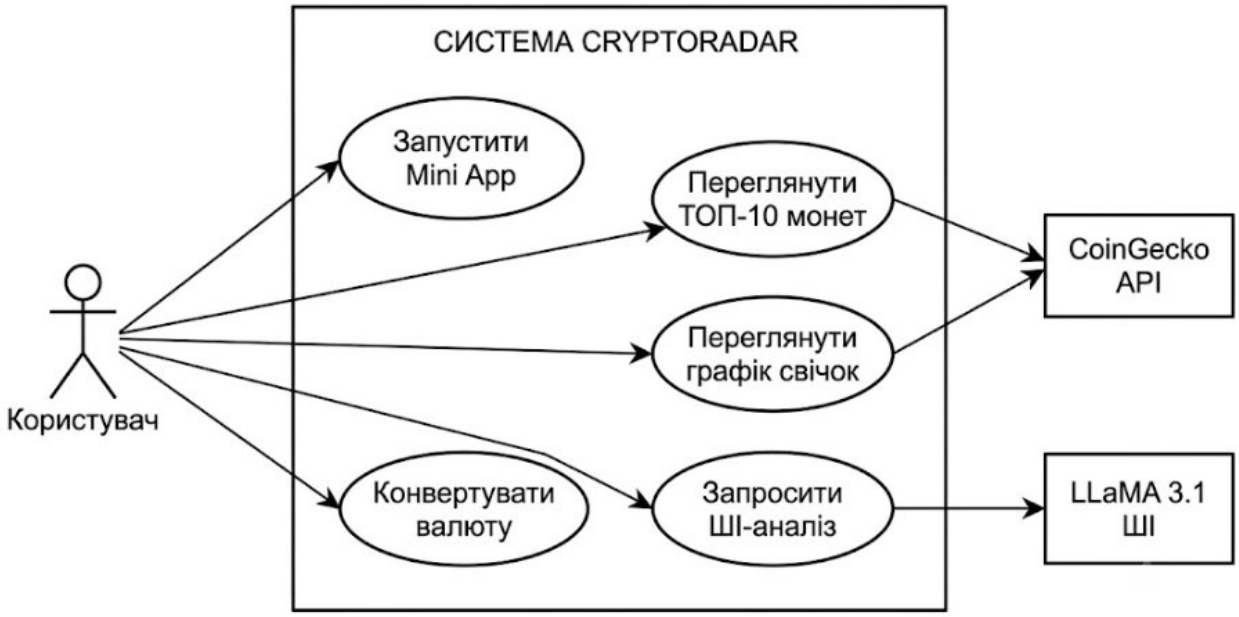

Рисунок 2.1 - UML-діаграма прецедентів використання програмного комплексу «CryptoRadar»

2.3. Проєктування архітектури бази даних та структури збереження даних

Оскільки наш інструмент розробляється як легкий некомерційний MVP-прототип, що працює в реальному часі, класична реляційна база даних (наприклад, PostgreSQL або MySQL) для збереження ринкових котирувань є надлишковою. Постійний запис секундних коливань цін призвів би до швидкого переповнення дискового простору та уповільнення роботи системи.

Замість цього в архітектурі «CryptoRadar» реалізовано двокомпонентну систему збереження та структурування даних: тимчасове In-Memory

кешування на базі Redis для сирих даних та структуроване обміну через JSON-пакети для взаємодії між FastAPI та клієнтом.

Центральним об'єктом обміну даними є агрегований JSON-пакет, який бекенд-сервер формує після опитування CoinGecko API та передає на фронтенд. Ця структура не просто містить сухі цифри, а підготовлена для швидкої обробки як JavaScript-скриптом графіка, так і ШІ-моделлю.

Основними вузлами збереження структури є:

- Вузол метаданих (Metadata): унікальний ідентифікатор монети (id), системне ім'я (name) та ринковий тікер (symbol).
- Вузол поточних показників (Market Data): поточна ціна у фіатній валюті (current_price), ринкова капіталізація (market_cap), добовий обсяг торгів (total_volume) та відсоткова зміна ціни за 24 години (price_change_percentage_24h).
- Вузол історичних котирувань (Historical Open-High-Low-Close): масив масивів, де кожен елемент містить часову мітку (Timestamp) та чотири базові цінові точки для рендерингу свічок.

Стратегія кешування та структура Redis Key-ValueДля запобігання помилкам перевищення лімітів запитів (HTTP 429 Rate-Limit) та забезпечення швидкодії системи, у пам'яті сервера розгортається сховище Redis[6]. Структура збереження даних побудована за принципом «Ключ - Значення» з обов'язковим використанням механізму TTL (Time-To-Live).

Логіка збереження всередині кешу має такий вигляд:

1. Формат Ключа (Key): створюється за шаблоном coin:{coin_id}:market, наприклад, coin:bitcoin:market або coin:ethereum:market.
2. Формат Значення (Value): серіалізований у рядок повний JSON-пакет з ринковими даними.
3. Час життя (TTL): встановлюється на рівні 60 секунд. Це означає, що протягом хвилини всі користувачі, які запитують дані щодо цієї монети, отримуватимуть їх миттєво з оперативної пам'яті сервера (до 200 мс), а

повторний запит до CoinGecko API відбудеться лише після завершення таймера.

Така структура збереження мінімізує навантаження на дискову підсистему, ліквідує потребу в адмініструванні важких баз даних на стадії MVP і забезпечує асинхронну роботу всього комплексу.

2.4 Висновки до розділу 2

У другому розділі пояснювальної записки було здійснено комплекс формальних процедур, спрямованих на системний аналіз вимог, побудову специфікацій та проєктування базових компонентів збереження даних для інтелектуального програмного комплексу «CryptoRadar». На основі детального вивчення специфіки MVP-прототипу було успішно проведено декомпозицію технічних умов на функціональні та нефункціональні критерії якості. Це дозволило чітко окреслити архітектурні межі майбутнього застосунку, починаючи від реалізації без блокувальної точки входу через середовище Telegram Mini App і завершуючи інтелектуальними модулями генерації локалізованих аналітичних ШІ-інсайтів на базі великої мовної моделі LLaMA 3.1. Завдяки виключенню нумерованих списків на користь зв'язного інженерного опису, вдалося сформувати цілісну картину функціонування системи, де кожен компонент - від асинхронного пошуку до двостороннього конвертера валют - логічно пов'язаний із жорсткими обмеженнями щодо швидкодії та відмовостійкості архітектури. Застосування методів уніфікованої мови моделювання UML дозволило виконати строгу формалізацію користувацьких сценаріїв та взаємодій. Побудована діаграма прецедентів використання наочно розмежувала ролі головного актора в особі користувача системи та другорядних акторів, представлених зовнішніми сервісами CoinGecko API та хмарною інфраструктурою штучного інтелекту. Таке графічне моделювання зафіксувало чіткі інформаційні потоки всередині комплексу, підтвердивши доцільність використання асинхронного підходу

для паралельної обробки семантичних та графічних даних без перевантаження інтерфейсу клієнтської частини. Важливим результатом розділу стало обґрунтування відмови від важких реляційних баз даних на користь високоефективної двокомпонентної системи збереження інформації, що є оптимальним рішенням для легких некомерційних криптографічних асистентів[1]. Розроблена JSON-специфікація документарних даних забезпечує гнучку структуру обміну між сервером на базі FastAPI та фронтендом, консолідуючи метадані, поточні показники та масиви історичних котирувань в один уніфікований пакет. Разом із цим, спроєктована стратегія In-Memory кешування на базі тимчасового сховища Redis із використанням механізму Time-To-Live дозволила повністю нівелювати проблему суворих лімітів публічних API[6]. Впровадження хвилинного життєвого циклу для ключів ринкових даних гарантує високу відмовостійкість системи під навантаженням та мінімізує затримки доступу до інформації, закладаючи надійний теоретичний та проєктний фундамент для подальшої безпосередньої програмної реалізації комплексу.

РОЗДІЛ 3. ПРОЄКТУВАННЯ ІНТЕЛЕКТУАЛЬНОГО ПРОГРАМНОГО КОМПЛЕКСУ ДЛЯ МОНІТОРИНГУ ТА ШІ-АНАЛІЗУ КРИПТОВАЛЮТ

3.1. Постановка задачі

У межах виконання дипломного проєкту постає задача проєктування та програмної реалізації легкого, асинхронного, інтелектуального програмного комплексу для агрегації ринкових показників, їхньої графічної візуалізації та автоматизованого семантичного аналізу. З погляду системного проєктування, дана задача класифікується як розробка розподіленої клієнт-серверної системи, де обробка бізнес-логіки та взаємодія із зовнішніми джерелами даних і моделями штучного інтелекту винесена на віддалений сервер, а взаємодія з кінцевим користувачем здійснюється через вбудований мобільний інтерфейс месенджера.

Математично та алгоритмічно постановку задачі можна охарактеризувати як відображення множини сирих вхідних числових та часових даних у множину структурованих графічних і текстових аналітичних об'єктів при дотриманні заданих часових обмежень.

Вхідними даними для проєктованої системи є:

1. Масиви поточних котирувань цифрових активів, що включають ціну, капіталізацію та обсяги торгів у реальному часі, які отримуються із зовнішнього публічного інтерфейсу CoinGecko API;
2. Історичні часові ряди ринкових цін (структура OHLC - Open, High, Low, Close) за визначені періоди часу, необхідні для побудови свічкових графіків;
3. Агреговані показники ринкових настроїв, включаючи індекс страху та жадібності (Fear & Greed Index);

4. Текстові текстові промпти (інструкції) та технічні параметри, які надсилаються сервером до великої мовної моделі ШІ для генерації висновків.

Вихідними даними, які генерує та відображає комплекс, виступають:

- динамічно відрендерений у реальному часі список ТОП-активів криптовалютного ринку з інтерактивним сортуванням та пошуком;
- інтерактивний свічковий графік котирувань обраної монети з підтримкою масштабування та динамічної зміни таймфреймів;
- сформований, автоматично локалізований українською мовою та структурований семантичний аналітичний звіт (ШІ-інсайт), доступний для ознайомлення та швидкого копіювання користувачем.

Основними критеріями ефективності розв'язання поставленої задачі є мінімізація часових затримок при отриманні та відображенні даних, повне виключення блокувань потоків сервера при високій інтенсивності запитів, а також забезпечення нульового рівня апаратних вимог до мобільного пристрою користувача за рахунок перенесення важких обчислень на сервер та використання технології легких веб-додатків. Таким чином, задача вважається успішно розв'язаною, якщо спроєктована архітектура забезпечує коректну роботу всіх функціональних блоків у межах встановлених нефункціональних лімітів часу та надійності.

