

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Інтелектуальний аналіз контенту соціальної
мережі Reddit із використанням методів обробки
природної мови та ШІ»**

Виконав студент групи ЗПЗм-24-1
БАШМАНІВСЬКИЙ Максим Олексійович

Керівник роботи:
ЧИЖМОТРЯ Олексій Володимирович

Рецензент:
ЧИЖМОТРЯ Олена Геннадіївна

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

зав. кафедри інженерії

програмного забезпечення

Тетяна ВАКАЛЮК

«18» вересня 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БАШМАНІВСЬКИЙ Максим Олексійович**

Керівник роботи: **ЧИЖМОТРИЯ Олексій Володимирович**

Тема роботи: **«Інтелектуальний аналіз контенту соціальної мережі Reddit із використанням методів обробки природної мови та ШІ»,**

затверджена Наказом закладу вищої освіти від **«18» вересня 2025 р. № 492/с**

Вихідні дані для роботи: **процес автоматизованого збору, попередньої обробки та аналітичної інтерпретації текстового контенту соціальних мереж; методи, алгоритми та програмні засоби інтелектуального аналізу текстів Reddit із використанням локальної мовної моделі штучного інтелекту**

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В	22.09.2025р.	22.09.2025р.
2	Чижмотря О.В	22.09.2025р.	22.09.2025р.
3	Чижмотря О.В	22.09.2025р.	22.09.2025р.

РЕФЕРАТ

Кваліфікаційна робота присвячена розробці інтелектуальної системи автоматизованого аналізу контенту Reddit із використанням методів NLP та локальних LLM. Досліджено підходи до збору та обробки публікацій для виявлення тематичних трендів і емоційних настроїв.

Робота складається з пояснювальної записки та вебзастосунку. Програма реалізує повний цикл обробки: авторизацію, збір даних через Reddit API, NLP-аналіз, генерацію інсайтів локальною мовною моделлю та візуалізацію звітів. Структура записки включає вступ, три розділи, висновки, список літератури та додатки.

Обсяг роботи становить 84 сторінки, містить 13 рисунків, 5 таблиць, 40 посилань та 1 додаток. У першому розділі проаналізовано специфіку контенту Reddit та обґрунтовано вибір технологій NLP і LLM. У другому розділі сформульовано вимоги, спроектовано архітектуру та представлено діаграми системи. У третьому розділі описано реалізацію, результати тестування та оцінку ефективності застосунку.

У висновках підбито підсумки дослідження та окреслено перспективи розвитку. Додатки містять вихідний код основних компонентів.

КЛЮЧОВІ СЛОВА: REDDIT, NLP, LLM, СОЦІАЛЬНА АНАЛІТИКА, ТОНАЛЬНІСТЬ, УЗАГАЛЬНЕННЯ, FIREBASE, NODE.JS, REACT.

ABSTRACT

The qualification work focuses on developing an intelligent system for automated Reddit content analysis using NLP methods and local Large Language Models (LLM). The study explores approaches to processing user posts to identify thematic trends and emotional dynamics.

The work comprises a web application, Reddit Content Analyzer, and an explanatory report. The application implements a full data processing cycle: authentication, data collection via Reddit API, NLP analysis, insight generation via local LLM, and visualization. The report includes an introduction, three chapters, conclusions, references, and appendices.

The thesis volume is 84 pages, including 13 figures, 5 tables, 40 references, and 1 appendix. Chapter 1 analyzes Reddit content specifics and justifies the technology stack. Chapter 2 defines requirements and describes the system architecture. Chapter 3 presents software implementation, testing results, and performance evaluation.

Conclusions summarize the research outcomes and outline future prospects. Appendices contain the source code.

KEYWORDS: REDDIT, NLP, LLM, SOCIAL ANALYTICS, SENTIMENT ANALYSIS, SUMMARIZATION, FIREBASE, NODE.JS, REACT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ДОДАТКУ.....	10
1.1 Постановка задачі.....	10
1.2 Методи та алгоритми аналізу отриманих даних.....	11
1.3 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	14
1.4 Огляд наявних методів, моделей, алгоритмів.....	16
1.5 Огляд аналогів та архітектура застосунку.....	20
Висновки до розділу 1.....	25
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ КОНТЕНТУ.....	27
2.1 Моделювання функціональних вимог та сценаріїв взаємодії.....	27
2.2 Архітектура програмного комплексу та інфраструктура розгортання	35
2.3 Програмна реалізація компонентів аналізу та обробки даних.....	42
Висновки до розділу 2.....	46
РОЗДІЛ 3. ТЕСТУВАННЯ ТА РОБОТА З СИСТЕМОЮ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ КОНТЕНТУ.....	48
3.1 Опис реалізації автоматичного тестування.....	48
3.2 Розгортання застосунку.....	51
3.3 Мануальне тестування.....	53
3.4 Демонстрація роботи та практичне використання програмного комплексу.....	56
Висновки до розділу 3.....	66
ВИСНОВКИ.....	67
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	69
ДОДАТКИ.....	74

