МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія (кафедра) комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

ТЕЛЕГРАМ БОТ З ІНТЕГРАЦІЄЮ ЗОВНІШНІХ АРІ ДЛЯ АВТОМАТИЗАЦІЇ ЗБОРУ ТА АНАЛІЗУ ДАНИХ ІЗ ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

Виконав: студент групи 2П-21
спеціальності
121 Інженерія програмного забезпечення
Денис КЛАДКО
Керівник
Вікторія НЕМЧЕНКО

Черкаси 2025

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

Кафедра комп’ютерної інженерії та інформаційних технологій

Спеціальність 121 "Інженерія програмного забезпечення"

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри КІ та ІТ

_____ Владислав ХОТУНОВ
(підпис)
«_____»_____2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Кладку Денису Сергійовичу

1. Тема кваліфікаційної роботи
Телеграм бот з інтеграцією зовнішніх API для автоматизації збору та аналізу даних із використанням штучного інтелекту

Керівник роботи . Немченко Вікторію Юріївну, викладач

затверджені наказом закладу вищої освіти від «07» жовтня 2024 року № 68у.
2. Строк подання студентом кваліфікаційної роботи 02.06.2025
3. Вихідні дані до кваліфікаційної роботи програмування мовою Python, використання API для отримання фінансових даних, використання API окремих моделей ШІ, методи побудови чатботу у Telegram.
4. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити) огляд сервісів зі штучним інтелектом для збору та аналізу даних, аналіз API моделей штучного інтелекту, використання чатботів у месенджерах, використання API для отримання фінансових даних, використання API для підключення ШІ до програми, розробка чатботу у месенджері Telegram.
5. Дата видачі завдання 16.09.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання з підписами керівника і студента
1	Вступ	14.10.2024	
2	Розділ 1 (Огляд поточного стану предметної області)	09.12.2024	
3	Розділ 2 (Проєктування та реалізація чатботу)	10.03.2025	
4	Розділ 3 (Тестування та супроводження чатботу)	28.04.2025	
5	Висновки	12.05.2025	
6	Оформлення кваліфікаційної роботи (чистовий варіант)	26.05.2025	
7	Перевірка кваліфікаційної роботи на наявність ознак плагіату (за 10 днів до захисту)	02.06.2025	
8	Подання кваліфікаційної роботи на затвердження завідувачу кафедри (за 7 днів до захисту)	10.06.2025	

Студент _____ Денис КЛАДКО

(підпис)

Керівник роботи _____ Вікторія НЕМЧЕНКО

(підпис)

АНОТАЦІЯ

Кваліфікаційна робота на тему “Телеграм бот з інтеграцією зовнішніх API для автоматизованого збору та аналізу даних з використанням технологій штучного інтелекту” включає в себе вступ, основну частину, яка складається з трьох розділів, висновків та списку використаних джерел. Загальний обсяг роботи становить 66 сторінок, в якій міститься 5 рисунків та 3 таблиці. Список використаних джерел налічує 33 позиції.

У рамках цієї роботи було створено інтелектуальний Telegram-бот для автоматизованого фундаментального аналізу криптовалютних проектів та токенів. Розроблена система забезпечує комплексну оцінку інвестиційної привабливості криптопроектів на основі об’єктивних даних та глибокого аналізу штучного інтелекту, демократизуючи доступ до професійного криптоаналізу.

Розроблене рішення базується на Python з використанням бібліотек telebot, requests, matplotlib та інтеграції з кількома зовнішніми API: CoinGecko для отримання базових даних про токени, CoinMarketCap для верифікації інформації та аудитів безпеки, GitHub API для аналізу активності розробників, та OpenAI GPT-3.5-turbo для проведення структурованого аналізу проектів за вісьмома ключовими критеріями. У роботі детально викладено етапи проєктування архітектури системи, логіки обробки даних, реалізації алгоритмів класифікації проектів за ринковою капіталізацією та створення візуалізацій токеноміки.

Система забезпечує автоматичну генерацію комплексних аналітичних звітів з оцінками від 8 до 40 балів, включаючи ринкові дані, технічний аналіз, оцінку ризиків та інвестиційні рекомендації. Додатково розглядаються аспекти тестування програмного забезпечення з розробкою комплексного тестового плану, що включає unit, інтеграційні, продуктивності та безпекові тести.

Запропонована система демонструє потенціал використання штучного інтелекту для автоматизації складних аналітичних процесів у фінансовій сфері та створює можливості для подальшого розширення функціональності.

Ключові слова: чатбот, Telegram, штучний інтелект, API, криптовалюти, автоматизація, фундаментальний аналіз, машинне навчання, OpenAI, Python.

ABSTRACT

The qualification work on the topic "Telegram bot with integration of external APIs for automated data collection and analysis using artificial intelligence technologies" includes an introduction, a main part consisting of three sections, conclusions, and a list of references. The total volume of the work is 66 pages, containing 5 figures and 3 tables. The list of references includes 33 items.

Within the framework of this work, an intelligent Telegram bot was created for automated fundamental analysis of cryptocurrency projects and tokens. The developed system provides comprehensive assessment of investment attractiveness of crypto projects based on objective data and deep artificial intelligence analysis, democratizing access to professional crypto analysis.

The developed solution is based on Python using telebot, requests, matplotlib libraries and integration with several external APIs: CoinGecko for obtaining basic token data, CoinMarketCap for information verification and security audits, GitHub API for analyzing developer activity, and OpenAI GPT-3.5-turbo for conducting structured project analysis according to eight key criteria. The work details the stages of system architecture design, data processing logic, implementation of project classification algorithms by market capitalization, and creation of tokenomics visualizations.

The system provides automatic generation of comprehensive analytical reports with scores from 8 to 40 points, including market data, technical analysis, risk assessment, and investment recommendations. Additionally, software testing aspects are considered with the development of a comprehensive test plan that includes unit, integration, performance, and security tests.

The proposed system demonstrates the potential of using artificial intelligence to automate complex analytical processes in the financial sector and creates opportunities for further functionality expansion.

Keywords: chatbot, Telegram, artificial intelligence, API, cryptocurrencies, automation, fundamental analysis, machine learning, OpenAI, Python.

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface, Інтерфейс програмування застосунків

CRM – Customer Relationship Management, Управління відносинами з клієнтами

ETF – Exchange-Traded Fund, Біржовий інвестиційний фонд

GPT – Generative Pre-trained Transformer, Генеративний попередньо навчений трансформер

IBM – International Business Machines Corporation

IoT – Internet of Things, Інтернет речей

LLM – Large Language Model, Велика мовна модель

LSTM – Long Short-Term Memory, Довга короткочасна пам’ять

MIT – Massachusetts Institute of Technology, Массачусетський технологічний інститут

NLP – Natural Language Processing, Обробка природної мови

RNN – Recurrent Neural Network, Рекурентна нейронна мережа

S&P 500 – Standard & Poor’s 500, Індекс

ВВП – Валовий внутрішній продукт, Gross Domestic Product (GDP)

ШІ – Штучний інтелект, Artificial Intelligence (AI)

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі, що характеризується стрімким розвитком інформаційних технологій та цифровізацією практично всіх сфер життя, збір та аналіз даних набуває безпрецедентної актуальності.

Прийняття обґрунтованих рішень сьогодні неможливе без опори на якісні дані. В умовах високої конкуренції та швидких змін ринкового середовища, організації й окремі користувачі потребують точної та своєчасної інформації для стратегічного планування та щоденної операційної діяльності. Керівники, які спираються лише на інтуїцію та досвід, ризикують прийняти помилкові рішення, що можуть коштувати компанії значних втрат. Натомість, ті, хто вміло використовує аналітику даних, отримують чітке розуміння ситуації та можливість прогнозувати наслідки своїх дій.

Цифрова трансформація, яку переживають підприємства усіх галузей, супроводжується створенням величезних масивів інформації, що потребують правильної обробки та інтерпретації. Кожна транзакція, кожен клік, кожен запит – все це генерує дані, які потенційно містять цінні інсайти. Лише ті організації, які налагодили ефективну систему збору, зберігання та аналізу цих даних, зможуть повноцінно скористатися перевагами цифрової економіки.

Персоналізація продуктів і послуг стала очікуванням сучасних споживачів, а не просто конкурентною перевагою. Глибокий аналіз даних дозволяє компаніям краще розуміти потреби, бажання та поведінку своїх клієнтів, що створює можливості для впровадження справді індивідуального підходу. Це, своєю чергою, підвищує задоволеність клієнтів, зміцнює їхню лояльність та, як наслідок, збільшує продажі і прибутки.

Сучасні підприємства постійно шукають шляхи оптимізації своїх процесів, і саме аналіз даних надає для цього необхідні інструменти. Детальна інформація про виробничі та бізнес-процеси допомагає виявляти вузькі місця, неефективні операції та надлишкові витрати. Більше того, аналітичні моделі дозволяють моделювати різні сценарії оптимізації та обирати найбільш ефективні з них.

У світі, де зміни відбуваються з неймовірною швидкістю, здатність прогнозувати майбутні тенденції стає надзвичайно цінною. Аналітика даних, особливо з використанням передових алгоритмів машинного навчання, дозволяє виявляти приховані закономірності та тренди, передбачати їх розвиток та готуватися до них завчасно. Це дає організаціям стратегічну перевагу та можливість бути на крок попереду конкурентів.

Управління ризиками в сучасних умовах також неможливе без якісного аналізу даних. Своєчасне виявлення потенційних загроз, оцінка їх ймовірності та можливих наслідків, а також розробка запобіжних заходів – все це базується на ретельному аналізі наявної інформації. Компанії, які вміло керують ризиками на основі даних, демонструють більшу стійкість у кризових ситуаціях та зберігають стабільність у довгостроковій перспективі.

Не можна також забувати про роль даних у наукових дослідженнях. Сучасна наука все більше базується на аналізі великих обсягів емпіричної інформації, яка дозволяє виявляти закономірності, перевіряти гіпотези та робити відкриття у різних галузях – від медицини та фізики до соціології та економіки. Розвиток методів аналізу даних відкриває нові горизонти для наукового пізнання та технологічного прогресу.

Особливого значення в контексті збору та аналізу даних набуває автоматизація цих процесів з використанням технологій штучного інтелекту. Сучасні системи штучного інтелекту революціонізують підходи до роботи з даними, значно полегшуючи та підвищуючи ефективність цього процесу. Алгоритми машинного навчання здатні автоматично обробляти величезні масиви структурованої та неструктурованої інформації, виявляти в них закономірності та аномалії, які людина могла б пропустити. Це дозволяє не тільки прискорити аналітичні процеси, але й суттєво підвищити їх якість та глибину.

Технології комп’ютерного зору дають можливість автоматизувати збір візуальних даних та їх інтерпретацію, що відкриває нові можливості в таких сферах як моніторинг виробництва, контроль якості, безпека та багато інших.

Системи обробки природної мови дозволяють аналізувати текстові дані – від відгуків клієнтів до наукових публікацій, витягуючи з них цінні інсайти без необхідності ручного опрацювання.

Інтелектуальні системи автоматизації не лише виконують рутинні аналітичні завдання, але й здатні до самонавчання та адаптації, постійно вдосконалюючи свої алгоритми на основі нових даних. Це створює передумови для переходу від реактивної до проактивної аналітики, коли система не просто відповідає на поставлені запитання, але й самостійно виявляє потенційно важливі тенденції та проблеми.

Автоматизація збору та аналізу даних за допомогою штучного інтелекту також значно знижує вартість цих процесів та розширює доступ до аналітичних інструментів для підприємств різного масштабу. Це демократизує використання аналітики даних та дозволяє навіть невеликим організаціям отримувати конкурентні переваги, які раніше були доступні лише великим корпораціям з потужними аналітичними відділами.

В епоху, коли щодня генеруються петабайти інформації, здатність ефективно збирати, обробляти та аналізувати дані за допомогою інтелектуальних автоматизованих систем стає не просто бажаною, а необхідною умовою для виживання та розвитку. Організації, які розуміють цю актуальність та інвестують у розвиток аналітичних компетенцій та відповідні технології, отримують значну конкурентну перевагу та кращі перспективи для зростання в довгостроковій перспективі.

Метою роботи є розробка чатботу у месенджері Telegram, який буде за допомогою API відповідних сервісів отримувати та обробляти фінансові дані.

Протягом виконання роботи необхідно розв’язати наступні ***завдання***:

1) провести огляд сервісів зі штучним інтелектом для збору та аналізу даних;
2) провести аналіз API моделей штучного інтелекту;
3) розглянути використання чатботів у месенджерах;

4) описати реалізацію використання API для отримання фінансових даних;

5) описати використання API для підключення ШІ до програми;

6) розробити чатбот у месенджері Telegram, що використовує API для отримання фінансових даних та API обраної моделі ШІ для аналізу отриманих даних;

7) розробити набір тестів та провести тестування розробленого чатботу.

Об'єктом дослідження є автоматизація збору та аналізу даних у цифрових інформаційних системах.

Предметом дослідження є технології розробки Telegram-ботів з інтеграцією API.

Під час виконання роботи використано наступні ***методи дослідження***: аналіз та синтез (вивчення існуючих рішень, складання власного), порівняльний аналіз (вибір оптимальних сервісів для інтеграції), моделювання (побудова логіки бота, схеми обробки даних), експеримент (створення і тестування прототипу бота).

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків, викладена на 66 сторінках та містить 5 рисунків і 3 таблиці.

РОЗДІЛ 1
ОГЛЯД ПОТОЧНОГО СТАНУ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Сервіси зі штучним інтелектом для збору та аналізу даних

Існує кілька програмних рішень, які використовують ШІ для збору та аналізу даних у різних сферах. Розглянемо деякі з них.

Google Cloud AI – це хмарна платформа, яка надає широкий спектр інструментів для розробки та впровадження моделей машинного навчання [1]. Вона дозволяє збирати дані з різних джерел, таких як сенсори, бази даних та API, а також забезпечує засоби для попередньої обробки та очищення даних. Завдяки інтеграції з іншими сервісами Google, такими як BigQuery, користувачі можуть ефективно аналізувати великі обсяги інформації та створювати прогнози. Наприклад, компанія Mattel використала BigQuery AI для аналізу відгуків про продукт Barbie Dreamhouse, що допомогло покращити розуміння потреб споживачів.

Amazon SageMaker – це повністю керована платформа для розробки, навчання та розгортання моделей машинного навчання [2]. Підтримує інтеграцію з різними джерелами даних і дозволяє автоматизувати процеси підготовки даних, вибору моделі та її оптимізації. SageMaker пропонує вбудовані алгоритми та підтримує популярні фреймворки, такі як TensorFlow та PyTorch. Це дозволяє швидко створювати та впроваджувати моделі для задач, пов’язаних із прогнозуванням, класифікацією та кластеризацією даних.