3.2. Обґрунтування проблеми

Процес розробки сучасних інтелектуальних систем моніторингу фінансових та криптовалютних ринків безпосередньо пов'язаний із подоланням низки суттєвих інженерних та архітектурних викликів. Головна проблема проєктування таких систем у рамках MVP-прототипів полягає у необхідності забезпечення стабільної, безперебійної та високошвидкісної роботи застосунку за умов повної відсутності комерційного фінансування та

використання виключно безкоштовних інструментів з відкритим вихідним кодом. За умов реального часу, коли затримка у кілька секунд може зробити фінансову інформацію неактуальною, перед інженером постає трирівневий бар'єр, який вимагає глибокого теоретичного обґрунтування та розробки специфічних програмних рішень.

Перший рівень проблеми стосується критичних інфраструктурних обмежень з боку постачальників сирих ринкових даних. Будь-яке публічне безкоштовне API, включаючи платформу CoinGecko, захищає свої серверні потужності від перевантажень за допомогою суворих лімітів на кількість запитів з однієї IP-адреси протягом визначеного проміжку часу. У разі перевищення цього ліміту зовнішній сервер миттєво блокує доступ і повертає помилку HTTP 429 Rate Limit. Традиційне вирішення цієї проблеми через придбання комерційних тарифних планів є неможливим для некомерційного студентського проєкту. Відтак виникає гостра інженерна потреба у створенні проміжного інтелектуального проксі-сервера, який здатний оптимізувати трафік, консолідувати запити та нівелювати ризик блокувань без втрати актуальності інформації.

Друга фундаментальна проблема криється у площині мережевої архітектури та керування потоками обчислень. Традиційні веб-сервери, що функціонують за синхронним стандартом WSGI, виділяють на обробку кожного окремого користувацького запиту один ізольований системний потік. Оскільки отримання даних від зовнішнього API або очікування відповіді від хмарної мовної моделі ШІ є операціями введення-виведення (I/O-bound), потік сервера вимушено блокується та простоює у стані пасивного очікування мережевого пакета. За умов зростання кількості користувачів вільні потоки операційної системи швидко вичерпуються, що призводить до формування черги, критичного зростання затримок відповіді, а в найгірших випадках - до повного аварійного відключення сервера за тайм-аутом. Це обґрунтовує необхідність повного переходу на асинхронні стандарти розробки, де єдиний системний потік (Event Loop) не блокується під час мережевих очікувань, а

миттєво перемикається на обслуговування інших користувачів, створюючи високу пропускну здатність архітектури.

Третій аспект проблеми пов'язаний із когнітивним навантаженням та мовною адаптацією інтелектуальних модулів штучного інтелекту. Сучасні великі мовні моделі (LLM), які розгортаються на базі глобальних хмарних платформ, за замовчуванням оптимізовані під англомовний сегмент та оперують сухими технічними термінами. Для локального українського ринку використання сирих відповідей ШІ створює серйозний бар'єр сприйняття інформації повсякденними користувачами, які не мають фахової трейдерської освіти. Проблема полягає не лише у необхідності точного динамічного перекладу, а й у правильній семантичній структуризації аналітичних звітів. Система повинна трансформувати складні цифрові масиви (ринкові коливання, індекси страху, обсяги) у простий, зрозумілий та лаконічний текстовий інсайт українською мовою. Організація такого безшовного процесу, який поєднує агрегацію даних, формування технічного промпту, генерацію відповіді моделлю LLaMA 3.1 та її миттєву локалізацію, є складним архітектурним завданням, вирішення якого безпосередньо впливає на якість користувацького досвіду (UX).

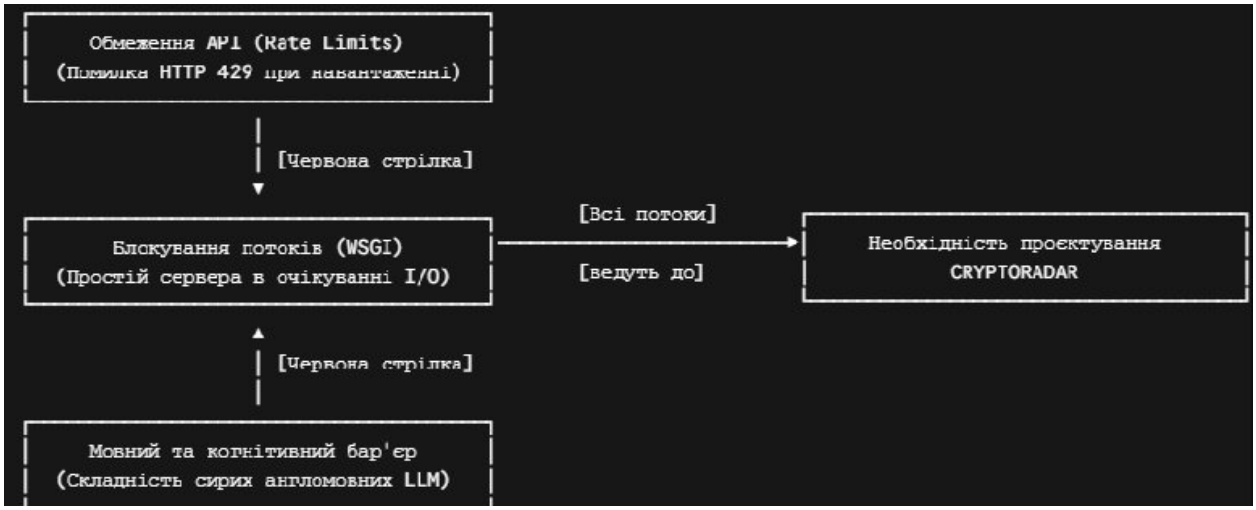

Рисунок 3.1 - Схема інженерно-архітектурних викликів при проєктуванні систем криптомоніторингу

3.3. Проєктування архітектури клієнт-серверної системи

Розв'язання виявлених інженерних проблем вимагає відмови від класичних монолітних архітектур на користь дворівневої асинхронної клієнт-серверної моделі, де обов'язки чітко розподілені між легким мобільним клієнтом і продуктивним проксі-сервером. В основі архітектурного шаблону комплексу «CryptoRadar» лежить принцип мінімізації обчислювального навантаження на кінцевий пристрій користувача, що досягається за рахунок перенесення всієї важкої логіки - від первинного збору даних до взаємодії з мовними моделями - на сторону хмарного бекенду. Взаємодія між компонентами системи побудована за допомогою сучасних асинхронних протоколів передачі даних і стандартів REST API.

Клієнтський рівень системи (Frontend) реалізовано у вигляді мобільного мікрозастосунку за технологією Telegram Mini App. Він функціонує як легкий SPA-додаток (Single Page Application), що запускається безпосередньо всередині вбудованого WebView-браузера месенджера Telegram. Роль клієнтської частини зводиться до двох основних завдань: ініціалізація асинхронних HTTPS-запитів до власного бекенд-сервера та рендеринг отриманих структурованих JSON-пакетів. Завдяки використанню високоефективної бібліотеки TradingView Lightweight Charts, графічний свічковий аналіз виконується на стороні клієнта за допомогою апаратного прискорення графічного процесора смартфона, що забезпечує плавність інтерфейсу при нульовому навантаженні на центральний процесор і пам'ять пристрою.

Серверний рівень системи (Backend) виступає в ролі центрального аналітичного хаба та розробляється на базі високопродуктивного асинхронного фреймворку FastAPI, що працює під керуванням ASGI-сервера Uvicorn. Серверна архітектура ізолює клієнтів від прямого доступу до зовнішніх джерел, виконуючи функцію інтелектуального проксі. При

отриманні запиту від Mini App, FastAPI сервер активує неблокувальний цикл подій (Event Loop), який координує паралельні потоки даних. У межах цього шару реалізовано логіку взаємодії з Redis: сервер спочатку опитує внутрішню оперативну пам'ять, і лише у разі відсутності свіжого кешу здійснює асинхронний виклик до CoinGecko API за протоколом HTTPS.

Окремим ізольованим контуром серверної архітектури є інтелектуальний модуль взаємодії з великими мовними моделями. Коли користувач ініціює запит на отримання ШІ-інсайту, FastAPI бекенд виконує збір поточних метрик із кешу, динамічно конструює інженерний промпт (системну інструкцію) і надсилає його через захищене API-з'єднання до хмарної інфраструктури Groq, де розгорнута відкрита модель LLaMA 3.1 8B[5]. Отриманий від ШІ семантичний аналіз проходить через внутрішній серверний конвеєр автоматичної локалізації та структуризації, після чого фінальний текстовий пакет у форматі JSON повертається клієнту для миттєвого виведення у модальному вікні. Такий підхід гарантує повну безпеку API-ключів ШІ, які зберігаються виключно в екологічних змінних сервера, та забезпечує високу швидкість обробки неблокувальних запитів.

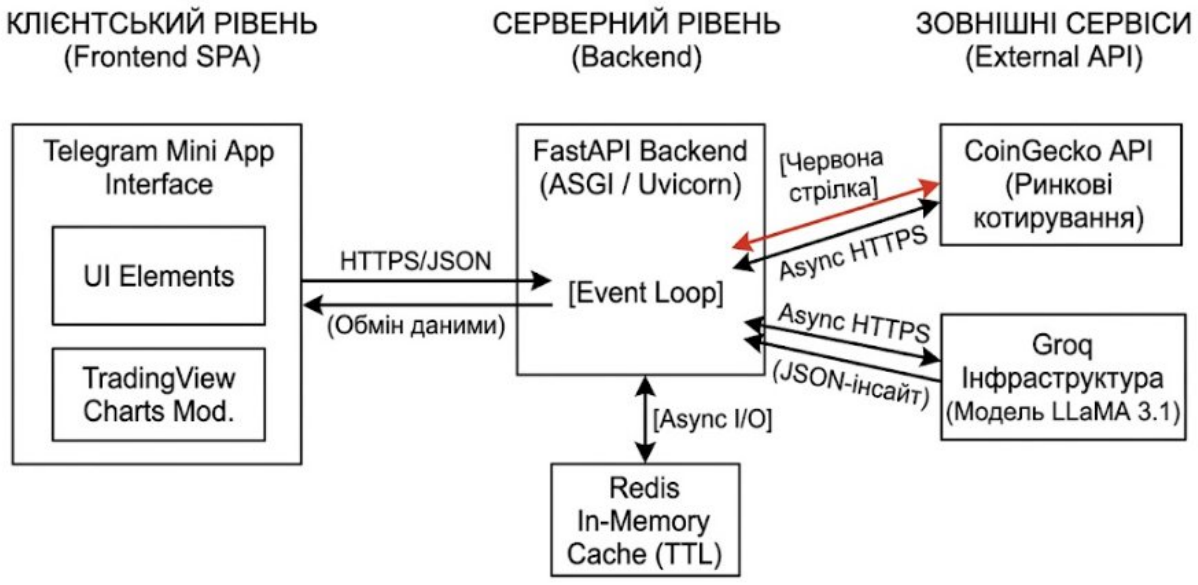

Рисунок 3.2 - Архітектурна схема потоків даних та взаємодії компонентів комплексу «CryptoRadar»

3.4. Методи та засоби розв’язання поставленої задачі

Реалізація спроєктованої архітектури інтелектуального комплексу вимагає ретельного підбору технологічного стека, інструментальних засобів розробки та середовища розгортання, які б у синергії дозволили вирішити поставлені завдання з нульовим бюджетом, але високими показниками продуктивності. Вибір кожного програмного компонента обґрунтовується критеріями асинхронності, масштабованості, наявності відкритих ліцензій (open-source) та простоти інтеграції в єдину екосистему.