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

NLP - обробка природної мови (Natural Language Processing)

LLM - велика мовна модель (Large Language Model)

API - програмний інтерфейс взаємодії застосунків (Application Programming Interface)

TF-IDF - статистичний показник важливості терма у документі

OAuth2 - протокол авторизації та доступу до ресурсів

UI/UX - користувацький інтерфейс та досвід взаємодії

ВСТУП

Темою даної кваліфікаційної роботи магістра є розробка системи інтелектуального аналізу контенту соціальної мережі Reddit із використанням методів обробки природної мови та ШІ.

Актуальність теми. У сучасному інформаційному просторі соціальні медіа стали одним із найважливіших джерел публічних висловлювань, відображаючи суспільні настрої, інтереси, цінності та тенденції. Платформа Reddit є одним із наймасштабніших комунікативних середовищ, що акумулює значні обсяги текстового контенту, створеного користувачами у форматі постів і коментарів. Цей контент становить цінне джерело для соціологічних, психологічних та лінгвістичних досліджень, однак його аналіз ускладнюється великими обсягами даних, високою динамічністю оновлень і неструктурованим характером текстів.

В умовах зростання ролі штучного інтелекту особливої актуальності набувають інтелектуальні системи, здатні автоматично збирати, обробляти й інтерпретувати дані соціальних мереж. Поєднання сучасних технологій обробки природної мови (NLP) з машинним навчанням відкриває можливість створення інструментів, які забезпечують глибоке аналітичне осмислення контенту, виявлення закономірностей комунікації та емоційних тенденцій. Таким чином, розроблення системи інтелектуального аналізу контенту Reddit є актуальним завданням, що поєднує наукові й практичні аспекти сучасної інформаційної аналітики.

Об'єктом дослідження є процес автоматизованого збору, попередньої обробки та аналітичної інтерпретації текстового контенту соціальних мереж.

Предметом дослідження є методи, алгоритми та програмні засоби інтелектуального аналізу текстів Reddit із використанням локальної мовної моделі штучного інтелекту.

Метою дослідження є розроблення інтелектуальної системи аналізу контенту Reddit, яка здійснює повний цикл обробки даних - від авторизації

користувача та збору публікацій через офіційний Reddit API до AI-аналізу текстів, візуалізації результатів та їх збереження у хмарному середовищі.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- Спроекувати комплексну архітектуру системи, що включає модулі аутентифікації користувача та взаємодії із зовнішніми джерелами даних.
- Реалізувати програмний інтерфейс для інтеграції з Reddit API з метою автоматизованого збору постів та коментарів.
- Розробити алгоритми попередньої обробки текстових даних, що забезпечують їх очищення, лематизацію, а також виявлення дублікатів і спам-повідомлень.
- Імплементувати методи інтелектуального аналізу на основі мовної моделі для класифікації тем, визначення тональності тексту, узагальнення змісту дискусій та детекції токсичності.
- Забезпечити збереження оброблених даних та результатів аналізу у хмарній базі даних Firebase Firestore, а також реалізувати механізми формування й візуалізації аналітичних звітів.

Наукова новизна роботи полягає у створенні інтегрованої програмної системи, яка поєднує сучасні методи обробки природної мови, машинного навчання та вебаналітики в єдиному середовищі. На відміну від традиційних підходів до контент-аналізу, запропонована модель використовує локальну мовну модель chatgpt-oss, що дозволяє зберігати конфіденційність даних і забезпечує можливість автономного виконання AI-обчислень без залежності від зовнішніх API. Запропонована архітектура передбачає модульність і масштабованість, що створює передумови для адаптації системи під інші джерела контенту, такі як Twitter, YouTube або Telegram.