Azure AI від Microsoft – це комплексна платформа, яка надає інструменти для створення та розгортання рішень на основі штучного інтелекту [3]. Пропонує широкий вибір моделей для різних завдань, включаючи обробку природної мови, комп’ютерний зір та аналіз даних. Azure AI забезпечує гнучкість у виборі моделей та їх налаштуванні, а також інтеграцію з іншими сервісами Azure для масштабування та управління рішеннями. Платформа

також включає засоби для забезпечення безпеки та відповідності, що є критично важливим для корпоративних клієнтів.

IBM Watson – це потужна ШІ-платформа, відома своїми можливостями в обробці природної мови та аналізі неструктурованих даних, таких як текст, аудіо та відео [4]. Використовується у різних галузях, включаючи охорону здоров'я для аналізу медичних карт, фінанси для оцінки ризиків та клієнтську підтримку для створення чатботів. Watson дозволяє компаніям отримувати глибші інсайти з даних та автоматизувати складні процеси, що сприяє підвищенню ефективності та прийняттю обґрунтованих рішень.

KNIME та RapidMiner – це платформи з відкритим кодом, які надають інструменти для аналізу даних із графічним інтерфейсом [5, 6]. Вони дозволяють інтегруватися з різними джерелами даних, такими як бази даних та файли Excel, і пропонують широкий набір операторів для обробки, аналізу та моделювання даних. Користувачі можуть створювати моделі машинного навчання без необхідності програмування, що робить ці платформи доступними для аналітиків та бізнес-користувачів. RapidMiner та KNIME широко використовуються для маркетингового аналізу, класифікації клієнтів та прогнозування відтоку клієнтів.

Комбінація TensorFlow та Apache Kafka використовується для обробки та аналізу потокових даних у реальному часі [7]. Apache Kafka забезпечує збір та передачу великих обсягів даних з високою пропускною здатністю, тоді як TensorFlow використовується для створення та навчання моделей машинного навчання. Це поєднання дозволяє будувати системи, які можуть автоматично навчатися на нових даних та надавати прогнози в реальному часі. Такий підхід застосовується в індустрії IoT, системах безпеки та для виявлення аномалій.

Power BI від Microsoft – це інструмент бізнес-аналітики, який інтегрує можливості штучного інтелекту для аналізу та візуалізації даних [8]. Він дозволяє підключатися до різних джерел даних, створювати інтерактивні дашборди та використовувати AI-візуалізації для кластеризації, прогнозування

та виявлення аномалій. Power BI допомагає організаціям отримувати інсайти з даних та приймати обґрунтовані рішення на основі візуалізованої інформації.

У сфері криптовалют штучний інтелект використовується для виявлення схем відмивання грошей на блокчейні. Дослідники з Elliptic, MIT та IBM розробили AI-модель, яка аналізує патерни транзакцій Bitcoin, пов'язані з відомими злочинними адресами, щоб виявити підозрілу активність [9]. Ця система використовує глибоке навчання (deep learning), яке дозволяє розпізнавати складні шаблони поведінки на блокчейні, що можуть свідчити про шахрайство, торгівлю на темному ринку або відмивання грошей. Подібні моделі можна інтегрувати з потоковими платформами типу Kafka для аналізу в реальному часі. Це відкриває можливість миттєво блокувати або маркувати транзакції як "ризиковані", що особливо важливо для бірж, банків і регуляторів.

У традиційній економіці системи на основі ШІ також знаходять широке застосування. Наприклад, індексні фонди та фінансові установи використовують моделі машинного навчання для прогнозування динаміки індексів, таких як S&P 500, або коливань валютних курсів, наприклад, долара до євро. Такі моделі враховують історичні дані, макроекономічні індикатори (ВВП, інфляція, відсоткові ставки), новинні стрічки (які також аналізуються за допомогою NLP), та навіть настрої інвесторів, які зчитуються з соцмереж і форумів.

Такі компанії, як Bloomberg або Refinitiv (раніше відома як Thomson Reuters) вже давно впроваджують ШІ у свої аналітичні сервіси, надаючи клієнтам прогнози щодо ринку на основі поведінкових моделей. Наприклад, Bloomberg Terminal використовує NLP для автоматичного аналізу корпоративних звітів та новин і подальшого оцінювання можливого впливу на курс акцій чи облігацій [10].

Також популярною практикою стало використання GPT-подібних моделей для створення автоматизованих фінансових аналітиків – програм, які аналізують щоденні звіти, формують короткі зведення для трейдерів та дають рекомендації щодо купівлі чи продажу активів.

Ще один цікавий приклад – це використання TensorFlow для створення системи прогнозування курсу біткоїна з використанням рекурентних нейронних мереж або LSTM-моделей. Ці моделі враховують як історичні ціни, так і зовнішні чинники, включно з твітами від Ілона Маска чи новинами про зміни регуляцій у країнах.

У результаті, штучний інтелект стає ключовим елементом фінансової аналітики та економічного прогнозування, дозволяючи приймати швидкі та обґрунтовані рішення в умовах швидкозмінного ринку.

Одним з напрямків використання штучного інтелекту є фінансові додатки, частина з яких реалізується через різноманітні фінансові платформи.

TradingView – це потужна платформа для трейдерів і аналітиків, яка використовує алгоритми машинного навчання для створення торгових стратегій, індикаторів та прогнозів. Вона дозволяє аналізувати графіки, дані по акціях, валютних парах, індексах і криптовалютах в реальному часі. ШІ тут застосовується у вигляді розширених аналітичних інструментів, таких як: прогнозні індикатори з автоматичним навчанням, сентимент-аналіз публічних ідей та новин, автоматичне визначення технічних паттернів (наприклад, “голова і плечі”, “трикутники” тощо). [11]

Користувачі платформи можуть писати власні скрипти мовою Pine Script і тренувати торгові стратегії з урахуванням історичних даних.

Numerai – це унікальний хедж-фонд, який працює на основі колективного інтелекту. Вони надають анонімізовані фінансові дані (без конкретних назв компаній чи ринків) і пропонують користувачам по всьому світу створювати власні моделі машинного навчання. Найкращі моделі отримують нагороду у вигляді токена $NMR. [12]

Фонд є прикладом децентралізованого використання ШІ в економіці: користувач створює модель, тестує її і відправляє фонду результати; якщо побудована модель добре передбачає реальні події на ринку – користувач отримує дохід.

На сьогодні це один із найбільш передових прикладів поєднання криптовалюти, ШІ та фінансової аналітики.

Robinhood – популярний мобільний додаток для торгівлі акціями, ETF та криптовалютами, який використовує ШІ для персоналізації досвіду користувачів. У додатку ШІ застосовується для надання рекомендацій на основі аналізу поведінки трейдера, індивідуальних сповіщень про ринкові рухи, що можуть бути релевантними, аналізу ризику портфеля та прогнозів на основі трендів.

Додаток також вміє проводити аналіз “мікронастроїв ринку”, орієнтуючись на активність мільйонів користувачів, і реагувати на них через динамічні інтерфейси, наприклад, сповіщення типу “95% Robinhood інвесторів тримають цю акцію”. [13]

CryptoHopper – це платформа-бот для автоматичної торгівлі криптовалютами, яка використовує штучний інтелект для налаштування торгових стратегій. Платформа аналізує ринкові сигнали, технічні індикатори, новини, а також використовує глибоке навчання для покращення ефективності торгівлі. [14]

Також платформа є вдалим прикладом того, як ШІ може допомогти навіть початківцям автоматизувати трейдинг без необхідності розуміти всі тонкощі технічного аналізу.

Kensho – це аналітична платформа, яка використовує AI для прогнозування подій на фінансових ринках. Вона аналізує величезні масиви економічної інформації, новин, аналітичних звітів та подій у реальному часі (наприклад, виступи глав центробанків, зміни відсоткових ставок тощо) і прогнозує їх вплив на фінансові інструменти. [15]

Kensho широко використовується на Уолл-стріт для глибокого аналізу індексів, ETF та глобальних макроекономічних факторів.

У таблиці 1.1 наведено порівняльний аналіз розглянутих платформ, що використовують у своїй функціональності ШІ.

Описані сервіси демонструють, як штучний інтелект стає ключовим інструментом у сучасній економіці – як у великих хедж-фондах, так і для окремих трейдерів.

Таблиця 1.1. Порівняльний аналіз платформ зі ШІ

Платформа	API	Безкоштовний план	Мови / Технології	Примітки
TradingView	Web API (обмежений), Webhooks	обмежений функціонал	Pine Script (власна мова для стратегії), Webhooks	API більше для сповіщень і взаємодії з ботами, ніж для повної інтеграції даних
Numerai	Full API (REST)	всі можуть подати модель	Python, R, Jupyter, TensorFlow, XGBoost, LightGBM	Публічний API для участі в змаганнях. Повна тех. документація доступна на GitHub
Robinhood	Напівофіційний API	немає офіційного безкоштовного API-доступу	Python (через сторонні бібліотеки, наприклад `robin_stocks`)	API існує, але не документований - використовується через сторонні рішення на свій ризик
CryptoHopper	Повнофункціональний API	безкоштовна версія з лімітами	Python, REST API, Webhooks	Підходить для підключення сторонніх ботів, створення власних стратегій
Kensho (S&P)	тільки для корпоративних клієнтів	немає відкритого доступу	Внутрішні сервіси, інтеграція через Python / C++ SDK	API недоступне для загалу – лише великі банки, фінансові установи з контрактом
Bloomberg Terminal	платний API – Bloomberg API / BLPAPI	висока абонплата	Python, C++, Java, Excel VBA	Потужний API, але тільки з підпискою на Bloomberg Terminal ($20–25k/рік приблизно)

Серед розглянутих платформ у випадку, коли необхідно використати функціональність якоїсь з них, можна розглянути наступні варіанти.

У випадку, коли необхідно писати обробник на Python, найзручніше буде під’єднатись до Numerai, CryptoHopper чи через сторонні інтерфейси до Robinhood.

Якщо потрібен безкоштовний доступ з простим API – краще за все обирати Numerai чи CryptoHopper.

Якщо ж користувачу необхідно отримати високоякісні професійні дані і він має бюджет, то Bloomberg та Kensho надають максимально глибокий функціонал, хоча API доступний лише за контрактом.

1.2 Аналіз API моделей штучного інтелекту

У сучасному технологічному ландшафті API моделей штучного інтелекту стали ключовим інструментом для компаній і розробників, які прагнуть впровадити передові можливості ШІ у свої продукти та сервіси. Ринок API штучного інтелекту постійно розвивається, пропонуючи широкий спектр можливостей – від обробки природної мови до генерації зображень, від аналізу даних до прогнозування. Розглянемо найбільш впливові API моделей штучного інтелекту.

OpenAI пропонує один із найвідоміших і найпотужніших наборів API для доступу до своїх моделей. Їхні API включають доступ до сімейства GPT моделей (GPT-3.5, GPT-4), які вважаються одними з найпотужніших у галузі. Особливістю API OpenAI є гнучкість у налаштуваннях параметрів взаємодії з моделями, таких як temperature (для контролю креативності відповідей) та max_tokens (для обмеження довжини генерованого тексту). [24]

Тарифікація OpenAI базується на кількості токенів (як вхідних, так і вихідних), що робить вартість використання передбачуваною, але потенційно високою для завдань, що вимагають обробки великих обсягів тексту. Модель

GPT-4 значно дорожча у використанні порівняно з GPT-3.5, що може стати обмежувальним фактором для деяких ас томн. Документація API OpenAI вважається однією з найбільш вичерпних і доступних для початківців, що полегшує інтеграцію та використання.

Модель GPT-4 демонструє найкращі результати в розумінні контексту, генерації тексту та розв’язанні складних задач, включаючи програмування та логічні міркування. Однак, усі моделі OpenAI страждають від проблеми “галюцинацій” – генерування неправдивої інформації, яку важко відрізнити від правдивої, особливо у спеціалізованих областях.

API Anthropic для доступу до моделей Claude, включаючи Claude 3 Opus, Claude 3.5 Sonnet та Claude 3.7 Sonnet, позиціонується як безпечна альтернатива іншим LLM. Anthropic приділяє особливу увагу етичним аспектам використання ШІ та намагається мінімізувати ризики шкідливого використання своїх моделей. [25]

Структура API Anthropic відрізняється від OpenAI своєю простотою та фокусом на безпеці. Моделі Claude часто демонструють кращі результати в уникненні генерації потенційно шкідливого контенту та дотриманні етичних обмежень. Вартість використання API Claude конкурентна з OpenAI, хоча специфіка тарифікації дещо відрізняється.

Сильною стороною моделей Claude є їхня здатність обробляти довгі контексти (до 100,000 токенів у деяких версіях), що значно перевищує можливості багатьох конкурентів. Це робить Claude особливо корисним для аналізу довгих документів, наукових статей або технічних специфікацій. Крім того, моделі Claude демонструють високу точність у розумінні та дотриманні інструкцій, що важливо для багатьох корпоративних застосувань.

Google відносно недавно вийшов на ринок API для великих мовних моделей зі своїм Gemini API (раніше відомим як PaLM API). Gemini пропонує кілька версій моделей різної потужності, включаючи Gemini Pro та Gemini Ultra, які конкурують з найпотужнішими моделями на ринку. [26]

Особливістю Gemini API є його інтеграція з екосистемою Google Cloud, що робить його привабливим для компаній, які вже використовують інші сервіси Google. Також важливою перевагою є можливість налаштування моделей під конкретні завдання та домени за допомогою тюнінгу та адаптації.

Моделі Gemini демонструють відмінні результати у науковій та математичній сферах, частково завдяки навчанню на великій базі наукових публікацій. Однак, API Gemini не такий гнучкий у налаштуваннях, як OpenAI, що може обмежити його використання для деяких специфічних завдань.

Вартість використання Gemini API конкурентна, особливо для моделей середньої потужності, що робить його привабливим для проектів з обмеженим бюджетом, які все ж потребують високої якості обробки мови.

Cohere API пропонує доступ до спеціалізованих моделей, оптимізованих для конкретних завдань обробки мови, таких як генерація тексту, класифікація та пошук інформації. На відміну від більш універсальних API, Cohere фокусується на забезпеченні високої продуктивності у специфічних бізнес-сценаріях. [27]

Особливістю Cohere API є його акцент на семантичному пошуку та вбудовуваннях (embeddings), які дозволяють ефективно перетворювати текст у числові вектори для подальшого аналізу та пошуку схожих документів. Це робить Cohere особливо цінним для проектів, пов’язаних з аналізом великих обсягів тексту та побудовою систем пошуку.

Моделі Cohere, такі як Command і Command R, демонструють хороші результати в задачах, пов’язаних з розумінням контексту та генерацією релевантних відповідей. Вартість використання Cohere API типово нижча, ніж у найпотужніших моделей OpenAI або Anthropic, що робить його доступним для ширшого кола проектів.

Cohere API також відзначається своєю надійністю та стабільністю, що важливо для комерційних застосувань, де простої можуть мати серйозні наслідки.