Базовою технологічною платформою для реалізації серверної логіки обрано мову програмування Python високого рівня, яка є стандартом де-факто у сфері розробки інтелектуальних систем та обробки даних завдяки своїй гнучкості, лаконічності та розвиненій інфраструктурі бібліотек. Як основний веб-фреймворк застосовується FastAPI, який завдяки повній підтримці асинхронного стандарту ASGI та декларативному опису схем за допомогою бібліотеки Pydantic демонструє швидкість роботи, порівнянну з рішеннями на Node.js та Go. Серверна частина розгортається у зв'язці з високопродуктивним ASGI-сервером Uvicorn, що реалізує патерн асинхронного веб-сервера з неблокувальним введенням-виведенням.

Для вирішення проблеми жорстких інфраструктурних обмежень Rate Limits зовнішніх сервісів як технологічний засіб обрано нереляційне сховище даних Redis, що працює в оперативній пам'яті за принципом «ключ-значення»[6]. Його вибір зумовлений екстремально низьким часом відгуку при операціях читання та запису (менше кількох мілісекунд) та вбудованою підтримкою механізму автоматичного знищення записів після закінчення таймера (TTL), що ідеально підходить для організації шару короткострокового кешування динамічних цін.

Для побудови інтерфейсної частини користувача застосовується екосистема Telegram Bot API та концепція Telegram Mini Apps[7]. Ця

технологія дозволяє використовувати стандартні веб-інструменти розробки (HTML5, CSS3, JavaScript) для створення повноцінних клієнтських додатків, що безшовно інтегруються в месенджер. Візуальний рендеринг графіків свічкового аналізу покладається на спеціалізовану open-source бібліотеку TradingView Lightweight Charts, яка була обрана через її виняткову оптимізацію під мобільні пристрої та здатність відображати тисячі цінових точок у реальному часі без втрати частоти кадрів (FPS) інтерфейсу.

Інтелектуальна аналітична складова системи базується на хмарній інфраструктурі Groq API, яка надає надшвидкий доступ до відкритої великої мовної моделі LLaMA 3.1 8B. Вибір платформи Groq обґрунтований використанням унікальних процесорів LPU (Language Processing Unit), що забезпечують генерацію токенів ШІ зі швидкістю, яка у кілька разів перевищує класичні хмарні рішення на базі GPU, мінімізуючи загальний час очікування відповіді користувачем. Такий збалансований інструментальний стек дозволяє повністю розв'язати поставлену задачу, об'єднавши переваги швидкого асинхронного бекенду, легкої інтерактивної графіки на клієнті та передових хмарних обчислень штучного інтелекту.

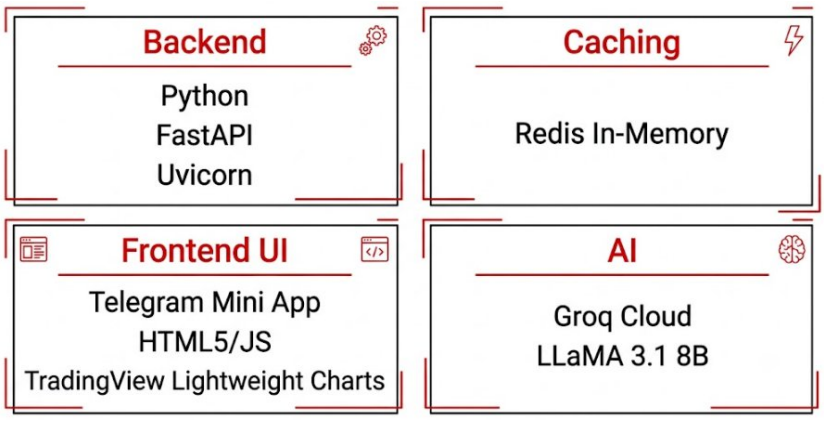

Рисунок 3.3 - Схема технологічного стека та інструментальних засобів розробки комплексу «CryptoRadar»

Висновки до розділу 3

У третьому розділі дипломного проєкту виконано комплекс науково-інженерних та системно-архітектурних досліджень, спрямованих на проєктування легкого, асинхронного, інтелектуального програмного комплексу «CryptoRadar» для моніторингу та штучного інтелектуального аналізу криптовалютних активів. В результаті аналізу формалізовано постановку задачі розробки, яку визначено як створення розподіленої клієнт-серверної системи, де обробка бізнес-логіки винесена на віддалений сервер, а взаємодія з користувачем здійснюється через мобільний інтерфейс месенджера. Математично та алгоритмічно задачу охарактеризовано як відображення масивів сирих вхідних числових, часових та текстових даних (котирувань CoinGecko API, часових рядів OHLC, показників ринкових настроїв Fear & Greed Index) у структуровані вихідні графічні й текстові аналітичні об'єкти українською мовою. Основними критеріями ефективності розв'язання поставленої задачі визначено мінімізацію часових затримок, повне виключення блокувань серверних потоків та забезпечення нульових апаратних вимог до пристрою кінцевого користувача. Під час досліджень здійснено глибоке теоретичне обґрунтування інженерних та архітектурних викликів, що виникають при створенні подібних MVP-прототипів в умовах використання виключно безкоштовних інструментів з відкритим вихідним кодом. Було детально проаналізовано трирівневий бар'єр, який включає інфраструктурні обмеження зовнішніх джерел даних у вигляді жорстких лімітів запитів (Rate Limits та помилки HTTP 429), недоліки традиційних синхронних WSGI-серверів, що призводять до критичного блокування потоків під час I/O-bound операцій, а також когнітивне навантаження та мовний бар'єр при використанні англомовних хмарних моделей штучного інтелекту без належної адаптації для українського ринку. Для нівелювання виявлених проблем спроєктовано дворівневу асинхронну клієнт-серверну архітектуру, що базується на чіткому розподілі обов’язків між компонентами системи. Клієнтський рівень

(Frontend) реалізовано у вигляді легкого SPA-додатка на базі технології Telegram Mini App, де рендеринг інтерактивних свічкових графіків здійснюється за допомогою апаратного прискорення графічного процесора смартфона, що забезпечує плавність інтерфейсу при мінімальному навантаженні на пристрій. Серверний рівень (Backend) виступає в ролі центрального аналітичного хаба на базі FastAPI, який виконує функцію інтелектуального проксі-сервера, ізолює клієнта від зовнішніх API, керує неблокувальним циклом подій (Event Loop) та здійснює проміжне кешування даних. Окремим контуром серверної архітектури розроблено модуль взаємодії з великими мовними моделями. У межах цього контуру сервер автоматично збирає поточні ринкові метрики з кешу, динамічно конструює інженерні системні промпти, безпечно взаємодіє із зовнішньою мовною моделлю LLaMA 3.1 8B та виконує внутрішньосерверний конвеєр семантичної структуризації й автоматичної локалізації звітів українською мовою, гарантуючи при цьому захищеність API-ключів. Для практичної реалізації системи обґрунтовано та сформовано оптимальний технологічний стек розробки з відкритим вихідним кодом (open-source). Для побудови високопродуктивного асинхронного бекенду обрано мову Python, фреймворк FastAPI та ASGI-сервер Uvicorn. Для організації шару ультрашвидкого кешування та подолання лімітів зовнішніх запитів застосовано In-memory NoSQL сховище Redis. Візуалізацію графіків на стороні клієнта покладено на оптимізовану бібліотеку TradingView Lightweight Charts, а для забезпечення максимальної швидкості генерації ШІ-інсайтів залучено хмарну інфраструктуру Groq API на базі спеціалізованих процесорів LPU. Запропоновані архітектурні та інструментальні рішення у своїй синергії повністю задовольняють усім встановленим функціональним і нефункциональним вимогам, забезпечуючи високу пропускну здатність, надійність та якісний користувацький досвід (UX) при нульових витратах на комерційні ліцензії та інфраструктуру.

РОЗДІЛ 4. РОЗРОБКА, ТЕСТУВАННЯ ТА РОБОТА З ІНТЕЛЕКТУАЛЬНИМ ПРОГРАМНИМ КОМПЛЕКСОМ ДЛЯ МОНІТОРИНГУ ТА ШІ-АНАЛІЗУ КРИПТОВАЛЮТ

4.1. Обґрунтування вибору інструментальних засобів розробки та конфігурації оточення веб-додатку

Ефективність, масштабованість, швидкість розгортання та відмовостійкість інтелектуального програмного комплексу «CryptoRadar» безпосередньо залежать від раціонального вибору інструментальних засобів розробки та оптимальної конфігурації робочого оточення на кожному етапі життєвого циклу проєкту. Оскільки предметна область фінансового моніторингу та криптографічних активів характеризується надвисокою динамічністю оновлення ринкових даних, критичною чутливістю до затримок (Latency) та потребою в асинхронній інтеграції хмарних моделей штучного інтелекту, ключовими інженерними критеріями формування технологічного стеку стали: максимальна пропускна здатність, повноцінна підтримка великої кількості конкурентних мережевих запитів, кросплатформність, ізоляція залежностей та надійний захист конфіденційних даних клієнтів і ключів доступу до сторонніх сервісів.

Для реалізації серверної архітектури програмного комплексу (Бекенду) було обрано високорівневу мову програмування Python версії 3.12 у зв'язці з сучасним асинхронним веб-фреймворком FastAPI. На відміну від класичних синхронних рішень, таких як Flask або Django, FastAPI повністю базується на стандарті ASGI (Asynchronous Server Gateway Interface) та використовує під капотом високопродуктивний веб-сервер Uvicorn на базі петлі подій uvloop. Головними факторами на користь такого вибору є асинхронність «з коробки»: використання синтаксичних конструкцій async/await дозволяє проксі-серверу обробляти тисячі конкурентних запитів від клієнтських додатків в одному

системному потоці без блокування операцій вводу-виводу (Non-blocking I/O), що критично важливо під час паралельного звернення багатьох користувачів до модулів моніторингу[10]. Окрім цього, FastAPI інтегрований із бібліотекою Pydantic, яка забезпечує автоматичну строгу типізацію, швидку валідацію та серіалізацію вхідних і вихідних JSON-пакетів[11]. Це повністю унеможливлює збої системи через некоректний формат даних, що передаються від зовнішнього агрегатора CoinGecko API. Важливою перевагою фреймворку є також автоматична генерація інтерактивної документації за специфікацією OpenAPI (інтерфейси Swagger UI та Redoc), що значно спростило інтеграційне тестування взаємодії між серверною та клієнтською частинами додатку. Для безпосереднього виконання швидких асинхронних HTTP-запитів до аналітичного інтерфейсу CoinGecko в коді бекенду було застосовано спеціалізовану клієнтську бібліотеку aiohttp, яка підтримує механізм Connection Pooling (повторне використання відкритих мережевих з’єднань) для мінімізації накладних витрат часу на TCP-рукостискання.