Практичне значення одержаних результатів полягає у створенні прикладного інструменту для аналітиків, дослідників і фахівців із цифрових комунікацій, який дозволяє оперативно оцінювати зміст та емоційне забарвлення дискусій у соціальних мережах. Розроблена система може бути

використана для моніторингу громадських настроїв, оцінки ефективності комунікаційних кампаній, виявлення проблемних тематик та інформаційних ризиків. Завдяки зручній аналітичній панелі користувач має змогу візуалізувати результати аналізу у вигляді графіків, розподілів та коротких узагальнень (TL;DR), що підвищує ефективність роботи з великими обсягами текстових даних.

Апробація. За темою кваліфікаційної роботи було опубліковано 2 тези конференцій [1, 2].

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ДОДАТКУ

1.1 Постановка задачі

У сучасних умовах функціонування цифрового середовища соціальні мережі виступають ключовими платформами для обміну інформацією, формування громадської думки та комунікації між користувачами. Платформа Reddit, як одна з найпопулярніших дискусійних спільнот у світі, містить величезний масив текстового контенту, який постійно оновлюється. Цей контент охоплює широкий спектр тематик - від наукових і технічних до соціальних, культурних і політичних. Аналіз таких даних може дати цінні інсайти щодо тенденцій у суспільстві, динаміки емоційного фону користувачів і характеру онлайн-дискусій.

Основною проблемою, що постає під час обробки даних Reddit, є їхній неструктурований характер, великий обсяг та різноманітність мовних форм [3]. Ручний аналіз такого контенту є неефективним, трудомістким і схильним до суб'єктивності. Тому постає необхідність у створенні автоматизованої системи, здатної здійснювати збір, попередню обробку, аналітичну інтерпретацію та візуалізацію результатів у зручному вигляді.

У межах кваліфікаційної роботи передбачається розроблення системи, яка забезпечить повний цикл обробки даних соціальної мережі Reddit - від моменту отримання публікацій до формування підсумкових аналітичних висновків. Для цього потрібно реалізувати низку взаємопов'язаних завдань. По-перше, необхідно забезпечити автентифікацію користувача, інтеграцію з офіційним Reddit API та можливість вибору тематичних розділів або хештегів для збору даних. По-друге, слід реалізувати модуль попередньої обробки контенту, який передбачає очищення тексту від шумів, нормалізацію, лематизацію, визначення мови повідомлень та видалення дубльованих або спам-записів.

Наступним завданням є створення аналітичного модуля на основі мовної моделі штучного інтелекту chatgpt-oss, який здійснює класифікацію тем, визначення емоційної тональності, виділення ключових фраз і сутностей, а також виявлення токсичних висловлювань. Отримані результати повинні зберігатися у хмарному середовищі Firebase Firestore з можливістю формування історії попередніх запитів та повторного аналізу даних. Важливою складовою є реалізація аналітичної панелі, яка відображатиме результати у вигляді інтерактивних графіків, статистичних розподілів та коротких текстових узагальнень (TL;DR).

Таким чином, основна задача дослідження полягає у створенні інтелектуальної інформаційної системи, що забезпечує автоматизований аналіз контенту Reddit з використанням технологій штучного інтелекту та обробки природної мови. Система повинна бути здатна ефективно працювати з великими обсягами даних, забезпечувати точність аналітичних висновків, зберігати результати для подальшої інтерпретації та підтримувати розширюваність архітектури для майбутньої інтеграції з іншими соціальними платформами.

1.2 Методи та алгоритми аналізу отриманих даних

Методологія аналізу у запропонованому застосунку ґрунтується на послідовному конвеєрі оброблення тексту, що забезпечує відтворюваність результатів та узгодженість характеристик даних на всіх етапах. Після отримання публікацій і коментарів з офіційного Reddit API виконується нормалізація вхідних документів: усуваються HTML-артефакти, емодзі та службові символи, вирівнюється регістр, зберігаються структурні маркери цитувань та посилань. Для багатомовного середовища застосовується визначення мови на рівні документа або речення з подальшим вибором відповідного токенізатора та лематизатора; це зменшує варіативність словоформ і стабілізує частотні розподіли. Дедуплікація виконується

комбіновано: спочатку використовуються хеші вмісту для суворих дублікатів, далі - порівняння векторних подань із косинусною мірою та пороговою фільтрацією майже ідентичних текстів. Для зниження впливу спаму та низькоінформативних записів застосовуються евристики за довжиною, часткою посилань та повторюваністю шаблонних фрагментів, а також швидкий класифікатор «спам/не спам», навчений на узагальнених ознаках.