Mistral AI – відносно новий гравець на ринку моделей штучного інтелекту – швидко завоював увагу своїми високопродуктивними моделями, такими як Mistral 7B, Mixtral 8x7B та Mistral Large. Ключовою перевагою API Mistral є висока ефективність їхніх моделей відносно розміру і вартості використання. [28]

Моделі Mistral відзначаються своєю архітектурою, яка оптимізована для забезпечення високої продуктивності навіть при відносно невеликій кількості параметрів. Це дозволяє досягти балансу між вартістю обчислень і якістю результатів, що особливо важливо для проектів з обмеженим бюджетом.

API Mistral пропонує гнучкі налаштування для різних завдань, від простої генерації тексту до складного логічного міркування. Тарифікація Mistral AI конкурентна, особливо для їхніх легших моделей, що робить цей API привабливим варіантом для стартапів та малих підприємств.

Особливістю Mistral також є їхня відкритість: компанія випустила кілька моделей з відкритим кодом, що дозволяє розробникам не тільки користуватися їхнім API, але й розгортати моделі на власній інфраструктурі для специфічних потреб.

Згадані вище моделі працюють виключно з текстом. Однак, існує також інший вид моделей, які здатні працювати з різними типами даних (ас томнихи) одночасно – мультимодальні моделі. На відміну від моделей, які спеціалізуються лише на одному типі даних, мультимодальні моделі можуть розуміти та обробляти текст, зображення, аудіо, відео, дані сенсорів.

Головна перевага мультимодальних моделей полягає у здатності інтегрувати інформацію з різних джерел, подібно до того, як люди використовують різні органи чуття для сприйняття світу. Наприклад, мультимодальна модель може аналізувати фотографію і відповідати на запитання про неї текстом, або генерувати зображення на основі текстового опису.

На ринку є в наявності низка мультимодальних моделей API, а саме: OpenAI DALL-E та Vision API, GPT-4 Vision API, Stability AI, Google Gemini Vision API.

Також, додатково до зазначених вище, існують спеціалізовані API. Наведемо кілька їх прикладів.

HuggingFace Inference API – надає доступ до тисяч моделей машинного навчання, включаючи моделі обробки мови, генерації зображень, аудіо та інші. Ключовою перевагою цього API є різноманітність доступних моделей – від легких і спеціалізованих до важких і універсальних. [29]

Amazon Bedrock – це сервіс, який надає єдиний API для доступу до різних фундаментальних моделей, включаючи моделі від Anthropic, AI21 Labs, Cohere, Stability AI та власні моделі Amazon. Ця уніфікація інтерфейсу є ключовою перевагою для підприємств, які хочуть використовувати різні моделі без необхідності управляти множинними інтеграціями. [30]

Microsoft Azure AI представляє комплексний набір API для різних задач штучного інтелекту, включаючи обробку мови, комп'ютерний зір, генерацію контенту та аналіз даних. Ключовою перевагою Azure AI є його інтеграція з іншими продуктами Microsoft, такими як Office 365, Dynamics і Power Platform.

При виборі API моделей штучного інтелекту важливо враховувати кілька технічних аспектів.

Латентність відповідей, або час, необхідний для отримання результату від API, критична для інтерактивних застосувань. У цьому аспекті легші моделі, такі як GPT-3.5 або Mistral 7B, часто перевершують важчі, хоча й потужніші альтернативи.

Підтримка тривалого контексту є важливим фактором для задач, що вимагають аналізу великих документів чи підтримки тривалих діалогів. Моделі Claude від Anthropic лідирують у цьому аспекті, підтримуючи контексти до 100000 токенів, тоді як більшість інших моделей обмежені 8000-32000 токенів.

Надійність і стабільність API є критичними для комерційних застосувань. У цьому аспекті встановлені провайдери, такі як Google, Microsoft і Amazon, часто пропонують кращі гарантії доступності та підтримки порівняно з меншими або новішими гравцями.

Можливості масштабування важливі для проектів, які потенційно можуть швидко зрости за кількістю запитів. API великих хмарних провайдерів, таких як Azure AI та Google Gemini, зазвичай пропонують кращі можливості автоматичного масштабування порівняно з незалежними провайдерами.

Вартість використання API є критичним фактором для багатьох проектів. Моделі відрізняються не тільки абсолютною вартістю, але й структурою тарифікації. Деякі API, такі як OpenAI, тарифікують за кількістю токенів, інші – за часом обчислень чи кількістю запитів.

Для проектів з обмеженим бюджетом легші моделі, такі як Mistral 7B або GPT-3.5, можуть пропонувати кращий баланс вартості та продуктивності порівняно з найпотужнішими, але дорогими альтернативами, такими як GPT-4 або Claude Opus.

Важливо також врахувати приховані витрати, такі як необхідність доопрацювання результатів моделі або додаткової обробки даних. Деякі дешевші API можуть вимагати більше ас томнихи, що збільшує загальну вартість проекту.

Для стартапів і малих підприємств важливими є також умови безкоштовних тарифів і програм для стартапів, які пропонують різні провайдери. Наприклад, OpenAI, Anthropic і Google пропонують спеціальні програми для стартапів, які можуть значно знизити початкові витрати.

Отже, вибір API моделей штучного інтелекту повинен базуватися на комплексному аналізі технічних, економічних, юридичних та етичних аспектів у контексті конкретних потреб проекту. Для задач, які вимагають найвищої якості обробки мови та розуміння контексту, API таких моделей, як GPT-4 від OpenAI або Claude Opus від Anthropic, можуть бути оптимальним вибором, незважаючи на вищу вартість.

Для проектів з обмеженим бюджетом або специфічними вимогами до приватності даних легші моделі, такі як Mistral 7B чи власні розгортання відкритих моделей через такі платформи, як HuggingFace, можуть пропонувати кращий баланс вартості, продуктивності та контролю.

Корпоративні користувачі з високими вимогами до безпеки, відповідності ас томни та інтеграції з існуючими системами можуть віддати перевагу рішенням від великих хмарних провайдерів, таких як Amazon Bedrock, Microsoft Azure AI або Google Gemini, які пропонують комплексні екосистеми ШІ-сервісів з корпоративним рівнем підтримки та масштабованості.

1.3 Використання чатботів для підвищення функціональності месенджерів

Чатбот – це програмне забезпечення, яке імітує спілкування з людиною через текст або голос. Воно створене для того, щоб автоматизувати взаємодію між людьми та комп'ютерними системами. Зазвичай чатботи працюють у популярних месенджерах, на вебсайтах, у мобільних додатках або навіть у голосових помічниках, таких як Siri, Google Assistant чи Alexa. Вони можуть відповідати на запитання користувачів, надавати інформацію, допомагати з вибором товару, оформленням замовлення, бронюванням квитків або навіть виконувати банківські операції.

Історія чатботів розпочалася ще у 1960-х роках. Одним із перших прикладів був бот ELIZA, створений у 1966 році, який імітував спілкування з психотерапевтом, використовуючи прості шаблони для аналізу тексту. З того часу технології значно просунулись уперед. Сьогоднішні чатботи можуть аналізувати мову, вчитися з досвіду та відповідати на складні запити, завдяки штучному інтелекту та машинному навчанню. [16]

Основою сучасних чатботів є кілька важливих технологій. Насамперед, це обробка природної мови (NLP), яка дозволяє ботам розуміти людську мову у текстовій або голосовій формі. Завдяки штучному інтелекту чатботи можуть

самонавчатися, аналізувати попередні діалоги, покращувати якість своїх відповідей і навіть адаптуватися до стилю спілкування конкретного користувача. Також чатботи часто інтегрують із зовнішніми базами даних, платіжними сервісами, CRM-системами чи іншими платформами, що дає їм змогу виконувати практичні завдання, наприклад, оформлювати замовлення або бронювати столик у ресторані.

Використання чатботів охоплює багато сфер. У бізнесі вони допомагають в обслуговуванні клієнтів, скорочуючи час очікування відповіді та полегшуючи роботу операторів. У сфері електронної комерції чатботи рекомендують товари, допомагають із оформленням покупок і збирають зворотний зв'язок. В освіті вони сприяють інтерактивному навчанню, перевірці знань, тренуванню навичок. У фінансових установах чатботи вже давно відповідають на запити щодо залишку на рахунку, переказів або банківських продуктів. Також вони використовуються у сфері охорони здоров'я для запису до лікаря, моніторингу симптомів або роз'яснення медичної інформації.

Попри широкі можливості, чатботи мають і свої обмеження. Найпоширеніші з них – проблеми з розумінням складного контексту або емоційних відтінків у мові. Багато ботів все ще діють за чітко прописаними сценаріями, що знижує ефективність у нестандартних ситуаціях. Крім того, якість їхньої роботи безпосередньо залежить від обсягу та якості даних, на яких вони навчались.

Чатботи стали невід'ємною частиною сучасних месенджерів, забезпечуючи автоматизовану взаємодію між бізнесом та користувачами. Розглянемо детальніше, як вони використовуються у різних платформах.

Telegram відомий своєю відкритою платформою для розробки ботів, що дозволяє створювати різноманітні автоматизовані сервіси. Боти в Telegram можуть надсилати повідомлення, відповідати на команди користувачів, керувати групами та каналами, а також інтегруватися з іншими сервісами. Розробники мають доступ до потужного API, що спрощує процес створення та впровадження ботів. [18]

Facebook Messenger надає можливість ас томн створювати ботів для автоматизації спілкування з клієнтами. Боти можуть відповідати на запитання, надавати інформацію про товари та послуги, приймати замовлення та навіть здійснювати платежі. Інтеграція з Facebook дозволяє використовувати дані профілів користувачів для персоналізації взаємодії. [19]

Viber пропонує API для створення ботів, які можуть надсилати текстові повідомлення, зображення, відео та інший контент. Боти у Viber використовуються для інформування користувачів, проведення маркетингових кампаній та надання підтримки клієнтам. Платформа підтримує різні типи повідомлень, включаючи каруселі та кнопки дій, що робить взаємодію з ботами більш інтерактивною. [20]

WhatsApp Business API дозволяє компаніям інтегрувати ботів для обробки повідомлень від клієнтів, надсилання сповіщень та автоматизації обслуговування. Боти можуть надсилати текстові повідомлення, зображення, документи та інший контент. Інтеграція з CRM-системами та іншими бізнес-інструментами дозволяє ефективно керувати взаємодією з клієнтами. [21]

Slack активно використовується у корпоративному середовищі для внутрішньої комунікації. Боти в Slack допомагають автоматизувати рутинні завдання, надавати інформацію, створювати нагадування та інтегруватися з іншими сервісами, такими як Google Drive або Trello. Розробники можуть використовувати Slack API для створення ас томних ботів, які відповідають специфічним потребам компанії. [22]

Discord популярний серед геймерів та онлайн-спільнот. Боти в Discord використовуються для модерації серверів, відтворення музики, надсилання сповіщень та інтеграції з іншими платформами. API Discord дозволяє розробникам створювати ботів з широким спектром функцій, що сприяє розвитку активних та інтерактивних спільнот. [23]

У таблиці 1.2 подано порівняльну таблицю використання чатботів у згаданих месенджерах.

Таблиця 1.2.

Порівняльний аналіз використання чатботів у деяких месенджерах

Месенджер	Основні функції ботів	Особливості платформи
Telegram	Надсилання повідомлень, керування групами та каналами, інтеграція з іншими сервісами.	Відкритий API, підтримка різних типів контенту, простота розробки та впровадження.
Facebook Messenger	Автоматизація спілкування з клієнтами, надання інформації про товари та послуги, прийом замовлень, здійснення платежів.	Глибока інтеграція з Facebook, використання даних профілів для персоналізації, підтримка різних типів повідомлень.
Viber	Надсилання текстових повідомлень, зображень, відео, проведення маркетингових кампаній, надання підтримки клієнтам.	Підтримка каруселей, кнопок дій, інтеграція з іншими сервісами, можливість перевірки статусу онлайн користувачів.
WhatsApp	Обробка повідомлень від клієнтів, надсилання сповіщень, автоматизація обслуговування, надсилання текстових повідомлень, зображень, документів	Інтеграція з CRM-системами, підтримка різних типів контенту, можливість отримання офіційного статусу (зелена галочка) для бізнес- акаунтів.
Slack	Автоматизація рутинних завдань, надання інформації, створення нагадувань, інтеграція з іншими сервісами (Google Drive, Trello).	Використання в корпоративному середовищі, розширені можливості інтеграції, підтримка кастомних ботів для специфічних потреб компанії.
Discord	Модерація серверів, відтворення музики, надсилання сповіщень, інтеграція з іншими платформами.	Популярність серед геймерів та онлайн-спільнот, потужний API для створення ботів з широким спектром функцій, сприяння розвитку активних та інтерактивних спільнот.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ЧАТБОТУ

2.1 Аналіз вимог до чатботу

Сучасний ринок криптовалют характеризується надзвичайною динамічністю та складністю для аналізу. Станом на 2024 рік на ринку були присутні понад 10000 різних криптопроектів, що створює значні труднощі для інвесторів у виборі перспективних активів. Більшість приватних інвесторів не мають доступу до професійних аналітичних інструментів, які використовуються інституційними гравцями.

Існуючі рішення для криптоаналізу мають ряд суттєвих обмежень. Професійні аналітичні платформи коштують від кількох сотень до тисяч доларів на місяць, що робить їх недоступними для більшості роздрібних інвесторів. Безкоштовні рішення часто надають лише поверхневу інформацію без глибокого фундаментального аналізу. Крім того, більшість існуючих інструментів фокусуються на технічному аналізі цін, ігноруючи фундаментальні аспекти проектів.

Основною цільовою аудиторією розроблюваного проекту є приватні інвестори з різним рівнем досвіду у криптовалютах. Це включає новачків, які потребують структурованої інформації для прийняття перших інвестиційних рішень, досвідчених трейдерів, що шукають додаткові джерела аналітики для верифікації своїх висновків, та інституційних інвесторів малого та середнього розміру, які потребують швидкого скринінгу проектів.

Вторинна аудиторія включає крипто-ентузіастів, що цікавляться технологічними аспектами блокчейн-проектів, фінансових консультантів, які надають поради клієнтам щодо криптоінвестицій, та освітні заклади, що вивчають криптовалюти та блокчейн-технології.

Розроблювана система повинна надавати комплексний аналіз криптовалютних проектів на основі їх назви або символу. Аналіз має включати

оцінку за вісьмома ключовими критеріями з присвоєнням балів від одного до п’яти та формуванням підсумкової оцінки від 8 до 40 балів.

Кожен аналіз повинен містити актуальну ринкову інформацію, включаючи поточну капіталізацію проекту, його позицію в секторі та загальний ранг на ринку. Система має автоматично класифікувати проекти за розміром капіталізації та визначати рівень ризику інвестицій.

Платформа повинна інтегруватися з мінімум трьома зовнішніми джерелами даних для забезпечення повноти та достовірності інформації. Основним джерелом має бути CoinGecko API для отримання базових даних про токени, ринкові показники та описи проектів.