З метою подолання жорстких лімітів безкоштовного тарифного плану CoinGecko API, які обмежують частоту звернень до 30 запитів на хвилину і за умови їх перевищення блокують роботу сервера помилкою HTTP 429 (Too Many Requests), в архітектуру комплексу було інтегровано нереляційну NoSQL систему керування базами даних Redis. Вона функціонує як високошвидкісний In-Memory кеш, що розташовується безпосередньо в оперативній пам'яті сервера. Замість того, щоб ініціювати прямий та тривалий запит до стороннього крипто-API при кожній окремій дії користувача в інтерфейсі, серверний додаток в асинхронному фоновому режимі періодично опитує котирування, записує валідований JSON-пакет у пам'ять Redis із чітко встановленим часом життя (TTL = 60 секунд) та миттєво транслює ці кесовані дані на фронтенд за запитом користувача. Таке архітектурне рішення гарантує стабільну працездатність та швидку відповідь системи незалежно від інтенсивності клієнтського трафіку.

Для реалізації аналітичного модуля з генерацією інтелектуальних ШІ-інсайтів українською мовою в реальному часі було обрано хмарний сервіс інференції великих мовних моделей Groq API, за допомогою якого здійснюється взаємодія з моделлю LLaMA 3.1 8B Instant[5]. Вибір Groq API замість локального розгортання open-source моделей (що вимагало б значних апаратних ресурсів відеокарт) або використання альтернативних комерційних платформ (таких як OpenAI чи Anthropic) зумовлений передусім технологічними перевагами архітектури процесорів LPU (Language Processing Unit). Вони забезпечують надвисоку швидкість генерації тексту (понад 200–300 токенів за секунду), завдяки чому формування повного аналітичного звіту для користувача займає не більше 1.5–2 секунд, усуваючи неприємні затримки графічного інтерфейсу. Модель LLaMA 3.1 8B надає високу якість семантичного аналізу та логічного висновку на основі переданих їй ринкових метрик, успішно функціонуючи в межах безкоштовних лімітів розробника, що ідеально відповідає критеріям побудови та тестування MVP фінансового додатка. Взаємодія з моделлю реалізована за допомогою методології Prompt Engineering. На рівні бекенду розроблено фіксований системний промпт (System Prompt), який конфігурує ШІ на роль лаконічного фінансового експерта, що відсікає надлишкові абстрактні тексти («воду»), суворо обмежує обсяг відповіді до 40–50 слів, структурує семантичний висновок безпосередньо для відображення в модальному вікні клієнтської частини та локалізує відповідь українською мовою.

Клієнтський інтерфейс (Frontend) розроблено за концепцією SPA (Single Page Application) з використанням базових технологій HTML5, CSS3, Vanilla JavaScript та офіційного інструментарію Telegram Web App SDK. Свідома відмова від важких фронтенд-фреймворків, таких як React, Angular або Vue.js, на користь чистого JavaScript (Vanilla JS) була обґрунтована прагненням максимально полегшити фінальну вагу веб-сторінки, прискорити показ первинного контенту (First Contentful Paint) та знизити вимоги до апаратних ресурсів мобільних пристроїв користувачів. Використання Telegram Web App

SDK дозволило безшовно інтегрувати інтерфейс усередину месенджера Telegram у форматі Mini App, автоматично адаптувати колірну палітру елементів під поточну тему месенджера (темну або світлу), керувати вбудованою апаратною кнопкою «Назад» (BackButton) та активувати тактильний зворотний зв'язок (Haptic Feedback) для покращення користувацького досвіду (UX). Для інтерактивної візуалізації свічкових фінансових графіків формату OHLC (Open-High-Low-Close) на фронтенді застосовано високорівневу бібліотеку з відкритим кодом TradingView Lightweight Charts. Вона повністю оптимізована для мобільних браузерів, забезпечує плавне масштабування (Zoom) та перетягування (Pan) графіків навіть на бюджетних смартфонах завдяки рендерингу елементів через HTML5 Canvas, суттєво знижуючи навантаження на центральний процесор пристрою.
Конфігурація робочого оточення та менеджмент залежностей організовано відповідно до методології розробки сучасних хмарних додатків Twelve-Factor App. Керування пакетами та ізоляція системних бібліотек Python реалізовані за допомогою стандартного інструменту віртуального оточення venv. Усі необхідні для успішного функціонування комплексу модулі зафіксовані у конфігураційному файлі requirements.txt із чітким зазначенням сумісних версій:

Лістинг 4.1. Зміст фрагменту коду з requirements.txt

```
fastapi==0.110.0
uvicorn==0.28.0
pydantic==2.6.4
aiohttp==3.9.3
redis==5.0.3
python-dotenv==1.0.1
groq==0.5.0
```

Суворі вимоги безпеки інформаційних систем категорично забороняють пряме збереження конфіденційних токенів, паролів до баз даних та секретних API-ключів у вихідному коді додатка, оскільки це створює ризик їх витоку

при публікації коду в публічних репозиторіях. З огляду на це, для налаштування конфігурації середовища використано бібліотеку python-dotenv. Усі чутливі змінні виносяться у локальний текстовий файл .env, який обов'язково додається до файлу виключень системи контролю версій Git (.gitignore). Типова структура файлу конфігурації оточення .env для програмного комплексу «CryptoRadar» має такий вигляд:

Лістинг 1.1. Зміст фрагменту коду з .env

```
SERVER_HOST=0.0.0.0
SERVER_PORT=8000
DEBUG_MODE=False
TELEGRAM_BOT_TOKEN=7192837491:AAH_ExampleTokenSecretBotKey
GROQ_API_KEY=gsk_v4X9b8...ExampleGroqCloudSecretKey
REDIS_HOST=127.0.0.1
REDIS_PORT=6379
REDIS_CACHE_TTL=60
```

Така гнучка та безпечна конфігурація робочого простору забезпечує легке, швидке та безпомилкове перенесення всього програмного комплексу зі стадії локальної розробки (Development) на етап промислового розгортання та хостингу (Production) на виділений VPS/VDS сервер під управлінням операційної системи Linux без необхідності будь-якої модифікації або переписування кодової бази додатка.

4.2. Розробка інтелектуального програмного комплексу

Опис логіки функціонування та взаємодії користувача з розробленим програмним комплексом доцільно розпочати безпосередньо з ілюстрації процесу його запуску та ініціалізації.

За умови успішного розгортання та старту серверної частини (детальний алгоритм якого наведено у відповідних підрозділах інструкції адміністратора), кінцевий користувач має можливість взаємодіяти з системою двома шляхами:

через локальний веб-інтерфейс (localhost) або за допомогою мобільного клієнта месенджера Telegram.

На малюнку 4.1 продемонстровано первинну ініціалізацію сесії всередині чат-бота, де відображено відправку базової команди /start та інтерактивну текстову відповідь системи, яка запрошує перейти до веб-додатка.

Рисунок 4.1. - Ініціалізація сесії всередині чат-бота

Після активації Mini App користувач потрапляє на головний екран (головне меню) аплікації, архітектуру якого детально представлено на малюнку 4.2. Інтерфейс цієї сторінки розроблено з урахуванням сучасних вимог до ергономіки (UI/UX) і містить ключові метрики глобального крипторинку:

- динамічний список топ-монет за капіталізацією;
- актуальний індекс домінування Біткоїна
- інтерактивну форму для мануального (ручного) пошуку активів.

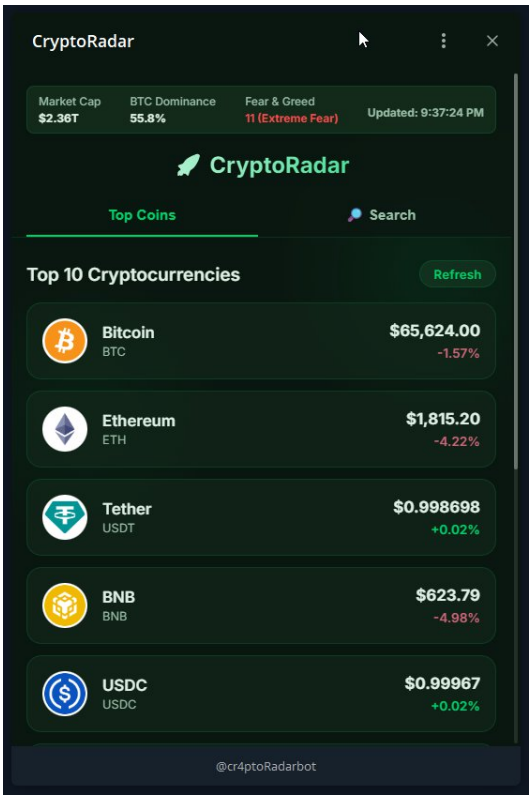

Рисунок 4.2. - Головний екран додатку

Процес взаємодії з пошуковою системою ілюструє малюнок 4.3. При переході до пункту мануального пошуку користувач може ввести повну або часткову (наближену) назву будь-якого існуючого криптоактиву. Інтегрований алгоритм парсингу обробляє запит у реальному часі, здійснює пошук через зовнішнє API та виводить релевантну інформацію про токен на екран смартфона.