Подання тексту формується у двох комплементарних просторах ознак. Для прозорості статистичної аналітики використовується TF-IDF для n-грам із нормуванням довжини документа; це дозволяє коректно будувати частотні профілі, колокації та коефіцієнти асоціації. Для семантичних задач створюються контекстуальні ембеддинги на базі трансформерної моделі [35, 37]; вони забезпечують інваріантність до формальних варіацій та краще відображають значення лексем у різних контекстах. Векторні подання застосовуються до подальших модулів класифікації, кластеризації та пошуку схожих фрагментів.

Тематику матеріалів визначають двома механізмами залежно від режиму роботи. У режимі без учителя виконується кластеризація векторних подань із подальшим підписуванням тем за топ-термами та репрезентативними документами, що забезпечує відкриття латентних напрямів обговорень. У наглянутому режимі використовується тонке донавчання трансформерної моделі на позначених прикладах; у випадку мульти-лейбл постановки оптимізується сума бінарних крос-ентропій із калібруванням порогів за метриками F1 (macro/micro). Це дозволяє коректно враховувати перетини тем і дисбаланс класів.

Аналіз тональності реалізовано як трирівневу класифікацію з урахуванням нейтральної зони [10]. Базовим провайдером ознак є контекстуальні ембеддинги, поверх яких тренується лінійний або невеликий двошаровий класифікатор з регуляризацією. Для підвищення стійкості до сарказму й заперечень використовуються аугментації тексту та вагове штрафування за прикладами із суперечливими сигналами. Додатково

застосовується калібрування ймовірностей (Platt/temperature scaling), що забезпечує інтерпретованість прогнозів на рівні довіри та коректну агрегацію в часових розрізах.

Виявлення токсичності та мовної агресії розглядається як багатокласова задача із градаціями порушень. З метою зниження хибнопозитивних спрацьовувань використовується двоступеневий конвеєр: спочатку легка модель відсіює явно нетоксичні приклади з високою впевненістю, після чого складні випадки обробляє тонко донавчена трансформерна модель. Порогові рішення узгоджуються через оптимізацію ROC-AUC та PR-AUC на валідаційній вибірці, а кінцеві мітки стабілізуються методом рухомого середнього в межах гілок дискусії.

Узагальнення дискусій (TL;DR) будується в абстрактивній парадигмі з попереднім структурним стисканням дерева коментарів [13]. Спочатку формується «доказова канва» через селекцію фрагментів із високою центральністю та різноманітністю, далі ця канва подається в генеративну модель із обмеженням довжини та пеналізацією повторів. Для великих гілок застосовується ієрархічне резюмування: локальні підсумки на рівні підгілок конденсуються в глобальний TL;DR, що зменшує ризик втрати ключових тез і підвищує фактичну відповідність.

Виділення ключових фраз і сутностей реалізовано комбіновано. На першому етапі ранжуються кандидатні фрази за графовими та частотними критеріями, на другому - виконується уточнювальне послідовне маркування трансформерною моделлю. Післяпроцесинг включає нормалізацію варіантів написання, згортання синонімів та лінування до канонічних сутностей, що підвищує стабільність аналітики в часі та дозволяє коректно агрегувати показники.

Статистична аналітика охоплює побудову розподілів за мовами, авторами та сабреддітами, оцінки лексичної насиченості й різноманітності, а також часовий аналіз. Для останнього застосовується згладжування ковзними вікнами, сезонна декомпозиція та робастні метрики, стійкі до віральних

сплесків. Порівняння тематичних і тональних профілів виконується у векторному просторі за косинусною подібністю; відстані транслюються у зміни трендів, що відображаються на аналітичній панелі.

Забезпечення якості охоплює контроль зсуву даних і періодичне оновлення порогів. Внутрішні метрики включають ассурасу, macro/micro-F1, ROC-AUC і калібровану Brier-міру, зовнішні - узгодженість із ручними експертними оцінками на підвибірках. Для операційної надійності використовується асинхронне опрацювання задач із повторними спробами при мережевих збоях і перевищенні лімітів API, кешування проміжних артефактів та ідемпотентні транзакції з маркерами стану, що запобігає дублюванню під час відновлення.