Додатково система повинна використовувати CoinMarketCap API для верифікації даних та отримання інформації про аудити безпеки. Для оцінки активності розробників необхідна інтеграція з GitHub API з аналізом кількості комітів та регулярності оновлень репозиторіїв.

Платформа повинна використовувати штучний інтелект для глибокого аналізу описів проектів та генерації експертних висновків. Аналіз на основі ШІ має оцінювати технологічні переваги проектів, їх бізнес-моделі, ризики та конкурентні позиції.

Система повинна автоматично генерувати візуалізації токеноміки у вигляді діаграм розподілу токенів з детальною розбивкою по категоріям. Всі графіки мають бути створені в високій якості з можливістю збереження та поширення.

Взаємодія з системою повинна відбуватися через Telegram-бота з інтуїтивно зрозумілим командним інтерфейсом. Бот має підтримувати обробку текстових запитів з назвами проектів та надавати детальні звіти у зручному для читання форматі.

Система повинна автоматично розбивати великі звіти на логічні частини з урахуванням обмежень Telegram на довжину повідомлень. Кожен звіт має включати текстову частину з аналізом та графічні матеріали з візуалізацією даних.

Система повинна забезпечувати доступність на рівні 99.5% з урахуванням планових технічних робіт. У випадку недоступності одного з зовнішніх API платформа має продовжувати роботу з обмеженим функціоналом та інформувати користувачів про тимчасові обмеження.

Всі помилки мають бути належним чином оброблені з наданням зрозумілих повідомлень користувачам. Система повинна автоматично відновлюватися після тимчасових збоїв та вести детальні логи для діагностики проблем.

Система не повинна зберігати персональні дані користувачів понад необхідний мінімум для функціонування. Всі тимчасові файли мають автоматично видалятися після завершення обробки запитів.

Основою системи має бути Python версії 3.8 або вище з використанням стандартних бібліотек для HTTP-запитів, обробки JSON та роботи з зображеннями. Для забезпечення стабільності всі зовнішні залежності повинні мати фіксовані версії.

Система повинна успішно аналізувати мінімум 95% проектів, представлених у топ-500 за капіталізацією на CoinGecko. Кожен звіт має містити всі заплановані розділи з коректно розрахованими оцінками та візуалізаціями.

ШІ-аналіз повинен генерувати осмислені та релевантні висновки для всіх основних категорій криптопроектів. Система має коректно обробляти проекти з неповною інформацією та чітко вказувати на відсутні дані.

2.2 Проєктування чатботу

Чатбот, запропонований до розробки, – це інтелектуальний Telegram-бот, що створений для автоматизованого фундаментального аналізу криптовалютних проектів та токенів. Проект покликаний надавати користувачам комплексну оцінку інвестиційної привабливості криптопроектів на основі об’єктивних даних та глибокого аналізу штучного інтелекту.

Основна ідея полягає у демократизації доступу до професійного криптоаналізу. Традиційно такі послуги були доступні лише великим інвестиційним фондам або коштували значні кошти для приватних інвесторів. Розроблюваний бот робитиме якісний фундаментальний аналіз безкоштовним і доступним кожному через популярний месенджер Telegram.

Система побудована за модульним принципом, що забезпечує гнучкість розробки та легкість подальшого розширення функціоналу (рис. 2.1). Основу становлять два ключові компоненти: модуль аналізу та Telegram-інтерфейс, які тісно взаємодіють між собою через чітко визначені API.

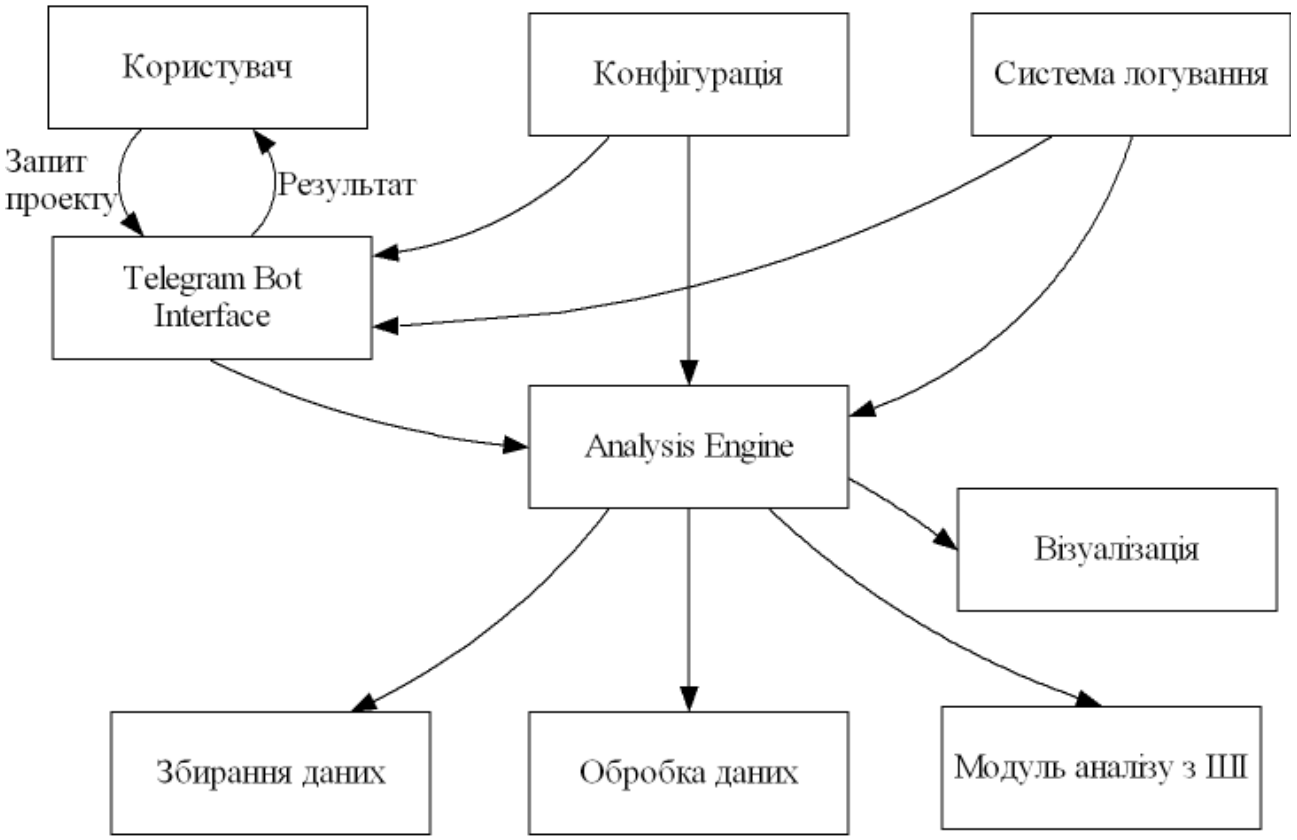

Рисунок 2.1 – Структура розроблюваної системи

Модуль аналізу відповідає за збір, обробку та аналіз даних про крипто проекти (рис. 2.2). Він інтегрується з трьома основними зовнішніми API: CoinGecko для отримання базових даних про токени, CoinMarketCap для верифікації інформації та додаткових даних про аудити, та GitHub API для

аналізу активності розробників. Цей підхід забезпечує максимальну повноту та достовірність інформації.

Рисунок. 2.2 – Модуль збору даних

Telegram-інтерфейс обробляє всі взаємодії з користувачами, від отримання запитів до доставки готових звітів. Він також відповідає за розбиття довгих повідомлень на читабельні частини та управління файлами зображень, що генеруються для візуалізації токеноміки.

Технологічний стек проекту базується на Python як основній мові розробки, що обумовлено її потужними бібліотеками для роботи з API, обробки даних та машинного навчання. Для HTTP-запитів використовується бібліотека Requests, візуалізація даних реалізована через Matplotlib, а математичні обчислення виконуються за допомогою NumPy.

Процес аналізу криптопроекту складається з п’яти послідовних етапів, кожен з яких має свою специфіку та важливість для формування фінального висновку (рис. 2.3).

Перший етап ініціалізації починається з отримання запиту від користувача та пошуку відповідного проекту в базі даних CoinGecko. Система використовує інтелектуальний пошук, який зможе знаходити проекти навіть при неточному введенні назви. Якщо проект не знайдено, користувач отримує відповідне повідомлення з пропозицією перевірити правильність назви.

Другий етап збору даних є найбільш ресурсомістким. Система послідовно звертається до різних API для отримання комплексної інформації про проект (рис. 2.4). З CoinGecko беруться основні дані про токен, його ринкову капіталізацію, обсяги торгів, категорії та детальний опис проекту. Паралельно перевіряється наявність проекту на CoinMarketCap та наявність інформації про аудити безпеки.

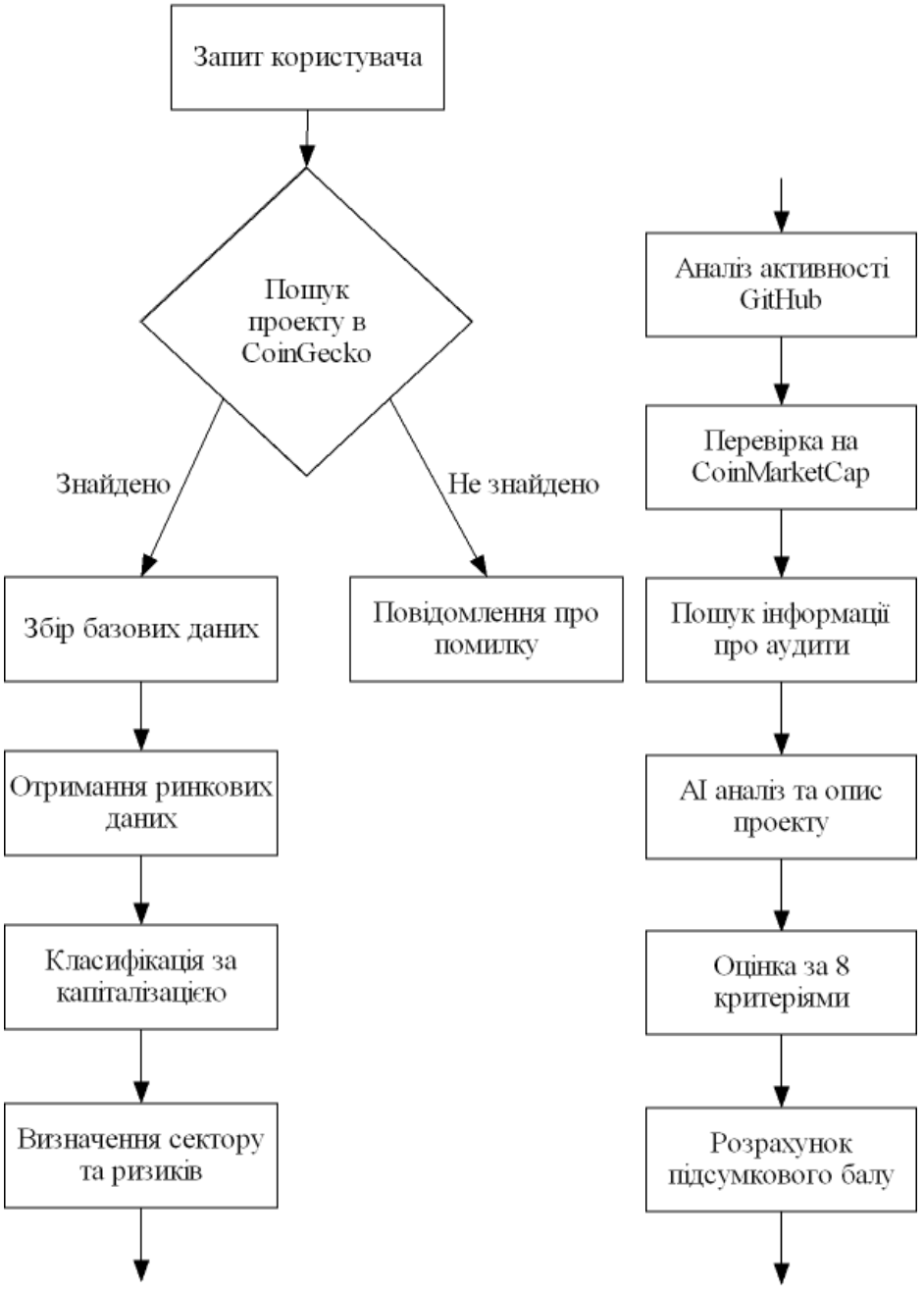

Рисунок 2.3 – Отримання даних про заданий проект, їх перевірка та аналіз проекту з використанням ШІ

Третій етап обробки та аналізу включає очищення отриманих даних від HTML-тегів, валідацію інформації та її структурування для подальшого аналізу. На цьому етапі також відбувається класифікація проекту за ринковою капіталізацією та визначення його позиції в секторі.

Четвертий етап передбачає глибокий аналіз проекту з використання ШІ за вісьмома ключовими критеріями. Система OpenAI аналізує опис проекту, його технічні особливості, бізнес-модель та ринкові перспективи, надаючи кожному критерію оцінку від одного до п’яти балів з детальним обґрунтуванням. [31, 32, 33]

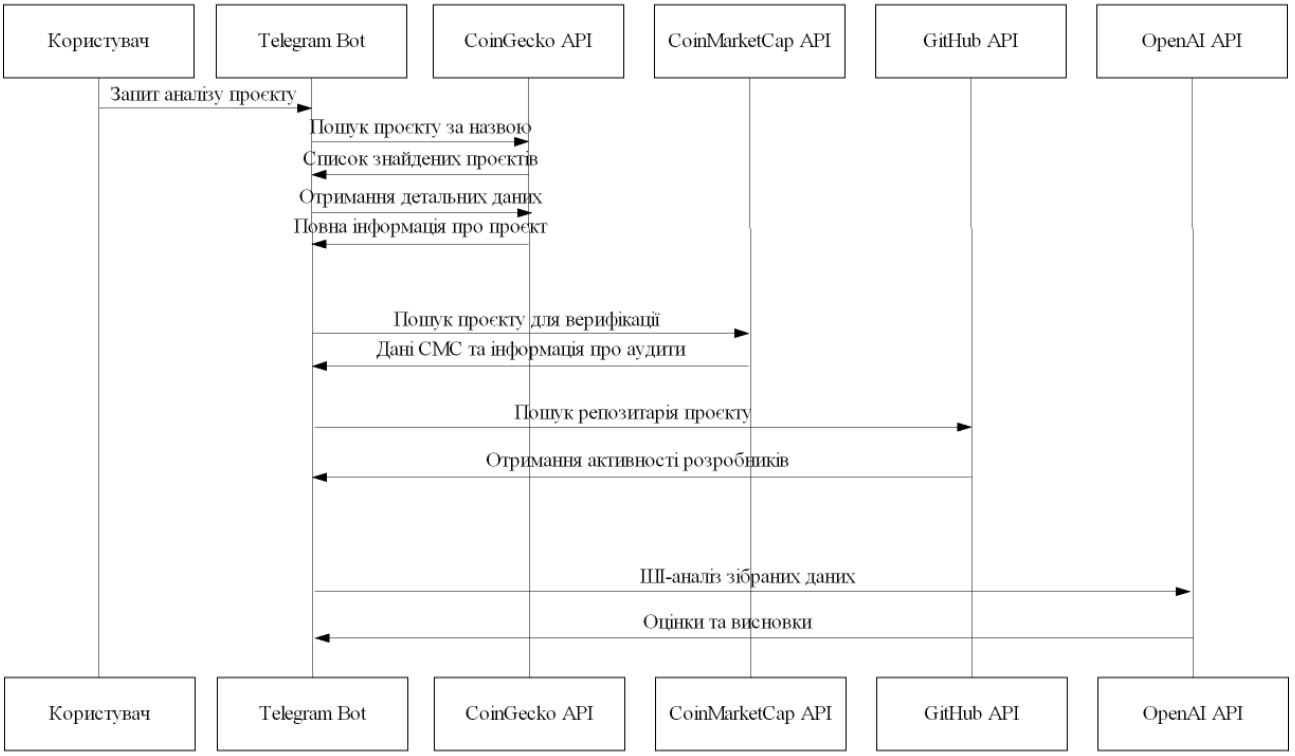

Рисунок 2.4 – Порядок використання різних API для отримання даних та їх аналізу

П’ятий, завершальний, етап (рис. 2.5) включає формування комплексного звіту, створення візуалізацій токеноміки та підготовку підсумкових рекомендацій. Готовий аналіз надсилається користувачу у зручному форматі з автоматичним розбиттям на читабельні частини.