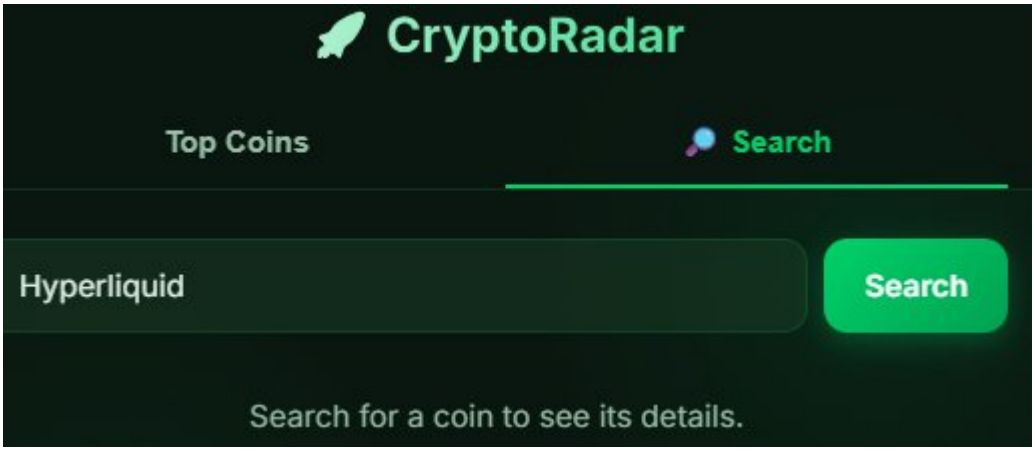

Рисунок 4.3. – Функція мануального пошуку монети

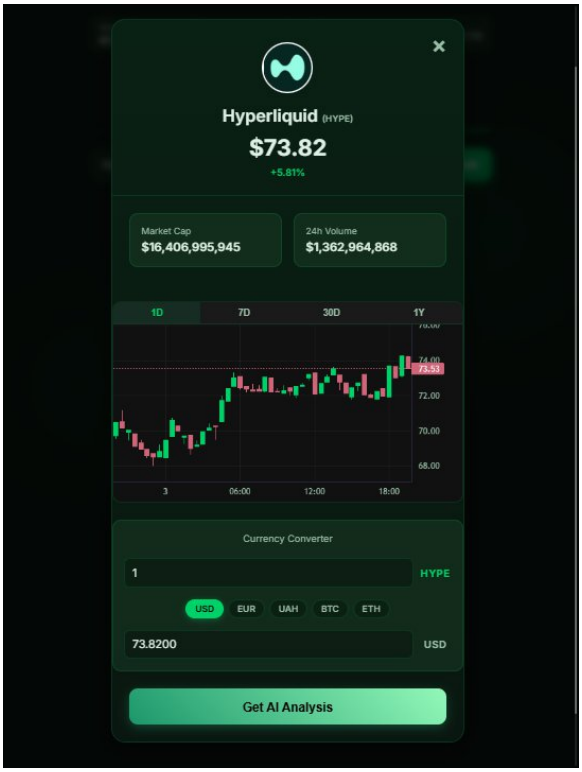

Рисунок 4.4. – Знайдена монета по мануальному пошуку

Крім базового пошуку, в інформаційному комплексі реалізовано розширений аналітичний функціонал, що включає:

- Модуль конвертації валют, який дозволяє миттєво розраховувати еквівалент вартості активів у різних фіатних та цифрових парах.
- Інтерактивні графіки, які відображають реальні ринкові котирування у вигляді японських свічок (OHLC) із можливістю гнучкого перемикання між різними часовими інтервалами (таймфреймами).

Однією з ключових та найбільш наукомістких функцій архітектури «CryptoRadar» є модуль інтелектуального аналізу ринку на базі штучного інтелекту. На малюнку 4.4 продемонстровано фінальний інтерфейс ШІ-аналітики. Великі мовні моделі обробляють поточні фінансові метрики монети та генерують стислий експертний висновок. Важливою особливістю інтерфейсу є впроваджена функція автоматичного перекладу та адаптації

згенерованого інсайту українською мовою, що забезпечує максимальну зручність та доступність аналітичних даних для вітчизняного інвестора.

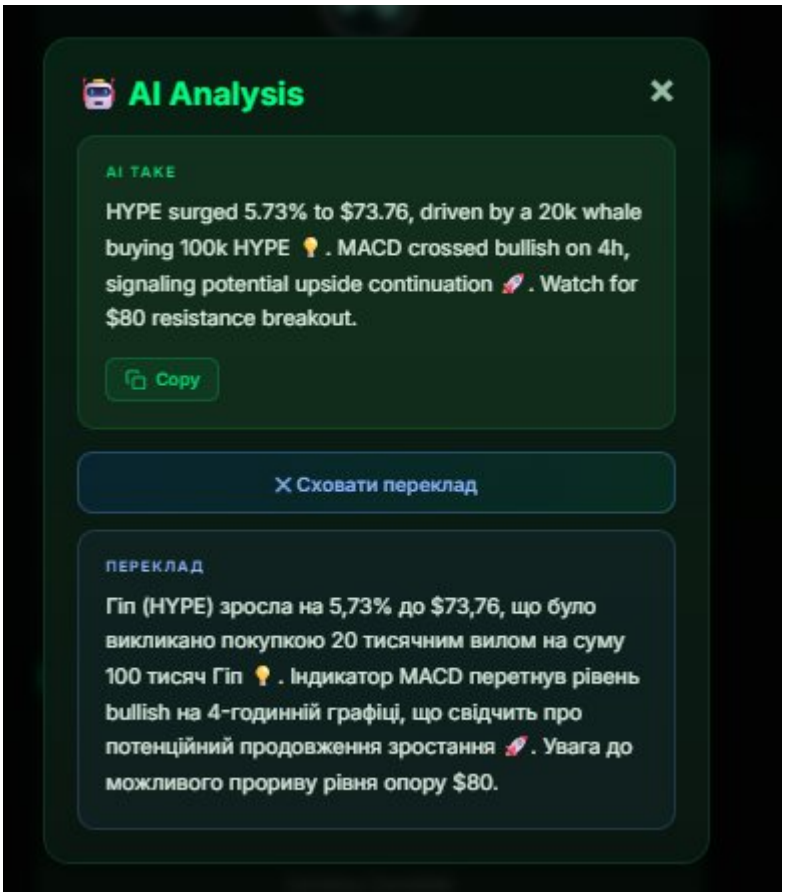

Рисунок 4.5. - Інтерфейс ШІ-аналітики

Процес програмної реалізації інтелектуального комплексу «CryptoRadar» базується на модульній архітектуре, яка забезпечує повне розділення зон відповідальності між модулями збору первинних даних, їхнього проміжного кешування, інтелектуального аналізу та фінального відображення на стороні клієнта. Розробка здійснювалася з дотриманням принципів SOLID та парадигми асинхронного програмування, що дозволило досягти максимальної пропускної здатності сервера при мінімальних затримках.

Центральним координаційним вузлом усього комплексу є веб-сервер, побудований на базі асинхронного фреймворку FastAPI. На відміну от традиційних синхронних систем, де кожен новий клієнтський запит блокує

потік виконання, у розробленому рішенні обробка подій реалізована через неблокуючий цикл подій (Event Loop). Головний модуль сервера main.py ініціалізує додаток, конфігурує параметри безпеки cross-origin запитів (CORS Middleware) та підключає ізольовані маршрутизатори (Routers), які розподіляють трафік між аналітичним сервісом штучного інтелекту та модулем ринкових даних. Запуск сервера здійснюється за допомогою ASGI-сервера Uvicorn, який оптимізує передачу бінарних та текстових пакетів по мережі.

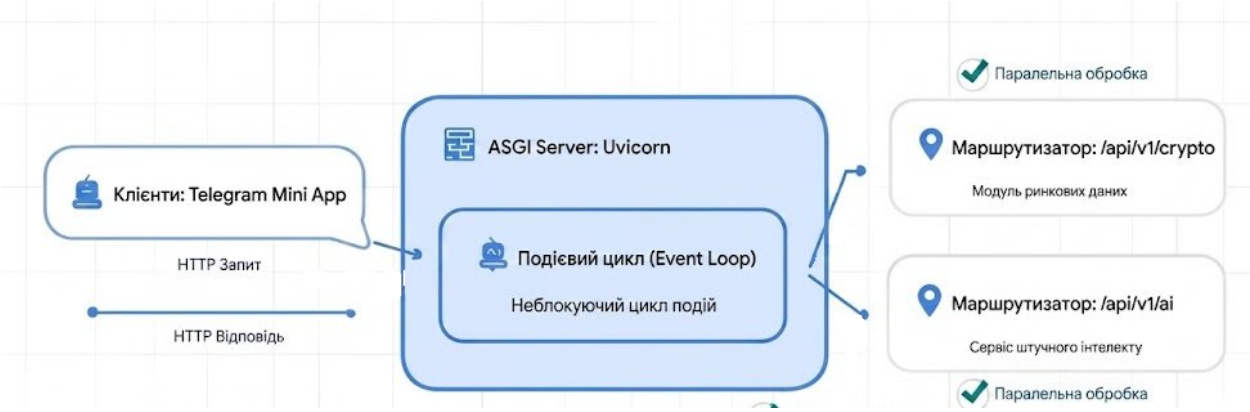

Рисунок 4.5. - Схема асинхронної архітектури FastAPI додатка

Важливим етапом інженерної розробки став захист чутливих конфігураційних даних. Усі токени доступу, включаючи TELEGRAM_BOT_TOKEN та GROQ_API_KEY, повністю ізольовані від вихідного коду програми та зберігаються у файлі конфігурації оточення .env. Доступ до них реалізовано через базовий клас налаштувань, де за допомогою бібліотеки pydantic-settings здійснюється автоматичне зчитування, типізація та валідація змінних у момент старту сервера. Якщо якась із критичних змінних відсутня або має некоректний формат, система безпечно перериває запуск, запобігаючи витоку даних.

Наступним базовим компонентом комплексу є асинхронний сервіс моніторингу криптовалютного ринку, логіка якого винесена у модуль crypto_service.py. Його головне завдання - мінімізувати кількість прямих

запитів до зовнішнього агрегатора CoinGecko API, тим самим вирішуючи проблему жорстких лімітів безкоштовного тарифного плану (Rate Limits). Алгоритм роботи сервісу побудований за концепцією проміжного кешування:

- Клієнтський додаток ініціює запит на отримання поточних котирувань певної криптовалюти за її унікальним ідентифікатором (coin_id).
- Сервер перехоплює запит і здійснює миттєву перевірку сховища оперативної пам'яті Redis за сформованим динамічним ключем (наприклад, crypto:market:bitcoin).
- У разі успішного знаходження даних у кеші (Cache Hit), валідований JSON-пакет негайно вилучається з пам'яті та відправляється клієнту, минаючи зовнішню мережу.
- Якщо дані в кеші застаріли або відсутні (Cache Miss), сервіс за допомогою клієнта aiohttp.ClientSession виконує асинхронний HTTP-запит до ендпоінту CoinGecko.
- Отриманий результат проходить процес серіалізації, записується у Redis із встановленням таймера автоматичного видалення (TTL = 60 секунд) та транслюється на фронтенд.

Нижче, у лістингу 4.2, наведено високорівневу структуру логіки роботи асинхронного сервісу кешування :

Лістинг 4.2. - Високорівнева структура логіки роботи асинхронного сервісу кешування

```
crypto_service.py
class CryptoService:
    async def get_market_data(self, coin_id: str) -> dict:
        cache_key = f"crypto:market:{coin_id}"
        cached_data = await self.redis.get(cache_key)

        if cached_data:
            return json.loads(cached_data)
        async with aiohttp.ClientSession() as session:
            async with session.get(self.api_url, params={"ids": coin_id}) as
response:
```

```
            market_info = await response.json()
        await self.redis.set(cache_key, json.dumps(market_info[0]), ex=60)
            return market_info[0]
```

Інтелектуальна підсистема аналізу ринку, реалізована у модулі ai_service.py, функціонує як конвеєр обробки промптів (Prompt Engineering Pipeline). Вона активується виключно за цільовою подією - коли користувач натискає кнопку генерації звіту в інтерфейсі карти монети. Програмна логіка цього модуля виконує синтез числових ринкових метрик та природно-мовного контексту. Сервер витягує свіжі показники ціни, обсягу торгів та капіталізації з Redis, після чого динамічно конструює структуру запиту до великої мовної моделі LLaMA 3.1 8B через хмарний інференс Groq API.