Результати аналізу зберігаються у документно-орієнтованій структурі з чіткими схемами метаданих, що забезпечує трасованість від початкового документа до агрегованих показників. На клієнтській стороні вони подаються у вигляді інтерактивних візуалізацій і текстових резюме, причому всі числові індикатори супроводжуються інтервалами довіри або каліброваними ймовірностями, що дозволяє приймати рішення на основі достовірних, відтворюваних оцінок. Сукупність описаних методів формує узгоджений конвеєр «очищення - подання - інтерпретація - агрегування - валідація», придатний для масштабування на інші джерела соціального контенту без зміни загальних принципів роботи.

1.3 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Розробка вебзастосунку для інтелектуального аналізу контенту Reddit вимагає поєднання високої продуктивності, гнучкості архітектури та простоти інтеграції з зовнішніми API й бібліотеками машинного навчання. З огляду на ці вимоги, для реалізації системи було обрано JavaScript-стек із використанням технологій Node.js для серверної частини, Express.js як фреймворку для REST-

API, а також React.js для клієнтського інтерфейсу [15, 16]. Такий вибір забезпечує узгодженість середовища розроблення, зручну роботу з асинхронними процесами, просту масштабованість і сумісність з інструментами штучного інтелекту, які можуть бути розгорнуті локально або через API-шлюзи.

Серверна частина системи базується на Node.js, що є подієво-орієнтованим середовищем виконання JavaScript-коду поза браузером. Воно забезпечує ефективне оброблення одночасних запитів, що є важливою умовою при зборі даних із Reddit API та під час асинхронної AI-обробки. Для маршрутизації запитів, керування чергами обробки, кешування результатів і комунікації з базою даних використовується фреймворк Express.js, який дозволяє швидко створювати гнучку REST-архітектуру та розширювати її додатковими модулями.

Клієнтська частина розробляється на основі React.js, що дає змогу створити інтерфейс користувача з високою швидкодією, підтримкою компонентного підходу та можливістю інтерактивного оновлення даних без перезавантаження сторінки. Це особливо важливо для реалізації аналітичної панелі, де необхідно візуалізувати результати аналізу у вигляді графіків, діаграм і таблиць. Для побудови візуалізацій планується використання бібліотек Chart.js або Recharts, які мають широкий спектр інтерактивних елементів і легко інтегруються з React-компонентами.

Для зберігання даних обрано Firebase Firestore, який забезпечує хмарне збереження результатів аналізу, історії запитів користувача та налаштувань застосунку. Крім того, у проєкті використовується Firebase Authentication для безпечної авторизації користувачів за допомогою електронної пошти або зовнішніх сервісів OAuth2 (Google, GitHub тощо).

AI-аналіз контенту виконується на базі локальної мовної моделі chatgpt-oss, інтегрованої через серверний модуль Node.js. Такий підхід дозволяє уникнути залежності від комерційних API-сервісів, забезпечити

конфіденційність оброблюваних даних і пришвидшити обчислення завдяки локальному виконанню.

Загалом, обраний стек технологій забезпечує зручну взаємодію між клієнтською, серверною та аналітичною частинами системи, а також простоту розгортання у хмарному середовищі або на локальному сервері.

Вимоги до апаратного забезпечення визначаються потребами у швидкому обробленні текстових даних і виконанні AI-моделі. Для локального розгортання системи достатньо персонального комп'ютера з процесором Intel Core i5 / AMD Ryzen 5 або вище, не менше 16 ГБ оперативної пам'яті та дискового простору 20–30 ГБ. За наявності GPU (наприклад, NVIDIA RTX 3060 або вище) можлива апаратна акселерація AI-модуля, що значно зменшує час обробки даних. Для віддаленого розгортання можуть використовуватися хмарні платформи типу Google Cloud, Render або Vercel, які підтримують Node.js-застосунки та Firebase-інтеграцію.

Таким чином, вибір інструментальних засобів обґрунтовано їхньою стабільністю, поширеністю, сумісністю з технологіями штучного інтелекту та зручністю реалізації вебаналітичних систем. Використання JavaScript-стеку дозволяє досягти високого рівня інтеграції між компонентами системи та забезпечує можливість подальшого розширення функціоналу без зміни основної архітектури.

1.4 Огляд наявних методів, моделей, алгоритмів

Сучасна практика інтелектуального аналізу контенту соціальних мереж спирається на поєднання класичних підходів обробки природної мови та глибинних нейронних моделей [25, 28]. Базовим етапом є попередня обробка тексту, яка включає нормалізацію регістру, очищення HTML-розмітки й емодзі, усунення артефактів кодування, токенізацію та лематизацію. Для багатомовних колекцій, характерних для Reddit, застосовуються алгоритми визначення мови документа або речення з подальшим вибором

мовоспецифічних правил лематизації й наборів стоп-слів. Якість цього етапу безпосередньо впливає на стабільність ознак і знижує варіативність словоформ, що є критично важливим для статистичних методів та моделей векторного подання.