Фундаментальний аналіз базується на восьми ключових критеріях, які всебічно характеризують потенціал криптопроекту. Кожен критерій оцінюється за п’ятибальною шкалою, де 5 балів означає відмінний показник з високим потенціалом, 3 бали відповідають середньому рівню з певними ризиками, а 1 бал сигналізує про серйозні проблеми або відсутність перспектив.

Застосування токена оцінює, наскільки токен інтегрований в екосистему проекту та чи має він реальну функціональну цінність. Високі бали отримують проекти, де токен є незамінним для доступу до послуг, участі в управлінні або забезпечення безпеки мережі. Низькі оцінки характерні для токенів, що мають переважно спекулятивну цінність без чіткого застосування.

Рисунок 2.5 – Генерація результатів аналізу про проект

Бізнес-модель аналізує економічну основу проекту, його здатність генерувати доходи та забезпечувати довгострокову стійкість. Найвищі бали

отримують проекти з диверсифікованими джерелами доходу, чіткою монетизацією та доведеною здатністю до масштабування.

Критерій вирішення проблеми оцінює актуальність завдань, які вирішує проект, та ефективність запропонованих рішень. Особлива увага приділяється розміру цільового ринку та наявності альтернативних рішень від конкурентів.

Система проводить комплексний ринковий аналіз, який включає класифікацію проекту за ринковою капіталізацією, визначення його позицій у відповідному секторі та порівняння з конкурентами. Класифікація за капіталізацією допомагає інвесторам розуміти рівень ризику та потенційної прибутковості інвестицій.

Проекти з капіталізацією понад 200 млрд. дол. США класифікуються як мегакап і характеризуються високою стабільністю але обмеженим потенціалом зростання. Велика капіталізація від 10 до 200 млрд. дол. США представляє збалансований варіант з помірними ризиками та хорошими перспективами. Середня капіталізація від 1 до 10 млрд. дол. США часто демонструє високий потенціал зростання при підвищених ризиках.

Мала капіталізація від 100 млн. дол. США до 1 млрд. дол. США типова для перспективних проектів на ранніх стадіях розвитку з високою волатильністю. Мікрокап від 10 до 100 млн. дол. США та нанокап до 10 млн. дол. США представляють найризикованіші інвестиції з потенціалом екстремального зростання або повної втрати коштів.

Оцінка активності розробників є критично важливим компонентом аналізу, оскільки технічний прогрес безпосередньо впливає на перспективи проекту. Система аналізує GitHub-репозиторії проектів, підраховуючи кількість комітів за останній місяць та оцінюючи регулярність оновлень.

Висока активність з понад 20 комітами на місяць свідчить про інтенсивну розробку та хороші перспективи технічного розвитку. Активність вище середнього з 15-20 комітами вказує на стабільну підтримку проекту. Середня

активність з 8-15 комітами може бути достатньою для зрілих проектів, але викликає питання для нових платформ.

Активність нижче середнього з 4-8 комітами та особливо низька активність з менш ніж 4 комітами можуть сигналізувати про проблеми з фінансуванням розробки або втрату інтересу команди до проекту.

Аналіз безпеки включає перевірку наявності професійних аудитів смарт-контрактів та загальної архітектури проекту. Система автоматично шукає інформацію про аудити у базі CoinMarketCap та надає детальні дані про аудиторські компанії, дати проведення аудитів та посилання на звіти.

Наявність аудитів від визнаних компаній значно підвищує рівень довіри до проекту та зменшує ризики для інвесторів. Відсутність аудитів не обов’язково означає ненадійність проекту, але вимагає додаткової обережності та ретельного вивчення коду.

Система генерує детальні візуальні звіти, що включають діаграми розподілу токенів, графіки ринкових позицій та інфографіку ключових показників. Токеноміка представляється у вигляді інтерактивних кругових діаграм з детальною розбивкою по категоріях: токени в обігу, загальна пропозиція та максимальна пропозиція.

Фінальний звіт структурується для максимальної читабельності та включає резюме з ключовими висновками, детальний аналіз за всіма критеріями, ринкові дані та рекомендації для інвесторів. Система автоматично адаптує довжину повідомлень під обмеження Telegram, розбиваючи великі звіти на логічні частини.

2.3 Програмна реалізація чатботу

2.3.1 Використання API для отримання фінансових даних

У проекті використовується кілька API для отримання фінансових даних про криптовалюти. Порядок використання API

Розглянемо реалізацію використання API.

Основним джерелом криптовалютних даних є CoinGecko API, для використання якого реалізована функція `get_coin_data()`:

```
@retry_on_rate_limit
def get_coin_data(url, params=None):
    try:
        response = requests.get(url, params=params)
        return response
    except requests.exceptions.RequestException as e:
        logging.error(f"Помилка при запиті до {url}: {e}")
        return None
```

У функції декоратор `@retry_on_rate_limit` автоматично повторює запит при отриманні HTTP 429 (перевищення ліміту).

Безпосередньо у функції виконується GET-запит до переданої URL з параметрами. При помилці повертається `None`, інакше – об’єкт `response`.

Для пошуку ідентифікатора монети застосовується функція `search_coin_id()`:

```
def search_coin_id(query):
                                                search_url            =
f"https://api.coingecko.com/api/v3/search?query={query}"
    response = get_coin_data(search_url)
    if response and response.status_code == 200:
        data = response.json()
        if data["coins"]:
            coin_id = data["coins"][0]["id"]
            logging.info(f"Знайдено ідентифікатор монети: {coin_id}")
            return coin_id
        else:
            logging.warning(f"Монета '{query}' не знайдена у пошуку.")
            return None
    else:
```

```
                logging.error(f"Не вдалося отримати дані пошуку для
'{query}'.")
        return None
```

Спочатку у функції формується URL для пошуку з назвою проекту. Далі виконується запит до CoinGecko Search API, після чого перевіряється статус відповіді (200 OK). На наступному кроці парситься отриманий JSON і витягується перша знайдена монета. У результаті роботи повертається унікальний `coin_id` для подальших запитів.

Ринкові дані монет отримуються за допомогою функції `get_all_coins_market_data()`:

```
def get_all_coins_market_data(coin_ids):
    if not coin_ids:
        logging.error("Список ідентифікаторів монет порожній.")
        return []

    url = "https://api.coingecko.com/api/v3/coins/markets"
    params = {
        "vs_currency": "usd",
        "ids": ",".join(coin_ids),
        "order": "market_cap_desc",
        "per_page": len(coin_ids),
        "page": 1,
        "sparkline": "false",
    }
    response = get_coin_data(url, params=params)
    if response and response.status_code == 200:
        coins_data = response.json()
        return coins_data
    else:
        logging.error("Не вдалося отримати дані монет.")
        return []
```

На першому кроці у функції перевіряється, чи переданий список ID не порожній. Далі формуються параметри запиту, де вказуються:

– `vs_currency` – валюта для відображення цін (USD);

– `ids` – список ID монет;

– `order` – сортування за ринковою капіталізацією;

– `per_page` – кількість результатів на сторінку;

На наступному кроці виконується запит до Markets API, після якого повертається масив з ринковими даними монет.

У функції `check_coin_on_cmc()` проводиться перевірка наявності даних на CoinMarketCap (CMC):

```
def check_coin_on_cmc(symbol):
    if not CMC_API_KEY:
        logging.warning("CMC_API_KEY не знайдено у .env файлі.")
        return False

    headers = {"X-CMC_PRO_API_KEY": CMC_API_KEY}
    params = {"symbol": symbol}

    url = "https://pro-api.coinmarketcap.com/v1/cryptocurrency/info"

    try:
        response = requests.get(url, headers=headers, params=params)
        if response.status_code == 200:
            data = response.json()
            return symbol.upper() in data.get("data", {})
        else:
                    logging.error(f"Помилка CoinMarketCap API. Код:
{response.status_code}")
            return False
    except requests.exceptions.RequestException as e:
        logging.error(f"Помилка при запиті до CoinMarketCap API: {e}")
        return False
```

Спочатку у функції перевіряється наявність API ключа CMC. У випадку успіху формуються заголовки з автентифікацією (`X-CMC_PRO_API_KEY`), у якості параметру для яких передається символ криптовалюти.

Далі відбувається запит до Cryptocurrency Info API, після чого перевіряється наявність символу в отриманих даних.

Перевірка аудиту точена проводиться у функції `check_token_audit()`:

```
def check_token_audit(coin_data):
    if not CMC_API_KEY:
        logging.warning("CMC_API_KEY не знайдено у .env файлі.")
            return "Інформація про аудит недоступна (CMC_API_KEY не
знайдено)."
```

```
    symbol = coin_data.get("symbol", "").upper()
    headers = {"X-CMC_PRO_API_KEY": CMC_API_KEY}
    url = "https://pro-api.coinmarketcap.com/v1/cryptocurrency/info"
    params = {"symbol": symbol}

    try:
        response = requests.get(url, headers=headers, params=params)
        if response.status_code == 200:
            data = response.json()
            if symbol in data.get("data", {}):
                coin_info = data["data"][symbol]
                audits = coin_info.get("audits", [])
                if audits:
                    audit_info = ""
                    for audit in audits:
                        name = audit.get("name", "N/A")
                        audit_date = audit.get("date", "N/A")
                        audit_link = audit.get("url", "N/A")
                             audit_info += f"- Назва аудитора: {name}\n
Дата: {audit_date}\n  Посилання: {audit_link}\n"
                    return f"Так, аудит доступний:\n{audit_info}"
                else:
                            return "Інформація про аудит не знайдена на
CoinMarketCap."
```

На початку функції витягується символ токена з даних CoinGecko й виконується запит до CMC Cryptocurrency Info API.

У отриманій інформації про токен шукається масив `audits`. Якщо аудити знайдені, формується детальна інформація про токен:

- назва аудиторської компанії;
- дата проведення аудиту;
- посилання на звіт.

У якості відповіді повертається форматований текст з результатами.

Активність розробників з використанням GitHub API перевіряється за допомогою функції `get_developer_activity()`, у якій проводиться аналіз комітів:

```
def get_developer_activity(coin_data):
    if "links" in coin_data and "repos_url" in coin_data["links"]:
           github_repos = coin_data["links"]["repos_url"].get("github",
[])
        if github_repos:
            github_url = github_repos[0]
```

```
            repo_name = github_url.split("https://github.com/")[-1]

            # Отримуємо дату місяць тому від поточної дати
            since_date = (
                                        datetime.datetime.utcnow()   -
datetime.timedelta(days=30)
            ).isoformat() + "Z"

                                                                api_url     =
f"https://api.github.com/repos/{repo_name}/commits"
            params = {"since": since_date}

            headers = (
                {"Authorization": f"token {GITHUB_API_TOKEN}"}
                if GITHUB_API_TOKEN
                else {}
            )

            try:
                                github_response  =  requests.get(api_url,
headers=headers, params=params)
                if github_response.status_code == 200:
                    commits_data = github_response.json()
                    commit_count = len(commits_data)
                    return commit_count, since_date
```

На початку функції проводиться пошук відповідного GitHub-репозиторію, для чого з даних CoinGecko береться посилання на GitHub та парситься назва репозиторію з URL

Далі формуються часові рамки: розраховується дата 30 днів від поточного моменту та конвертується в ISO формат з суфіксом ‘Z’.

На наступному кроці проводиться автентифікація. Якщо заданий GitHub-токен існує, то він додається у заголовки, інакше запит виконується без автентифікації (з лімітами).

Далі проводиться запит до Commits API, для якого вказується параметр since, який фільтрує коміти після вказаної дати, та отримується масив усіх комітів за період.

Отримана кількість комітів та початкова дата повертаються з функції в якості відповіді.

Аналіз сектору ринкових даних проводиться за допомогою функції `get_sector_market_cap_and_rank()`:

```
def get_sector_market_cap_and_rank(
    coin_id, main_sector, project_market_cap, market_cap_classification
):
    # Отримуємо ідентифікатор категорії (сектору) за назвою сектору
    category_id = get_category_id_by_name(main_sector)
    if not category_id:
            logging.warning(f"Категорія для сектору '{main_sector}' не
знайдена.")
        return None, "Дані недоступні"

    # Отримуємо дані про монети в цій категорії (секторі)
    url = "https://api.coingecko.com/api/v3/coins/markets"
    params = {
        "vs_currency": "usd",
        "category": category_id,
        "order": "market_cap_desc",
        "per_page": 250,  # Максимальна кількість монет за запит
        "page": 1,
        "sparkline": "false",
    }
    response = get_coin_data(url, params=params)
    # ... подальша обробка
```

Спочатку за допомогою виклику функції `get_category_id_by_name()` знаходиться ID категорії. Для отримання монет сектору проводиться запит до Markets API з фільтром по категорії.

З отриманої відповіді відбираються монети з такою ж класифікацією (мікрокап, мала капіталізація тощо).

На наступному кроці розраховується сумарна капіталізація сектору.

Нарешті, у кінці функції проводиться визначення рангу проекту серед конкурентів у секторі.