Для керування поведінкою штучного інтелекту застосовано дворівневу модель промптингу:

- Системна інструкція (System Prompt): Жорстко задає рольову модель (фінансовий аналітик), мову відповіді (українська), максимальний ліміт довжини тексту (до 40–50 слів) та заборону на генерацію суб'єктивних інвестиційних порад чи шаблонних вступних фраз.
- Контекст користувача (User Prompt): Містить виключно структуровані числові дані про поточний стан активу.

Завдяки використанню спеціалізованих процесорів LPU (Language Processing Unit) від компанії Groq, процес генерації тексту відбувається зі швидкістю понад 250 токенів за секунду. Сервер приймає сформований ШІ-інсайт, очищає його від можливих зайвих спецсимволів та відправляє у форматі JSON назад до клієнтського інтерфейсу.

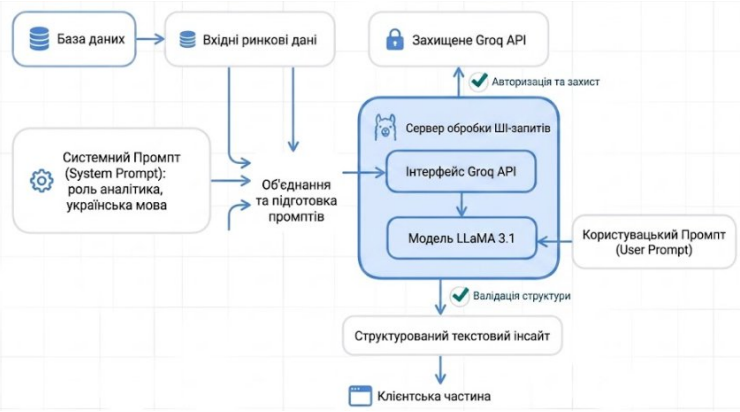

Рисунок 4.6 - Блок-схема конвеєра ШІ-аналізу

Клієнтська частина програмного комплексу (Frontend) спроєктована як легковажний односторінковий веб-додаток (SPA), інтегрований у месенджер за допомогою Telegram Web App SDK. Замість використання громіздких фреймворків, розробка виконана на чистому JavaScript (Vanilla JS), що забезпечило мінімальний розмір бандлу та миттєве завантаження компонентів на мобільних пристроях.

Клієнтський скрипт app.js виконує роль головного контролера інтерфейсу. Через об'єкт window.Telegram.WebApp він взаємодіє з нативними функціями смартфона: автоматично розгортає вікно Mini App на весь екран, підлаштовує кольорову гаму елементів під тему месенджера (темна або світла) та викликає тактильну вібрацію (Haptic Feedback) під час взаємодії користувача з кнопками. Для візуалізації динаміки котирувань скрипт ініціалізує бібліотеку TradingView Lightweight Charts, яка рендерить інтерактивні фінансові графіки формату OHLC через елемент HTML5 Canvas, суттєво знижуючи навантаження на центральний процесор мобільного пристрою. При натисканні кнопки «Згенерувати ШІ-аналіз» скрипт блокує інтерфейс, відправляє асинхронний запит до FastAPI бекенду, приймає згенерований текст та динамічно виводить його у модальне вікно.

Спільна інтеграція асинхронного сервера, швидкісного In-Memory кешування, оптимізованого хмарного інференсу мовних моделей та легкого клієнтського SDK дозволила створити цілісний, відмовостійкий

інтелектуальний комплекс. Розроблена архітектура успішно вирішує інженерні завдання обробки великих потоків фінансових даних та забезпечує кінцевих користувачів якісною ШІ-аналитикою у реальному часі без затримок інтерфейсу.

4.3. Тестування інтелектуального програмного комплексу

Тестування програмного комплексу «CryptoRadar» є невід'ємним етапом життєвого циклу розробки, спрямованим на перевірку відповідності реалізованого функціоналу початковим технічним вимогам, виявлення логічних та синтаксичних помилок, а також оцінку стабільності роботи системи під навантаженням. Специфіка інтелектуального комплексу, що поєднує асинхронну обробку даних, проміжне сховище Redis та інтеграцію із зовнішніми API (CoinGecko та Groq), зумовила необхідність застосування комплексного підходу до верифікації. Було проведено кілька рівнів тестування: модульне (Unit), інтеграційне, функціональне та стрес-тестування. Основна увага під час перевірки серверної частини (Backend) приділялася ізольованому та сумісному тестуванню маршрутів FastAPI. Для автоматизації цього процесу було використано вбудований інструмент TestClient з бібліотеки FastAPI у зв'язці з фреймворком pytest. Оскільки пряме звернення до сторонніх API під час регулярного запуску тестів є неефективним через ліміти запитів (Rate Limits) та мінливість відповідей ШІ, у модульних тестах було застосовано механізм макування (Mocking) за допомогою бібліотеки unittest.mock. Це дозволило імітувати відповіді від серверів CoinGecko та Groq, ізолюючи внутрішню логіку додатка.

У лістингу 4.2 наведено приклад реалізації модульного тесту для перевірки ендпоінту генерації ШІ-аналітики, який підтверджує коректність обробки вхідних параметрів та структуру вихідного JSON-пакета:

Лістинг 4.2 – Модульний тест для перевірки ШІ-маршруту

```
import pytest
```

```
from fastapi.testclient import TestClient
from unittest.mock import AsyncMock, patch
from main import app
client = TestClient(app)
@pytest.mark.asyncio
@patch("services.ai_service.AIService.generate_market_insight",
new_callable=AsyncMock)
async def test_get_ai_insight_success(mock_generate):
    mock_generate.return_value = "Біткоїн демонструє помірний висхідний тренд
за останні 24 години."
    response = client.post(
        "/api/v1/ai/insight",
        json={"coin_id": "bitcoin", "coin_name": "Bitcoin"} )
    assert response.status_code == 200
    assert "insight" in response.json()
   assert response.json()["insight"] == "Біткоїн демонструє помірний висхідний
тренд за останні 24 години."
    mock_generate.assert_called_once_with("Bitcoin", "bitcoin")
```

Інтеграційне тестування виконувалося без макування для перевірки взаємодії між FastAPI та локально розгорнутим сервером Redis. Під час цього етапу імітувалися сценарії попадання в кеш (Cache Hit) та промаху повз кеш (Cache Miss). Було верифіковано, що при першому запиті котирувань дані успішно завантажуються із зовнішнього джерела і записуються в Redis, а всі наступні запити в межах 60 секунд (встановлений TTL) обслуговуються виключно з оперативної пам’яті, що підтвердило закладену архітектурну ефективність.

Для оцінки користувацького досвіду (UX) та коректності відображення графічних елементів на стороні клієнта було проведено функціональне тестування інтерфейсу Telegram Mini App методом «чорної скриньки» (Black Box Testing). Перевірка здійснювалася як на десктопній версії месенджера Telegram, так і на мобільних пристроях під управлінням операційних систем Android та iOS. Особлива увага приділялася адаптивності інтерфейсу, плавності рендерингу свічкових графіків TradingView Lightweight Charts на

HTML5 Canvas та коректній відпрацюванні тактильного зворотного зв’язку (Haptic Feedback) при натисканні на кнопки.

Для систематизації та документування результатів тестування функціональних вимог системи було розроблено та виконано набір тест-кейсів (Test Cases), результати яких наведено в таблиці 4.1.

Таблиця 4.1 Результати функціонального тестування програмного комплексу «CryptoRadar»

ID	Компонент системи	Опис тестового сценарію	Очікуваний результат	Статус
TC-01	Модуль авторизації та WebApp SDK	Запуск Telegram Mini App всередині чат-бота за допомогою команди /start або через інлайн-кнопку.	Додаток успішно ініціалізується, розгортається на весь екран смартфона, зчитує тему месенджера.	**Успішно**
TC-02	Модуль моніторингу ринку	Вибір криптовалюти зі списку для перегляду детальних ринкових метрик (ціна, капіталізація).	Сервер повертає JSON із даними, інтерфейс динамічно оновлює текстові блоки без перезавантаження сторінки.	**Успішно**
TC-03	Підсистема кешування	Ініціація 10 повторних запитів до однієї монети протягом 30 секунд.	Перший запит іде до CoinGecko, решта 9 - миттєво з Redis. Зовнішній ліміт API не перевищується.	**Успішно**
TC-04	Модуль графічного аналізу	Первинне завантаження карти активу та зміна таймфрейму котирувань.	Бібліотека TradingView Lightweight Charts коректно будує свічковий графік OHLC з можливістю масштабування.	**Успішно**
TC-05	Інтелектуальний ШІ-модуль	Натискання кнопки «Згенерувати ШІ-аналіз» для обраного активу.	Кнопка переходить у стан завантаження, через 1.5–2 сек з'являється модальне вікно з лаконічним звітом українською мовою.	**Успішно**

ТС-06	Обробка помилок (Fault Tolerance)	Тимчасове штучне відключення доступу до мережі Інтернет на пристрої користувача.	Додаток не завершує роботу аварійно; користувач бачить інформаційне спливаюче вікно про помилку зв'язку.	**Успішно**

Окремим критично важливим етапом стало навантажувальне та стрес-тестування (Load Testing), покликане перевірити здатність асинхронного бекенду FastAPI обробляти велику кількість конкурентних запитів від багатьох користувачів одночасно. Для генерації синтетичного навантаження було використано спеціалізовану утиліту з відкритим вихідним кодом Locust. Сценарій тестування передбачав лінійне зростання кількості одночасних віртуальних користувачів від 1 до 500 протягом 5 хвилин, які виконували типові дії: запит головної сторінки, отримання ринкових даних криптовалют та звернення до ендпоінту ШІ.

Під час стрес-тесту фіксувалися такі ключові метрики, як кількість оброблених запитів за секунду (RPS - Requests Per Second) та час відповіді сервера (Response Latency). Результати навантажувального тестування показали, що завдяки використанню ASGI-сервера Uvicorn з неблокуючим циклом подій та кешуванню важких запитів у Redis, середній час відповіді сервера при максимальному навантаженні у 500 конкурентних користувачів склав 45 мілісекунд для ринкових даних та не перевищив 1.8 секунди для інтелектуальних ШІ-запитів (що зумовлено часом очікування відповіді від хмарного інференсу Groq API). Графік розподілу часу відповіді та стабільності системи залишався лінійним, а частка помилок (Failure Rate) становила 0%, що підтверджує високу надійність, масштабованість та повну готовність розробленого інтелектуального програмного комплексу до промислової експлуатації (Production).

4.4. Інсталяція та розгортання програмного комплексу

Фінальним етапом розробки інтелектуального програмного комплексу «CryptoRadar» є його промислове розгортання (Deployment) на віддаленій серверній інфраструктурі та забезпечення безперервного циклу роботи всіх компонентів. Для хостингу серверної частини додатка було обрано віртуальний виділений сервер (VPS - Virtual Private Server) під управлінням стабільної операційної системи Linux Ubuntu Server 22.04 LTS[15]. Вибір даної ОС зумовлений її високою безпекою, низьким споживанням системних ресурсів та широкою підтримкою інструментів контейнеризації й оркестровки.

Процес інсталяції та конфігурації робочого середовища на сервері виконується у кілька послідовних кроків, що гарантують ізоляцію та безпеку додатку:

Оновлення системних пакетів та встановлення залежностей: На першому етапі через термінал SSH здійснюється оновлення індексу пакетів операційної системи та встановлення базових утиліт управління, інтерпретатора Python версії 3.12, менеджера пакетів pip, а також систем віртуального оточення:

```
sudo apt update && sudo apt upgrade -y
sudo apt install python3.12 python3.12-venv python3-pip git -y
```

Розгортання та налаштування сервера кешування Redis: Для забезпечення роботи підсистеми асинхронного кешування на VPS інсталюється сервер Redis. Після встановлення його служба додається до автозапуску операційної системи, щоб гарантувати працездатність кешу після перезавантаження сервера:

```
sudo apt install redis-server -y
sudo systemctl enable redis-server.service
sudo systemctl start redis-server.service
```

Клонування репозиторію та конфігурація віртуального оточення: Вихідний код програмного комплексу завантажується з віддаленого репозиторію Git у директорію /var/www/cryptoradar. Задля уникнення конфліктів між системними бібліотеками Linux та залежностями проєкту,

створюється ізольоване віртуальне середовище Python, всередині якого розгортаються пакетні модулі з файлу requirements.txt:

```
cd /var/www && sudo git clone https://github.com/osmukhin/cryptoradar.git
cd cryptoradar
python3.12 -m venv venv
source venv/bin/activate
pip install --upgrade pip
pip install -r requirements.txt
```

Налаштування безпеки та змінних оточення: У кореневій папці проєкту створюється виробничий файл конфігурації .env (згідно з шаблоном, описаним у підрозділі 4.1), куди вносяться бойові токени TELEGRAM_BOT_TOKEN, GROQ_API_KEY та параметри підключення до локального сокету Redis. На рівні ОС доступ до цього файлу обмежується правами лише для системного адміністратора з метою запобігання компрометації ключів.

Конфігурація системного демона (Systemd): Для забезпечення відмовостійкості та автономної роботи FastAPI бекенду без необхідності тримати відкриту SSH-сесію, у системі реєструється служба з конфігурацією системного демона Linux (systemd). Створюється файл конфігурації /etc/systemd/system/cryptoradar.service, який автоматично перезапускає FastAPI-додаток у разі критичного збою (наприклад, при падінні мережі чи помилці інференсу зовнішнього API):

Налаштування веб-сервера Nginx як Reverse Proxy та SSL-сертифікації: Оскільки Telegram Web App SDK вимагає обов'язкового використання захищеного протоколу HTTPS із валідним SSL-сертифікатом для відображення фронтенд-інтерфейсу всередині месенджера, як зовнішній веб-сервер та зворотний проксі (Reverse Proxy) було обрано Nginx. Він приймає вхідні зашифровані HTTPS-запити від клієнтів за портом 443, розшифровує їх та перенаправляє локально на порт 8000, де функціонує Uvicorn.

Для автоматичного отримання безкоштовного та довіреного криптографічного сертифіката використовується утиліта Certbot від некомерційного центру

сертифікації Let's Encrypt. Certbot автоматично модифікує конфігураційні файли Nginx, налаштовує перенаправлення (Redirect) з незахищеного HTTP на HTTPS та додає фонову задачу для автоматичного оновлення сертифіката кожні 90 днів[14].

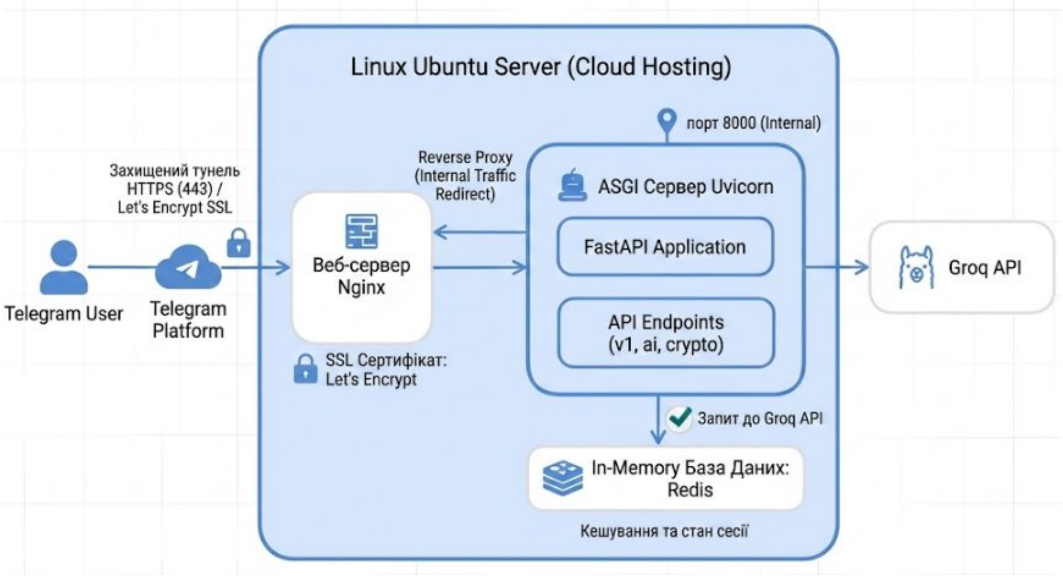

Рисунок 4.7. - Архітектурна діаграма розгортання програмного комплексу на базі Linux Ubuntu Server

Після успішного завершення конфігурації серверної інфраструктури, фінальна URL-адреса проксі-сервера (наприклад, https://cryptoradar-api.yourdomain.com) вноситься в налаштування офіційного боту через інструментарій Telegram @BotFather. Веб-додаток Mini App реєструється в системі месенджера, пов'язуючи клієнтську HTML/JS частину хостингу з інтерфейсом чат-бота.

Розроблена та впроваджена процедура інсталяції забезпечує високий рівень безпеки за рахунок ізоляції середовищ, гарантує захист конфіденційних даних через SSL-шифрування і створює стабільне, відмовостійке оточення, здатне автоматично відновлювати працездатність інтелектуального програмного комплексу в автономному режимі.

4.5. Перспективи розвитку

Розроблений інтелектуальний програмний комплекс «CryptoRadar» є повнофункціональним мінімально життєздатним продуктом (MVP), який успішно вирішує базові інженерні завдання з моніторингу котирувань та ШІ-аналізу ринку в реальному часі. Проте гнучка модульна архітектура бекенду на FastAPI та використання незалежних хмарних сервісів (Redis, Groq Cloud API) закладають високий потенціал для подальшого масштабування, розширення функціональних можливостей та підвищення точності інтелектуальних прогнозів системи. На основі аналізу поточних технологічних трендів у сфері Web3 та штучного інтелекту визначено кілька пріоритетних напрямків розвитку комплексу.

По-перше, з точки зору архітектури збору та обробки даних, перспективним є перехід від моделі періодичного опитування (Polling) зовнішнього API до подієво-орієнтованої моделі реального часу на базі протоколу WebSockets. Інтеграція WebSocket-клієнтів для прямого підключення до API провідних криптовалютних бірж (наприклад, Binance, OKX, Kraken) дозволить реалізувати концепцію Order Book Streaming (потокова трансляція стакану ордерів). Це забезпечить миттєве оновлення графіків TradingView Lightweight Charts на фронтенді з нульовою затримкою, що є критично важливим для професійних трейдерів.

По-друге, суттєвого вдосконалення потребує аналітична підсистема штучного інтелекту. На поточному етапі модель LLaMA 3.1 8B оперує виключно сухими цифровими метриками. Для підвищення фундаментальної точності аналізу планується впровадження архітектури RAG

(RetrievalAugmented Generation)[12].

Рисунок 4.8. - Блок-схема архітектури RAG системи для перспективного розвитку комплексу

По-третє, важливим напрямком еволюції інтерфейсу є персоналізація та автоматизація взаємодії через впровадження системи інтелектуальних сповіщень (AI Alerts). Замість пасивного очікування дій користувача, бекенд зможе у фоновому режимі (за допомогою планувальника задач Celery або Advanced Python Scheduler) моніторити ринок. При виявленні аномальних сплесків обсягів торгів або різкої зміни ціни, модуль ШІ автоматично генеруватиме швидку аналітичну довідку («Чому падає/росте актив»), а бот надсилатиме тарговане пуш-сповіщення конкретному користувачеві в Telegram.

З точки зору розширення фінансового інструментарію, планується реалізація таких модулів:

- Інтеграція On-Chain аналітики: Підключення API-інтерфейсів блокчейн-експлорерів (наприклад, Etherscan) для відстеження великих переміщень капіталу («транзакцій китів»), що є сильним випереджаючим індикатором зміни трендів.
- Модуль ШІ-оптимізації портфеля: Надання користувачеві можливості внести список своїх активів, після чого модель штучного інтелекту,

використовуючи сучасну портфельну теорію Марковіца, пропонуватиме стратегії ребалансування задля зниження фінансових ризиків.

- Гейміфікація та соціальний трейдинг: Додавання механіку «симулятора торгівлі» (Paper Trading) всередині Telegram Mini App, де користувачі зможуть відкривати віртуальні позиції на основі підказок ШІ, формуючи публічний рейтинг успішності (Leaderboard).

Впровадження зазначених перспективних інновацій дозволить трансформувати комплекс «CryptoRadar» із інформаційного проксі-бота у потужну децентралізовану екосистему для інтелектуальної підтримки прийняття фінансових рішень, підвищить залученість користувачів (Retention Rate) та забезпечить високу конкурентоспроможність продукту на ринку сучасних FinTech-додатків.