У функції `process_project_info_with_ai()` проводиться обробка через ШІ шляхом інтеграції з OpenAI API:

```
def process_project_info_with_ai(full_prompt):
    try:
             logging.info("Обробка інформації про проект через OpenAI
API...")

        response = openai.ChatCompletion.create(
            model="gpt-3.5-turbo",
            messages=[
```

```
                {
                    "role": "system",
                            "content": "Ти аналітик криптовалют, який
відповідає на питання по проекту.",
                },
                {
                    "role": "user",
                    "content": full_prompt,
                },
            ],
            max_tokens=1500,
            n=1,
            stop=None,
            temperature=0.5,
        )
        result = response.choices[0].message["content"].strip()
        logging.info("Інформацію про проект оброблено.")
        return result
    except openai.error.RateLimitError:
        logging.warning("Перевищено ліміт запитів OpenAI. Очікування 60
секунд...")
        time.sleep(60)
        return process_project_info_with_ai(full_prompt)
```

У запиті, що будується, спочатку формується контекст: встановлюється роль “аналітик криптовалют”. Далі весь зібраний опис проекту передається як користувацький запит, при цьому використовуються наступні параметри:

– `max_tokens` – максимум 1500 токенів у відповіді;

– `temperature` – 0.5 для балансу між креативністю та точністю;

При перевищенні лімітів запитів функція чекає 60 секунд і повторює запит знову.

2.3.2 Обробка фінансових даних

Розглянемо основні дії, що проводяться з фінансовими показниками.

Отримання базових фінансових даних з CoinGecko:

```
def analyze_project(project_name):
    # ... код пошуку монети ...

    gecko_url = f"https://api.coingecko.com/api/v3/coins/{coin_id}"
    response = get_coin_data(gecko_url)

    if response and response.status_code == 200:
        coin_data = response.json()
```

У функції отримується повний набір фінансових даних про криптопроект з API CoinGecko, включаючи ринкову капіталізацію, обсяги торгів, ціни та інші метрики.

Для обробки ринкової капіталізації проекту отримується її значення з вкладеної структури JSON, для чого використовується безпечний метод `.get()` для уникнення помилок при відсутності даних. Наступним кроком форматується число з роздільниками тисяч для зручності читання та обробляється випадок відсутності даних:

```
# Ринкова капіталізація проекту
project_market_cap = (
     coin_data.get("market_data", {}).get("market_cap", {}).get("usd",
None)
)
formatted_project_market_cap = (
     f"{project_market_cap:,.0f} USD" if project_market_cap else "Дані
недоступні"
)
```

Класифікація проектів за ринковою капіталізацією проводиться у функції `classify_by_market_cap()`:

```
def classify_by_market_cap(market_cap):
    if market_cap >= 200_000_000_000:
        return "Мегакап"
    elif 10_000_000_000 <= market_cap < 200_000_000_000:
        return "Велика капіталізація"
    elif 1_000_000_000 <= market_cap < 10_000_000_000:
        return "Середня капіталізація"
    elif 100_000_000 <= market_cap < 1_000_000_000:
        return "Мала капіталізація"
    elif 10_000_000 <= market_cap < 100_000_000:
        return "Мікрокап"
    else:
        return "Нанокап"
```

Функція приймає числове значення ринкової капіталізації. Використовуючи ієрархічну систему умов для класифікації, функція визначає категорію капіталізації та повертає її відповідно до розміру капіталізації. До

можливих категорій входять інвестиційні категорії від “Мегакап” (>$200 млрд) до “Нанокап” (<$10 млн).

Аналіз позиції проекту в секторі проводиться за допомогою функції `get_sector_market_cap_and_rank()`.

Спочатку у функції проводиться пошук ID категорії за назвою й отримується список усіх монет у секторі, відсортованих за капіталізацією. З отриманого списку відбираються лише проекти того ж класу капіталізації:

```
def get_sector_market_cap_and_rank(
    coin_id, main_sector, project_market_cap, market_cap_classification
):
    # Отримуємо ідентифікатор категорії (сектору) за назвою сектору
    category_id = get_category_id_by_name(main_sector)

    # Отримуємо дані про монети в цій категорії (секторі)
    url = "https://api.coingecko.com/api/v3/coins/markets"
    params = {
        "vs_currency": "usd",
        "category": category_id,
        "order": "market_cap_desc",
        "per_page": 250,
        "page": 1,
        "sparkline": "false",
    }
    response = get_coin_data(url, params=params)
```

Для усіх відфільтрованих проектів знаходиться сума ринкових капіталізацій, за якою проекти сортуються й відшукується позиція аналізованого проекту:

```
# Фільтруємо монети за класифікацією ринкової капіталізації
sector_coins_filtered = [
    coin
    for coin in all_sector_coins_data
                if    classify_by_market_cap(coin["market_cap"])    ==
market_cap_classification
]

# Розраховуємо капіталізацію сектору
sector_market_cap = sum(
    coin["market_cap"]
    for coin in sector_coins_filtered
    if coin["market_cap"] is not None
```

```
)

# Визначаємо місце проекту в секторі
sorted_sector_coins = sorted(
        sector_coins_filtered, key=lambda x: x["market_cap"] or 0,
reverse=True
)

project_rank = next(
    (i + 1 for i, coin in enumerate(sorted_sector_coins) if coin["id"]
== coin_id),
    "Дані недоступні",
)
```

Обробка токеноміки проводиться у функції `get_tokenomics_data()`:

```
def get_tokenomics_data(coin_data):
    if "market_data" in coin_data:
        market_data = coin_data["market_data"]
        tokenomics_data = {
                 "Обіг в обігу": market_data.get("circulating_supply",
"N/A"),
                "Загальна пропозиція": market_data.get("total_supply",
"N/A"),
               "Максимальна пропозиція": market_data.get("max_supply",
"N/A"),
        }
        for key, value in tokenomics_data.items():
            if isinstance(value, (int, float)):
                tokenomics_data[key] = f"{value:,.0f}"
            elif value is None:
                tokenomics_data[key] = "N/A"
        return tokenomics_data
```

Спочатку витягуються три ключові показники: circulating supply, total supply, max supply.

Далі форматуються числові значення з роздільниками тисяч та обробляються відсутні дані (`None`) шляхом перетворення їх у `N/A`.

Структуровані дані повертаються для подальшої візуалізації.

2.3.3 Використання API моделей ШІ для аналізу фінансових даних

Налаштування та підключення до OpenAI здійснюється шляхом підключення необхідних модулів та виконання певної послідовності функцій.

Програма використовує OpenAI GPT-3.5-turbo для проведення фінансового аналізу криптопроектів. Для безпеки API ключ зберігається в окремому env-файлі.

```
import openai
from dotenv import load_dotenv

# Завантаження змінних оточення
load_dotenv()

# Отримуємо токен API з .env
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
openai.api_key = OPENAI_API_KEY

# Перевірка наявності ключа API
if not OPENAI_API_KEY:
    logging.error("OPENAI_API_KEY не знайдено у .env файлі.")
```

Основною функцією обробки через ШІ є функція `process_project_info_with_ai()`, описана раніше.

Найскладніша частина використання ШІ – структурований аналіз, який проводиться у функції `ai_detailed_analysis()`:

```
def ai_detailed_analysis(description):
    prompt = f"""
Проведи аналіз і дай оцінку виходячи з наступної інформації:

{description}

Думай як фінансовий аналітик Wall Street.

Оціни проект за наступними 8 критеріями, використовуючи надані питання
і критерії оцінки...

1. Застосування токена (Utility)
Питання: Наскільки корисний токен в екосистемі проекту?
Критерії оцінки:
5 балів: Токен має ключову функціональну роль...
3 бали: Токен має обмежене застосування...
1 бал: Токен не має значущої функціональної ролі...

[... аналогічно для всіх 8 критеріїв ...]
"""
```

Критеріями аналізу є наступні:

- застосування токена –функціональна роль точена;
- бізнес-модель – стійкість та масштабованість;
- вирішення проблеми – актуальність рішення;
- перспективи Web3 – відповідність трендам децентралізації;
- технологічна перевага – інноваційність технологій;
- ризики – регуляторні та ринкові загрози;
- конкурентне середовище – позиції відносно конкурентів;
- економічна модель токена – стимули для утримання.

У проекті для структурованого парсингу відповіді ШІ та витягування оцінок і пояснень використовуються регулярні вирази, наприклад:

```
# Розбиваємо відповідь ШІ на рядки
lines = analysis_result.strip().split("\n")

for line in lines:
    line = line.strip()
    if re.match(r"^Підсумкова оцінка", line):
        # Парсимо підсумкову оцінку
                                    match    =    re.match(r"^Підсумкова
оцінка\s*:\s*(\d+)\s*балів?\s*(.*)", line)
        if match:
            total_score = int(match.group(1).strip())
            final_evaluation = match.group(2).strip()
    else:
        # Парсимо оцінки за критеріями
        match = re.match(r"^(\d\.)\s*(.*?)\s*:\s*(\d)\s*балів?\s*(.*)",
line)
```

У проекті для виведення остаточної відповіді поєднуються результати ШІ-аналізу з реальними фінансовими даними:

```
def analyze_project(project_name):
    # Отримання даних з CoinGecko API
    coin_data = response.json()

    # Аналіз ринкових показників
                project_market_cap    =    coin_data.get("market_data",
{}).get("market_cap", {}).get("usd")
                                       market_cap_classification        =
classify_by_market_cap(project_market_cap)
```

```
    # ШІ-аналіз опису проекту
    description = clean_html(coin_data.get("description", {}).get("en",
""))
          ai_scores,  ai_answers,  total_score,  final_evaluation  =
ai_detailed_analysis(description)

    # Поєднання в єдиному звіті
    report = generate_report(
        name, symbol, main_sector, risk_group,
        formatted_sector_market_cap, market_cap_classification,
        # ... інші параметри ...
        ai_answers, ai_scores, total_score, final_evaluation
    )
```

Описана архітектура дозволяє автоматизувати складний процес фундаментального аналізу криптопроектів, поєднуючи потужність ШІ з актуальними ринковими даними.

2.3.4 Створення чатботу та наповнення функціональністю

Розглянемо архітектуру та функціональність створеного Telegram-бота.

Проект складається з двох основних модулів:

– `bot.py` – основний файл бота з обробкою команд і повідомлень;

– `analysis.py` – модуль аналітики з інтеграцією зовнішніх API.

На початку роботи налаштовується основна інфраструктура бота:

– імпортуються необхідні бібліотеки;

– налаштовується система логування для відстеження роботи;

– завантажуються змінні оточення з env-файлу.

```
import os
import telebot
from analysis import analyze_project
from dotenv import load_dotenv
import logging

# Налаштування логування
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)

# Завантаження змінних оточення
```

```
load_dotenv()
```

Безпечна ініціалізація бота з валідацією наявності токена та аварійним завершенням при його відсутності виконується за допомогою наступного коду:

```
TOKEN = os.getenv('TELEGRAM_BOT_TOKEN')
if not TOKEN:
    logging.error("TELEGRAM_BOT_TOKEN не знайдено у .env файлі.")
    exit("Необхідно вказати TELEGRAM_BOT_TOKEN у файлі .env")

bot = telebot.TeleBot(TOKEN)
```

Оскільки у Telegram наявне обмеження на довжину повідомлень (4096 символів), дуже великі повідомлення (якими, наприклад, і є запити до OpenAI API) необхідно розбивати на частини. Для цього реалізована функція `split_message()`, у якій текст розбивається за абзацами для збереження логічної структури, створює масив повідомлень, кожне з яких не перевищує ліміт, та повертається у місце виклику. Таким чином, за допомогою цієї функції зберігається цілісність контенту.

Код функції має вигляд:

```
def split_message(message, max_length=4000):
    paragraphs = message.split('\n\n')
    messages = []
    current_message = ''
    for paragraph in paragraphs:
        if len(current_message) + len(paragraph) + 2 <= max_length:
            current_message += paragraph + '\n\n'
        else:
            messages.append(current_message)
            current_message = paragraph + '\n\n'
    if current_message:
        messages.append(current_message)
    return messages
```

Функція `send_welcome()` є привітальним обробником бота. Загалом вона обробляє команди `/start` та `/help`, а на початку роботи надсилає привітальне повідомлення з інструкціями:

```
@bot.message_handler(commands=['start', 'help'])
def send_welcome(message):
    logging.info(f"Користувач {message.chat.id} запустив бота.")
    bot.reply_to(
        message,
           "Вітаю! Я бот для фундаментального аналізу криптопроектів. Просто надішліть мені назву проекту, і я підготую для вас детальний аналіз."
    )
```

Основним обробником проектів є функція `handle_project()`. Ця функція приймає всі текстові повідомлення, для кожного отриманого повідомлення витягує назву проекту та інформує користувача про початок аналізу:

```
@bot.message_handler(func=lambda message: True)
def handle_project(message):
    project_name = message.text.strip()
    user_id = message.chat.id
      logging.info(f"Отримано запит на аналіз проекту '{project_name}' від користувача {user_id}")

    # Надсилаємо сповіщення про початок аналізу
        bot.send_message(user_id, f"Починаю детальний аналіз проекту '{project_name}'. Це може зайняти кілька хвилин, будь ласка, зачекайте...")
```

Виклик функції аналізу проекту відбувається у наступному блоці команд:

```
try:
    # Аналіз проекту
    response, tokenomics_image_path = analyze_project(project_name)

    # Надсилаємо текстовий звіт
    if response:
        messages = split_message(response)
        for idx, msg in enumerate(messages):
            bot.send_message(user_id, msg)
            if idx < len(messages) - 1:
                time.sleep(1)  # Затримка між повідомленнями
```

У поданому коді викликається функція аналізу з модуля `analysis.py`, отримується текстовий звіт і шлях до зображення токеноміки, після чого довгий звіт розбивається на частини та надсилається з затримкою.

Загальний запуск бота проводиться у результаті виконання наступного блоку команд:

```
if __name__ == '__main__':
    logging.info("Бот запущено і готовий до прийому повідомлень.")
    bot.infinity_polling()
```

Бот запускається в режимі постійного опитування Telegram серверів.

Характеристиками створеного проекту є наступні:

- модульність – розділення логіки бота та аналітики дозволяє сконцентруватись на конкретній задачі;
- надійність – використання системи retry та обробки помилок підвищує ймовірність виконання команд боту;
- масштабованість – за рахунок використаної архітектури наявна можливість додавання нових API та функцій.

Отриманий бот надає комплексний аналіз криптопроектів, поєднуючи дані з кількох джерел, та ШІ-аналіз для формування обґрунтованих висновків.

Повний код реалізації чабботу подано у додатку А.

РОЗДІЛ 3
ТЕСТУВАННЯ ТА СУПРОВОДЖЕННЯ ЧАТБОТУ

3.1 Вибір методів тестування

Методи тестування програмного забезпечення можна класифікувати за різними критеріями, кожен з яких має свої особливості та сфери застосування. Розуміння цих методів критично важливе для вибору оптимальної стратегії тестування конкретного проекту.