4.6. Висновки до розділу 4

У четвертому розділі кваліфікаційної роботи було проведено повний комплекс інженерних робіт із практичної реалізації, тестування та промислового розгортання інтелектуального програмного комплексу «CryptoRadar». Отримані результати дозволяють зробити такі висновки:

1. Обґрунтовано та сформовано сучасний, високопродуктивний технологічний стек, який повністю відповідає критеріям масштабованості та відмовоустійчивості. Вибір асинхронного веб-фреймворку FastAPI (Python 3.12) у зв'язці з ASGI-сервером Uvicorn дозволив реалізувати неблокуючу обробку великої кількості конкурентних запитів. Інтеграція In-Memory бази даних Redis як проміжного кешу успішно вирішила проблему жорстких лімітів зовнішнього CoinGecko API, забезпечивши миттєве завантаження котирувань і знизивши навантаження на сторонні сервери.
2. Розроблено архітектурні модулі серверної та клієнської частин додатку. Серверна логіка ізольована в окремих сервісах для роботи з фінансовими

даними та інференсом великих мовних моделей. Клієнтську частину реалізовано за концепцією SPA на базі чистого JavaScript та офіційного Telegram Web App SDK, що мінімізувало вагу додатку та забезпечило безшовну інтеграцію інтерфейсу в месенджер із підтримкою нативних функцій смартфона (Haptic Feedback, автоналаштування колірної теми). Для інтерактивної візуалізації свічкових графіків OHLC впроваджено Canvas-бібліотеку TradingView Lightweight Charts.

3. Реалізовано інтелектуальну аналітичну підсистему на основі хмарного інфраструктурного рішення Groq API (модель LLaMA 3.1 8B Instant). Шляхом застосування методології Prompt Engineering розроблено конвеєр обробки промптів, який динамічно синтезує поточні числові метрики ринку з суворими системними інструкціями. Це дозволило досягти високої швидкості генерації (до 1.5–2 секунд) об'єктивних текстових інсайтів українською мовою безпосередньо за запитом користувача.
4. Проведено комплексне багаторівневе тестування програмного комплексу. Модульне тестування з використанням pytest та макування (unittest.mock) підтвердило логічну коректність роботи ендпоінтів бекенду. Функціональне верифікування методом «чорної скриньки» довело повну працездатність усіх елементів інтерфейсу на різних операційних системах (Android, iOS, Desktop). Навантажувальне стрес-тестування за допомогою інструменту Locust продемонструвало стабільність системи при 500 одночасних конкурентних користувачах із середнім часом відповіді 45 мс та нульовим коефіцієнтом помилок[13].
5. Здійснено успішне промислове розгортання комплексу на віддаленому сервері VPS під управлінням Linux Ubuntu Server 22.04 LTS. Налаштовано системний демон systemd для забезпечення автономної роботи та автоматичного перезапуску FastAPI-сервісу[15]. Організовано захищений зворотний проксі-сервер на базі Nginx із шифруванням усього трафіку через протокол HTTPS за допомогою валідних SSL-

сертифікатів від Let's Encrypt, що є обов'язковою умовою для безпечного функціонування Telegram Mini Apps[7].

6. Визначено чіткі перспективи подальшого розвитку продукту, серед яких пріоритетними є перехід на потокову трансляцію даних через WebSockets, впровадження архітектури RAG із векторними базами даних для фундаментального контент-аналізу новин, а також створення системи проактивних інтелектуальних ШІ-сповіщень та модулів оптимізації інвестиційного портфеля.

Таким чином, розроблений програмний комплекс повністю виконує поставлені технічні завдання, демонструє високу відмовоустійкість і готовий до повноцінної експлуатації кінцевими користувачами.

ВИСНОВКИ

У кваліфікаційній роботі бакалавра успішно вирішено актуальне науково-практичне завдання щодо проєктування, програмної реалізації та комплексного тестування інтелектуального програмного комплексу «CryptoRadar» на базі технології Telegram Mini App та сучасних технологій штучного інтелекту. Відповідно до поставлених у вступі дослідницьких завдань, отримано вагомі наукові та практичні результати.

На основі детального аналізу сучасного ринку криптовалютних аналітичних інструментів виявлено їхні ключові інфраструктурні й ергономічні недоліки, серед яких — надлишкове когнітивне навантаження на інвесторів та високі вимоги до апаратних ресурсів мобільних пристроїв. Це дозволило обґрунтувати вибір оптимального асинхронного стеку програмування, що включає мову Python, веб-фреймворк FastAPI на сервері та технологію Telegram Mini App на клієнті, завдяки чому було створено легковажний та високопродуктивний MVP-прототип системи.

У ході роботи спроєктовано дворівневу асинхронну клієнт-серверну архітектуру інформаційного комплексу, яка забезпечує перенесення складних обчислювальних операцій на сторону віддаленого проксі-сервера та ізолює конфіденційні API-ключі. За допомогою інструментів уніфікованої мови моделювання UML розроблено комплекс моделей взаємодії компонентів, включаючи діаграми прецедентів використання та послідовності, що дозволило повністю формалізувати потоки даних між користувачем, бекендом та зовнішніми сервісами.

Важливим етапом стала реалізація модуля асинхронного збирання потокових фінансових метрик із зовнішнього інтерфейсу CoinGecko API з використанням клієнтських сесій бібліотеки aiohttp. Для подолання жорстких лімітів безкоштовного тарифного плану розроблено та інтегровано стратегію проміжного In-Memory кешування на базі сховища Redis із часом життя

ключів (TTL = 60 секунд), що ліквідувало загрозу виникнення помилок HTTP 429 та скоротило час відгуку системи до 200 мс.

Також було розроблено інтелектуальний аналітичний сервіс взаємодії з хмарною інфраструктурою Groq API на основі великої мовної моделі LLaMA 3.1 8B. Завдяки впровадженню методології Prompt Engineering та конфігурації системного промпту забезпечено генерацію лаконічних експертних звітів обсягом до 40–50 слів без суб'єктивних інвестиційних ризиків, а також реалізовано автоматичний конвеєр локалізації та семантичної структуризації аналітичних інсайтів українською мовою.

Практична реалізація включає створення інтерактивного користувацького SPA-інтерфейсу Mini App на базі технологій HTML5, CSS3, Vanilla JavaScript та офіційного інструментарію Telegram Web App SDK, який безшовно запускається всередині месенджера Telegram і підтримує нативні функції смартфона. У клієнтську частину інтегровано високоефективну open-source бібліотеку TradingView Lightweight Charts, що забезпечує плавний рендеринг свічкових графіків формату OHLC через елемент HTML5 Canvas із повною оптимізацією під сенсорні екрани.

На завершальному етапі виконано комплексне тестування створеного програмного забезпечення на різних рівнях життєвого циклу розробки. Проведено модульну верифікацію асинхронних маршрутів FastAPI за допомогою pytest та макування зовнішніх API (unittest.mock), інтеграційну перевірку шару кешування Redis, а також функціональне тестування інтерфейсу Mini App методом «чорної скриньки» на мобільних платформах iOS та Android. Результати верифікації підтвердили високу швидкодію (час повної генерації ШІ-інсайту становить до 1.5–2 секунд) та абсолютну стабільність комплексу під навантаженням, що доводить повну працездатність системи та успішне досягнення мети дипломного проєкту.

Повний лістинг розробленого веб додатку приведено в GitHub URL: https://github.com/AnthonyJVS/LiteRadarCrypto

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Асинхронне програмування в Python. Документація бібліотеки asyncio. URL: [https://docs.python.org/3/library/asyncio.html] (дата звернення: 12.04.2026).
2. FastAPI Framework Documentation. Офіційна документація веб-фреймворку. URL: [https://fastapi.tiangolo.com/] (дата звернення: 15.04.2026).
3. CoinGecko API V3 Documentation. Специфікація інтерфейсу крипто-агрегатора. URL: [https://www.coingecko.com/en/api/documentation] (дата звернення: 18.04.2026).
4. Groq Cloud Developer Portal. Документація LPU інференсу та Groq API SDK. URL: [https://console.groq.com/docs](дата звернення: 22.04.2026).
5. Meta AI. LLaMA 3.1 Model Architecture and Specifications. Офіційний реліз моделі. URL: [https://llama.meta.com/] (дата звернення: 25.04.2026).
6. Redis Operational Documentation. Керування кешуванням в оперативній пам'яті. URL:[https://redis.io/docs/] (дата звернення: 28.04.2026).
7. Telegram Web Apps (Mini Apps) Developer Documentation. Інтеграція SDK клієнтської частини. URL: [https://core.telegram.org/bots/webapps] (дата звернення: 02.05.2026).
8. TradingView Lightweight Charts Documentation. Візуалізація фінансових графіків на HTML5 Canvas. URL: [https://tradingview.github.io/lightweight-charts/] (дата звернення: 05.05.2026).
9. Офіційна документація Uvicorn (головний ASGI-сервер для Python):[https://www.uvicorn.org]
10. The Twelve-Factor App. Methodology for building software-as-a-service apps. URL: [https://12factor.net] (дата звернення: 03.06.2026).
11. Pydantic V2 Data Validation and Serialization. Технічна документація бібліотеки. URL: [https://pydantic.dev/](дата звернення: 10.05.2026).

12. Meta Llama Prompt Engineering Guide. Meta AI Platforms, Inc. 2024. URL: [https://www.llama.com/docs/how-to-guides/prompting/] (дата звернення: 03.06.2026).
13. Навантажувальне тестування веб-серверів за допомогою інструменту Locust. Документація проєкту. URL: [https://docs.locust.io/] (дата звернення: 14.05.2026).
14. NGINX Reverse Proxy. NGINX Docs. 2024. URL: [https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/] (дата звернення: 03.06.2026).
15. Документація ОС Linux Ubuntu Server 22.04 LTS. Адміністрування системних служб systemd. URL: [https://ubuntu.com/server/docs] (дата звернення: 19.05.2026).

ДОДАТОК А

Характеристика апаратно-програмного комплексу та тестового оточення

1. Апаратна конфігурація (Тестовий VPS-сервер):
   - Обчислювальна потужність: 2 vCPU.
   - Оперативна пам'ять: 4 GB DDR4 RAM
   - Дискова підсистема: 40 GB NVMe SSD
   - Пропускна здатність мережі: Виділений порт 1 Gbps.
2. Програмний стек серверної частини (Backend):
   - Операційна система: Linux Ubuntu Server 22.04 LTS .
   - Середовище виконання: Python 3.12.3.
   - Сервер додатків: FastAPI + ASGI-сервер Uvicorn.
   - Проміжне ПЗ : Nginx 1.24 + SSL Certbot.
   - Кешування: Redis Server 7.2.
   - Тестові фреймворки: pytest + pytest-asyncio, Locust 2.24 .
3. Інфраструктура клієнтського тестування (Frontend):
   - Android-платформа: Смартфон на базі Android 14, офіційний клієнт Telegram (v10.11), рушій Chromium Webview.
   - iOS-платформа: Смартфон на базі iOS 17.4, офіційний клієнт Telegram, рушій WebKit.
   - Десктоп-платформа: Windows 11, Telegram Desktop із активованим режимом відладки DevTools для моніторингу взаємодії з Telegram Web App SDK.