Класифікація за рівнем тестування включає модульне тестування, яке перевіряє окремі компоненти в ізоляції, інтеграційне тестування для перевірки взаємодії між модулями, системне тестування всієї системи як єдиного цілого, та приймальне тестування з боку користувачів або замовників.

За способом виконання методи поділяються на ручні, де тестувальник особисто виконує тестові сценарії, та автоматизовані, що використовують спеціальні інструменти та скрипти для виконання тестів.

За знанням внутрішньої структури розрізняють тестування “чорної скриньки”, коли тестується лише функціональність без знання коду, “білої скриньки” з повним доступом до вихідного коду, та “сірої скриньки”, що поєднує обидва підходи.

За характером перевірки виділяють функціональне тестування, яке перевіряє відповідність функціональним вимогам, та нефункціональне, що включає продуктивність, безпеку, зручність використання тощо.

У таблиці 3.1 наведено характеристики методів тестування та визначено застосовність методів до розроблюваного проекту.

Критично важливі методи для розроблюваного проекту включають Unit тестування, оскільки система містить багато функцій обробки даних, класифікації та валідації, які потребують ретельної перевірки в ізоляції. Integration тестування є обов’язковим через критичну залежність від зовнішніх API сервісів – CoinGecko, CoinMarketCap, GitHub та OpenAI. API тестування

виокремлюється як спеціалізований вид integration тестування, оскільки система практично повністю побудована навколо API взаємодій.

Таблиця 3.1. Характеристики методів тестування та їх застосовність до розроблюваного проекту

Метод тестування	Опис	Застосов-ність до проекту	Пріори-тет	Обґрунтування вибору
Unit-тестування	Тестування окремих функцій і методів у ізоляції	Високо-застосовний	Критич-ний	Необхідний для тестування функцій класифікації, обробки даних, валідації. Швидке виявлення помилок у бізнес-логіці
Integration-тестування	Перевірка взаємодії між модулями та зовнішніми API	Високо-застосовний	Критич-ний	Критично важливо через численні залежності від API (CoinGecko, GitHub, OpenAI)
End-to-End тестування	Тестування повного циклу від користувацького вводу до результату	Застосовний	Високий	Забезпечує перевірку повного циклу: Telegram → аналіз → відповідь
Тестування продуктив-ності	Перевірка швидкодії та навантаження	Застосовний	Високий	Важливо через обмеження Telegram API та очікування користувачів
Тестування безпеки	Перевірка захисту від атак та витоку даних	Застосовний	Високий	Критично для захисту API ключів та обробки користувацьких даних
Smoke-тестування	Базова перевірка ключової функціональності	Застосовний	Середній	Швидка перевірка після деплою, особливо для API інтеграцій

Продовження таблиці 3.1

Метод тестування	Опис	Застосов-ність до проекту	Пріори-тет	Обґрунтування вибору
Regression-тестування	Перевірка, що нові зміни не зламали існуючий функціонал	Застосовний	Високий	Важливо через часті оновлення зовнішніх API та AI моделей
Тестування API	Спеціалізоване тестування REST API взаємодій	Високо-застосовний	Критич-ний	Система критично залежить від зовнішніх API сервісів
Завантажувальне тестування	Перевірка поведінки під навантаженням	Умовно-застосовний	Середній	Потрібно для перевірки одночасних запитів користувачів
Stress-тестування	Тестування за межами нормального навантаження	Обмежено-застосовний	Низький	Обмежено через rate limits зовнішніх API
Usability-тестування	Перевірка зручності використання	Застосовний	Середній	Важливо для Telegram інтерфейсу та зрозумілості звітів
Тестування сумісності	Перевірка сумісності з різними платформами	Обмежено застосовний	Низький	Telegram бот працює на сервері, клієнтська сумісність менш критична
A/B-тестування	Порівняння різних варіантів функціональності	Обмежено застосовний	Низький	Можливо для різних форматів звітів, але не критично
Тестування доступності	Перевірка доступності для людей з обмеженнями	Не застосовний	Не релевантний	Telegram бот має обмежені можливості доступності
Тестування бази даних	Перевірка операцій з базою даних	Не застосовний	Не реле-вантний	Проект не використовує власну базу даних
UI-тестування	Тестування користувацького інтерфейсу	Не застосовний	Не реле-вантний	Немає вебінтерфейсу, лише Telegram

Високоприоритетні методи включають End-to-End тестування для забезпечення коректності повного циклу роботи від користувацького запиту в Telegram до отримання аналітичного звіту. Performance тестування критично важливе через обмеження швидкості роботи Telegram API та очікування користувачів щодо швидкості відповіді. Security тестування необхідне для захисту API ключів та запобігання витоку чутливої інформації. Regression тестування набуває особливої важливості через динамічність криптовалютного ринку та часті оновлення зовнішніх сервісів.

Ключові особливості вибору зумовлені архітектурою системи як Telegram-бота з багатьма зовнішніми залежностями. Високий пріоритет API та Integration тестування пояснюється тим, що більшість функціональності залежить від зовнішніх сервісів. Важливість Security тестування зростає через роботу з фінансовими даними та необхідність захисту ключів API. Performance тестування критично важливе через користувацькі очікування швидкості відповіді в месенджері.

Вказана комбінація методів тестування забезпечить комплексну перевірку всіх аспектів системи при оптимальному використанні ресурсів розробки, враховуючи специфіку криптоаналітичного бота та його технічні особливості.

3.2 Тестовий план проекту

Представлена система являє собою складний програмний комплекс, що складається з двох взаємопов’язаних модулів. Модуль `bot.py` реалізує інтерфейс взаємодії з користувачами через Telegram API, забезпечуючи прийом запитів та відправку результатів аналізу. Модуль `analysis.py` містить основну бізнес-логіку системи, включаючи інтеграцію з множинними зовнішніми API сервісами, обробку даних криптовалют та генерацію аналітичних звітів з використанням штучного інтелекту.

Для ефективного тестування такої системи доцільно застосувати підхід пірамідки тестування, де основу складають швидкі та надійні unit-тести,

покриваючи приблизно 70% всього тестового покриття. Інтеграційні тести займають середній рівень з 20% покриття, фокусуючись на взаємодії між компонентами та зовнішніми сервісами. Вершину пірамідки формують end-to-end тести з 10% покриття, що перевіряють повний цикл роботи системи від користувацького вводу до фінального результату.

Особливістю даної системи є її висока залежність від зовнішніх API сервісів, таких як CoinGecko, CoinMarketCap, GitHub та OpenAI. Це створює додаткові виклики для тестування, оскільки необхідно забезпечити стабільність тестів навіть при недоступності або нестабільній роботі зовнішніх сервісів. Також важливо враховувати волатильність даних криптовалютного ринку, де ціни та рейтинги можуть змінюватися кожну секунду.

Загальний план тестування має наступний вигляд:

1) проведення Unit-тестів;
2) проведення інтеграційних тестів;
3) тестування продуктивності системи;
4) тестування безпеки системи;
5) повне тестування Telegram-бота.

Unit-тести формують фундамент тестової стратегії, забезпечуючи перевірку окремих функцій в ізоляції від зовнішніх залежностей. Для модуля `analysis.py` ключовими об’єктами тестування є функції обробки даних, класифікації та валідації вхідної інформації.

Відповідний файл `test_analysis.py` для проведення тестування має наступний вигляд:

```
import pytest
from analysis import clean_html, classify_by_market_cap, evaluate_developer_activity

class TestAnalysisFunctions:
    def test_clean_html_removes_tags(self):
        # Тестуємо функцію видалення HTML тегів з тексту
        # Вхідні дані містять різні типи HTML елементів
```

```
                    html_text = "<p>Test <b>text</b> with <a
href='#'>links</a></p>"
        expected = "Test text with links"
        # Перевіряємо, що всі HTML теги видалені коректно
        assert clean_html(html_text) == expected

    @pytest.mark.parametrize("market_cap,expected", [
        # Тестуємо класифікацію мегакап токенів (понад 200 млрд)
        (250_000_000_000, "Мегакап"),
        # Тестуємо класифікацію великої капіталізації (10-200 млрд)
        (50_000_000_000, "Велика капіталізація"),
        # Тестуємо середню капіталізацію (1-10 млрд)
        (5_000_000_000, "Середня капіталізація"),
        # Тестуємо малу капіталізацію (100 млн - 1 млрд)
        (500_000_000, "Мала капіталізація"),
        # Тестуємо мікрокап (10-100 млн)
        (50_000_000, "Мікрокап"),
        # Тестуємо нанокап (менше 10 млн)
        (5_000_000, "Нанокап")
    ])
    def test_classify_by_market_cap(self, market_cap, expected):
            # Параметризований тест для перевірки всіх категорій
капіталізації
          # Використовуємо різні значення для покриття всіх граничних
випадків
        assert classify_by_market_cap(market_cap) == expected

    def test_evaluate_developer_activity_high(self):
        # Тестуємо оцінку високої активності розробників
        # При 25 комітах за місяць очікуємо високу активність
        activity, rating = evaluate_developer_activity(25)
        assert activity == "Висока активність"
        assert rating == "9-10"
```

У наведеному коді проводиться тестування трьох ключових функцій системи. Функція `test_clean_html_removes_tags()` перевіряє коректність видалення HTML-розмітки з описів проектів, що критично важливо для безпеки та читабельності звітів. Параметризований тест `test_classify_by_market_cap()` покриває всі можливі категорії ринкової капіталізації, використовуючи представницькі значення для кожного діапазону. Тест `test_evaluate_developer_activity_high()` перевіряє правильність оцінки активності розробників на GitHub.

Інтеграційні тести фокусуються на перевірці взаємодії системи з зовнішніми сервісами. Ключовою особливістю таких тестів є використання

мокування для симуляції відповідей API без реальних запитів до зовнішніх сервісів.

Код програми інтеграційного тестування `test_api_integration.py` має наступний вигляд:

```
import pytest
import responses
from analysis import get_coin_data, search_coin_id

class TestAPIIntegration:
    @responses.activate
    def test_search_coin_id_success(self):
        # Мокуємо успішну відповідь від CoinGecko API
        # Створюємо структуру даних, аналогічну реальній відповіді API
        mock_response = {
            "coins": [{"id": "bitcoin", "name": "Bitcoin"}]
        }
        # Реєструємо мок для конкретного URL та методу
        responses.add(
            responses.GET,
            "https://api.coingecko.com/api/v3/search",
            json=mock_response,
            status=200
        )

        # Викликаємо функцію пошуку та перевіряємо результат
        result = search_coin_id("bitcoin")
        assert result == "bitcoin"

    @responses.activate
    def test_api_rate_limit_handling(self):
        # Тестуємо обробку обмеження частоти запитів (rate limiting)
        # Перший запит повертає помилку 429 з заголовком Retry-After
        responses.add(
            responses.GET,
            "https://api.coingecko.com/api/v3/search",
            status=429,
            headers={"Retry-After": "1"}
        )
        # Другий запит (після повтору) повертає успішну відповідь
        responses.add(
            responses.GET,
            "https://api.coingecko.com/api/v3/search",
            json={"coins": [{"id": "bitcoin"}]},
            status=200
        )

        # Перевіряємо, що система коректно обробляє rate limiting
        result = search_coin_id("bitcoin")
        assert result == "bitcoin"
```

Поданий код ілюструє два критично важливих аспекти інтеграційного тестування. Перший тест перевіряє базову функціональність пошуку криптовалют через API CoinGecko, використовуючи мокування для створення передбачуваної відповіді. Другий тест демонструє перевірку механізму обробки rate limiting, що особливо важливо для API криптовалютних сервісів, які часто мають суворі обмеження на кількість запитів.

Враховуючи специфіку роботи з зовнішніми API та обробку великих обсягів даних, критично важливо *тестувати продуктивність системи*. Особливу увагу слід приділити часу відгуку та здатності системи обробляти одночасні запити.

Код відповідної програми, `test_performance.py`, подано нижче:

```
import time
import pytest
from concurrent.futures import ThreadPoolExecutor
from analysis import analyze_project

class TestPerformance:
    def test_analysis_response_time(self):
        # Тестуємо час відгуку системи аналізу
        # Записуємо час початку виконання
        start_time = time.time()
        # Викликаємо повний аналіз популярного токена
        analyze_project("bitcoin")
        # Записуємо час завершення
        end_time = time.time()

            # Перевіряємо, що аналіз завершується не більше ніж за 30
секунд
        # Це прийнятний час очікування для користувача Telegram бота
        assert (end_time - start_time) < 30

    def test_concurrent_requests(self):
        # Тестуємо здатність системи обробляти одночасні запити
        def analyze_wrapper(project):
            # Обгортка для виклику аналізу проекту
            return analyze_project(project)

        # Список популярних токенів для тестування
        projects = ["bitcoin", "ethereum", "cardano"]

          # Використовуємо ThreadPoolExecutor для симуляції одночасних
запитів
        with ThreadPoolExecutor(max_workers=3) as executor:
            # Запускаємо аналіз всіх проектів паралельно
```

```
            futures = [executor.submit(analyze_wrapper, project) for
project in projects]
            # Збираємо результати всіх завдань
            results = [future.result() for future in futures]

        # Перевіряємо, що всі запити завершилися успішно
        assert len(results) == 3
        assert all(result[0] is not None for result in results)
```

Тести продуктивності перевіряють два ключових аспекти роботи системи. Перший тест встановлює максимально допустимий час виконання аналізу, що критично важливо для користувацького досвіду в Telegram-боті. Другий тест перевіряє здатність системи обробляти множинні одночасні запити, що може статися при активному використанні бота кількома користувачами одночасно.

Безпека є критично важливим аспектом для будь-якої системи, що обробляє користувацькі дані, особливо в контексті фінансової інформації. Необхідно забезпечити захист від різних типів атак та некоректного вводу.

Це робиться за допомогою програми `test_security.py`, що має наступний код:

```
import pytest
from analysis import search_coin_id

class TestSecurity:
    @pytest.mark.parametrize("malicious_input", [
        # Тестуємо захист від XSS атак через JavaScript код
        "<script>alert('xss')</script>",
        # Тестуємо захист від SQL ін'єкцій
        "'; DROP TABLE coins; --",
        # Тестуємо захист від path traversal атак
        "../../../etc/passwd",
        # Тестуємо захист від JavaScript псевдо-протоколів
        "javascript:alert('xss')"
    ])
    def test_input_sanitization(self, malicious_input):
        # Тестуємо захист системи від шкідливого вводу
        result = search_coin_id(malicious_input)
        # Очікуємо, що система поверне None або безпечний результат
        # Головне - система не повинна виконувати шкідливий код
        assert result is None or isinstance(result, str)

    def test_api_key_protection(self):
        # Тестуємо захист API ключів від випадкового логування
        import logging
```

```
        from unittest.mock import patch

        # Перехоплюємо всі виклики логування
        with patch('logging.info') as mock_log:
            search_coin_id("test")
            # Отримуємо всі повідомлення, що були залоговані
                        logged_messages = [call.args[0] for call in
mock_log.call_args_list]
            # Перевіряємо, що чутливі дані не потрапили в логи
            for message in logged_messages:
                assert "api_key" not in message.lower()
                assert "token" not in message.lower()
```

Тести безпеки покривають два критичних аспекти захисту системи. Параметризований тест `input_sanitization()` перевіряє стійкість системи до різних типів атак через користувацький ввід. Тест `api_key_protection()` забезпечує, що чутлива інформація, така як ключі API, не потрапляє в логи системи, що могло б призвести до компрометації облікових даних.

Тестування Telegram-бота вимагає особливого підходу через асинхронну природу взаємодії та специфіку обробки повідомлень. Основний фокус робиться на мокуванні bot API та перевірці коректної обробки різних сценаріїв користувацької взаємодії.

Повний код програми тестування `test_bot.py` наведено у додатку Б.

Тести Telegram-бота демонструють три ключові аспекти функціональності. Тест split_message_functionality() перевіряє коректність розбиття довгих аналітичних звітів на частини, що відповідають обмеженням Telegram API. Тест handle_project_success() моделює успішний сценарій обробки користувацького запиту, включаючи виклик аналізу та відправку результатів. Тест handle_project_error() перевіряє graceful degradation системи при виникненні помилок, забезпечуючи можливість отримувати користувачем зрозуміле повідомлення про проблему.

Поданий план тестування дозволяє створити надійну, безпечну та масштабовану систему для аналізу криптопроектів, здатну забезпечити високу

якість сервісу для кінцевих користувачів при мінімальній кількості критичних помилок.

ВИСНОВКИ

Отже, у ході виконання роботи було реалізовано повноцінну систему, здатну надавати комплексний фундаментальний аналіз криптовалютних проєктів у зручному та доступному форматі через месенджер Telegram. Було розроблено Telegram-бота, який автоматично здійснює пошук, перевірку, аналіз та візуалізацію даних про криптопроєкти на основі відкритих джерел CoinGecko, CoinMarketCap та GitHub. Важливою особливістю є інтеграція з API OpenAI, що дозволило здійснювати якісний семантичний аналіз текстових описів проєктів та формувати релевантні експертні висновки.

Завдяки оцінці проєктів за вісьмома критеріями було досягнуто високого рівня структурованості та об’єктивності оцінювання. Генерація діаграм токеноміки додатково візуалізує ключові аспекти структури активів, а обробка повідомлень із урахуванням обмежень Telegram підвищує зручність взаємодії. Розроблений бот працює без потреби у збереженні персональних даних, що відповідає вимогам безпеки.

Протягом виконання роботи було розв’язано наступні завдання:

1) проведено огляд сервісів зі штучним інтелектом для збору та аналізу даних;
2) проведено аналіз API моделей штучного інтелекту;
3) розглянуто використання чатботів у месенджерах;
4) проведено аналіз вимог до розроблюваного чатботу;
5) розроблено проект чатботу та реалізовано його, при цьому, описано основні функції чатботу та аналізу криптовалютного проекту;
6) на основі проведеного огляду методів тестування обрано методи, що можуть бути використані у даному проекті, та розроблено й описано набір тестів, використовуваних для тестування розробленого чатботу.

Таким чином, проєкт довів практичну можливість створення недорогих, надійних та функціональних аналітичних систем, що відкривають доступ до складної фінансової аналітики широкому колу користувачів. Отримані

результати можуть стати основою для подальших досліджень у сфері автоматизації фінансового аналізу, децентралізованих інвестиційних рішень та освітніх платформ з криптовалютної грамотності.

Подальша робота над проектом може полягати у розширенні використовуваних API та збільшенні функціональності розробленого чатботу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Axios: вебсайт. URL: https://www.axios.com/2025/04/09/google-ai-mattel-barbie (дата звернення: 12.03.2025).

2. Amazon SageMaker AI Features: вебсайт. URL: https://aws.amazon.com/sagemaker-ai/features/ (дата звернення: 12.03.2025).

3. Azure AI solutions: вебсайт. URL: https://azure.microsoft.com/en-US/solutions/ai (дата звернення: 12.03.2025).

4. IBM Watson to watsonx: вебсайт. URL: https://www.ibm.com/watson (дата звернення: 13.03.2025).

5. Analytics made intuitive, AI made reliable: вебсайт. URL: https://www.knime.com/ (дата звернення: 13.03.2025).

6. RapidMiner Integration for KNIME: вебсайт. URL: https://hub.knime.com/aborg/extensions/com.mind_era.knime_rapidminer.knime.feature/latest (дата звернення: 14.03.2025).

7. Apache Kafka + KSQL + TensorFlow for Data Scientists via Python + Jupyter Notebook: вебсайт. URL: https://www.datasciencecentral.com/apache-kafka-ksql-tensorflow-for-data-scientists-via-python/ (дата звернення: 14.03.2025).

8. The Impact of AI Visuals: вебсайт. URL: https://www.quanthub.com/the-impact-of-ai-visuals/ (дата звернення: 14.03.2025).

9. Accurate, actionable, decisioning for every digital asset: вебсайт. URL: https://www.elliptic.co (дата звернення: 14.03.2025).

10. The financial world in full focus built on next-gen technology: вебсайт. URL: https://www.bloomberg.com/professional/products/bloomberg-terminal/ (дата звернення: 15.03.2025).

11. TradingView: Look first / Then leap: вебсайт. URL: https://www.tradingview.com (дата звернення: 15.03.2025).

12. Numerai: The hardest data science tournament in the world: вебсайт. URL: https://numer.ai (дата звернення: 17.03.2025).

13. RobinHood: Built for the Future of Trading: вебсайт. URL: https://robinhood.com (дата звернення: 17.03.2025).

14. The world’s most customizable crypto trading bot: вебсайт. URL: https://www.cryptohopper.com (дата звернення: 18.03.2025).

15. Kensho: We illuminate nsights in the world’s data: вебсайт. URL: https://www.kensho.com (дата звернення: 18.03.2025).

16. Від тесту Тьюринга до ChatGPT: що таке чат-боти, для чого їх використовують та яка їхня роль у сучасному світі: вебсайт. URL: https://mc.today/uk/shho-take-chat-boti/ (дата звернення: 18.03.2025).

17. Чат-бот. Дізнайтесь про переваги чат-ботів: вебсайт. URL: https://sendpulse.ua/support/glossary/chatbot (дата звернення: 19.03.2025).

18. Telegram Bot API: вебсайт. URL: https://core.telegram.org/bots/api (дата звернення: 19.03.2025).

19. Bots for Workplace: вебсайт. URL: https://developers.facebook.com/docs/ workplace/integrations/custom-integrations/bots (дата звернення: 19.03.2025).

20. Чат-бот Rakuten Viber: вебсайт. URL: https://www.forbusiness.viber.com/ua/ chatbots/ (дата звернення: 19.03.2025).

21. Trusted WhatsApp messaging at scale: вебсайт. URL: https://www.twilio.com/en-us/messaging/channels/whatsapp (дата звернення: 20.03.2025).

22. An introduction to Slackbot: вебсайт. URL: https://slack.com/help/articles/ 202026038-An-introduction-to-Slackbot (дата звернення: 20.03.2025).

23. Build Where the World Plays: вебсайт. URL: https://discord.com/developers/ docs/intro (дата звернення: 21.03.2025).

24. OpenAI developer platform: вебсайт. URL: https://platform.openai.com/docs/ overview (дата звернення: 21.03.2025).

25. Claude: Your ideas, amplified: вебсайт. URL: https://claude.ai/new (дата звернення: 22.03.2025).

26. Gemini: вебсайт. URL: https://gemini.google.com/app (дата звернення: 22.03.2025).

27. Cohere docs: вебсайт. URL: https://docs.cohere.com/cohere-documentation (дата звернення: 24.03.2025).

28. Mistral AI API: вебсайт. URL: https://docs.mistral.ai/api/ (дата звернення: 24.03.2025).

29. Join the Hugging Face community: вебсайт. URL: https://huggingface.co/docs/inference-providers/index (дата звернення: 24.03.2025).

30. Amazon Bedrock: вебсайт. URL: https://aws.amazon.com/bedrock/ (дата звернення: 25.03.2025).

31. Auger T., Saroyan E. Generative AI for Web Development. Building Web Applications Powered by OpenAI APIs and Next.js. Apress, 2024. 247 p.

32. El Amri A. OpenAI GPT For Python Developers. The art and science of AI-powered apps with GPT-4, Whisper, Weaviate, and beyond. Leanpub, 2024. 323 p.

33. Evelyn L., Hopkins B. Beginning ChatGPT for Python. Build Intelligent Applications with OpenAI API. Apress, 2024. 202 p.

ДОДАТКИ

Додаток А

Повний код чатботу (файл `bot.py`):

```
import os
import telebot
from analysis import analyze_project
from dotenv import load_dotenv
import logging
import textwrap

# Налаштування логування
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)

# Завантаження змінних оточення
load_dotenv()

# Отримуємо токен бота з .env
TOKEN = os.getenv('TELEGRAM_BOT_TOKEN')
if not TOKEN:
    logging.error("TELEGRAM_BOT_TOKEN не знайдено у .env файлі.")
    exit("Необхідно вказати TELEGRAM_BOT_TOKEN у файлі .env")

bot = telebot.TeleBot(TOKEN)

# Функція для розбиття повідомлення на частини за абзацами
def split_message(message, max_length=4000):
    paragraphs = message.split('\n\n')
    messages = []
    current_message = ''
    for paragraph in paragraphs:
        if len(current_message) + len(paragraph) + 2 <= max_length:
            current_message += paragraph + '\n\n'
        else:
            messages.append(current_message)
            current_message = paragraph + '\n\n'
    if current_message:
        messages.append(current_message)
    return messages

# Привітальне повідомлення
@bot.message_handler(commands=['start', 'help'])
def send_welcome(message):
    logging.info(f"Користувач {message.chat.id} запустив бота.")
    bot.reply_to(
        message,
             "Вітаю! Я бот для фундаментального аналізу криптопроектів. Просто
надішліть мені назву проекту, і я підготую для вас детальний аналіз."
    )

# Обробляємо введення від користувача
@bot.message_handler(func=lambda message: True)
def handle_project(message):
    project_name = message.text.strip()
```

```
    user_id = message.chat.id
        logging.info(f"Отримано запит на аналіз проекту '{project_name}' від користувача {user_id}")

    # Надсилаємо сповіщення про початок аналізу
    bot.send_message(user_id, f"Починаю детальний аналіз проекту '{project_name}'. Це може зайняти кілька хвилин, будь ласка, зачекайте...")

    try:
        # Аналіз проекту
        response, tokenomics_image_path = analyze_project(project_name)

        # Надсилаємо текстовий звіт, розбиваючи його на частини, якщо необхідно
        if response:
            messages = split_message(response)
            for idx, msg in enumerate(messages):
                bot.send_message(user_id, msg)
                    # Додаємо невелику затримку між повідомленнями, щоб уникнути обмеження за частотою надсилання
                if idx < len(messages) - 1:
                    time.sleep(1)
        else:
                 bot.send_message(user_id, f"Вибачте, не вдалося отримати дані по проекту '{project_name}'. Можливо, проект не знайдено або сталася помилка.")

        # Надсилаємо зображення токеноміки, якщо воно доступне
        if tokenomics_image_path and os.path.isfile(tokenomics_image_path):
            try:
                with open(tokenomics_image_path, 'rb') as photo:
                       bot.send_photo(user_id, photo, caption=f"Токеноміка проекту '{project_name}'")
                        logging.info(f"Зображення токеноміки для '{project_name}' надіслано користувачу {user_id}")
            except Exception as e:
                    logging.error(f"Помилка при надсиланні зображення токеноміки: {e}")
                      bot.send_message(user_id, f"Не вдалося надіслати зображення токеноміки для проекту '{project_name}'.")
            finally:
                # Видаляємо зображення після надсилання або у випадку помилки
                if os.path.exists(tokenomics_image_path):
                    os.remove(tokenomics_image_path)
        else:
                    logging.warning(f"Зображення токеноміки для '{project_name}' недоступне.")
                   bot.send_message(user_id, f"Зображення токеноміки для проекту '{project_name}' недоступне.")
    except Exception as e:
        logging.error(f"Помилка при обробці проекту '{project_name}': {e}")
            bot.send_message(user_id, "Сталася помилка при аналізі проекту. Будь ласка, спробуйте ще раз пізніше.")

# Запуск бота
if __name__ == '__main__':
    logging.info("Бот запущено і готовий до прийому повідомлень.")
    bot.infinity_polling()
```

Додаток Б

Повний код програми тестування `test_bot.py`.

```
import pytest
from unittest.mock import Mock, patch
from bot import handle_project, split_message

class TestTelegramBot:
    def test_split_message_functionality(self):
        # Тестуємо функцію розбиття довгих повідомлень
        # Створюємо тестове повідомлення, що перевищує ліміт Telegram
        long_message = "Абзац 1\n\n" + "Дуже довгий текст " * 200 + "\n\nАбзац 2"
        # Розбиваємо повідомлення на частини з максимальною довжиною 100 символів
        messages = split_message(long_message, max_length=100)

        # Перевіряємо, що повідомлення дійсно розбилося на кілька частин
        assert len(messages) > 1
        # Перевіряємо, що кожна частина не перевищує встановлений ліміт
        assert all(len(msg) <= 100 for msg in messages)

    @patch('bot.bot.send_message')
    @patch('bot.analyze_project')
    def test_handle_project_success(self, mock_analyze, mock_send):
        # Тестуємо успішний сценарій обробки запиту користувача
        # Мокуємо успішний результат аналізу проекту
        mock_analyze.return_value = ("Test report", None)
        # Створюємо мок об'єкт повідомлення від користувача
        mock_message = Mock()
        mock_message.text = "bitcoin"
        mock_message.chat.id = 12345

        # Викликаємо обробник повідомлення
        handle_project(mock_message)

        # Перевіряємо, що функція аналізу була викликана з правильними параметрами
        mock_analyze.assert_called_once_with("bitcoin")
        # Перевіряємо, що було надіслано принаймні два повідомлення
        # (початкове повідомлення про початок аналізу + результат)
        assert mock_send.call_count >= 2

    @patch('bot.bot.send_message')
    @patch('bot.analyze_project')
    def test_handle_project_error(self, mock_analyze, mock_send):
        # Тестуємо обробку помилки під час аналізу проекту
        # Мокуємо виникнення виключення в процесі аналізу
        mock_analyze.side_effect = Exception("API Error")
        # Створюємо мок повідомлення з неіснуючим токеном
        mock_message = Mock()
        mock_message.text = "invalid_token"
        mock_message.chat.id = 12345

        # Викликаємо обробник повідомлення
        handle_project(mock_message)

        # Перевіряємо, що користувачу було надіслано повідомлення про помилку
        error_calls = [call for call in mock_send.call_args_list
                       if "помилка" in call[0][1].lower()]
        assert len(error_calls) >
```