Представлення тексту у вигляді числових ознак відбувається двома основними сімействами підходів. Класичні моделі ознак використовують вагові схеми TF-IDF для уніграм і n-грам, інколи з частотним обрізанням і нормалізацією довжини документа; вони лишаються конкурентоспроможними для коротких повідомлень, де контекст обмежений. Семантичні векторні подання охоплюють незконтекстуальні ембеддинги на основі спільної появи (word2vec, GloVe, fastText) та контекстуальні трансформерні представлення, у яких значення слова залежить від оточення [26, 30]. Саме контекстуальні ембеддинги слугують стандартом де-факто для задач класифікації, виділення сутностей, узагальнення й перенесення знань між доменами [12, 27]. Для коротких, іронічних чи меметичних текстів додатково доцільно враховувати символні або субсловні ознаки, що підвищує стійкість до помилок і творчих варіацій правопису.

Класифікація тем реалізується двома лініями підходів. У моделях без нагляду використовуються латентно-тематичні розкладення на кшталт LDA чи NMF для відкриття схованої тематичної структури [40], а також більш сучасні методи, що поєднують контекстуальні ембеддинги з кластеризацією і подальшою інтерпретацією тем (наприклад, підходи типу BERTopic). У наглянутих сценаріях застосовується тонке донавчання трансформерних архітектур на розмічених корпусах або навчання класичних класифікаторів (логістична регресія, SVM, градієнтний бустинг) поверх стійких ознак TF-IDF чи фіксованих ембеддингів [18, 32]. Умовою практичної придатності є механізми боротьби з дисбалансом класів і мульти-лейбл постановками, характерними для кількох одночасних тем у дописі.

Аналіз тональності історично починався з лексикон-орієнтованих методів, що узагальнюють полярності словників і коригують інверсії через

заперечення [22]. Ці підходи зручні через прозорість і малу потребу в обчислювальних ресурсах, але поступаються сучасним моделям, що враховують контекст, сарказм і міжреченнєві залежності. У практиці соціальних мереж найкращі результати дає донавчання трансформерів на доменно-специфічних вибірках [11, 24], у тім числі з багатомовною підтримкою та переносом знань між спорідненими субреддітами.

Виявлення токсичності й мовної агресії розглядається як бінарна або багатокласова класифікація з урахуванням різновидів некоректних висловлювань. Проблемою є тонкі форми образ, евфемізми, іронія та креативні шляхи обходу фільтрів. Тому поряд із донавчанням великих мовних моделей доцільно застосовувати стратегії контрастивного навчання й методи аугментації, що розширюють покриття рідкісних патернів [23]. Для зниження хибнопозитивних спрацьовувань використовуються порогові ансамблі й калібрування ймовірностей, а також двоетапні конвеєри: спочатку грубе фільтрування, потім уточнювальна модель.

Узагальнення дискусій представлено двома парадигмами. Екстрактивні методи (TextRank, LexRank і споріднені графові підходи) формують TL;DR через вибір інформативних речень, забезпечуючи відтворюваність і контрольованість, проте часто втрачають зв'язність. Абстрактивні підходи, реалізовані трансформерними seq2seq-архітектурами, генерують стислі перефразування зі збереженням ключових смислів і доцільні для довгих тредів [4, 5, 21]. Для Reddit-структур із вкладеними коментарями результативною є стратегія стискання дерев обговорення: агрегування доказів на рівні гілок, нормалізація цитат і усунення повторів перед подачею в генератор, що зменшує галюцинації й покращує фактичну відповідність.

Виділення сутностей і ключових фраз розв'язується як послідовне маркування або ранжування фрагментів. Класичні CRF-моделі, підсилені символічними ознаками, залишаються ефективними на ресурсно-обмежених налаштуваннях, проте найвищу якість забезпечують трансформери з тонким донавчанням на анотованих даних домену [33]. Для користувацько-

генерованих текстів важливим є післяпроцесинг: нормалізація імен власних, згортання варіантів написання, групування псевдонімів і лінкування до канонічних сутностей.

Генерація ознак і моделювання тем підсилюються статистичною аналітикою частот, колокацій і n-грам, оцінками різноманітності та лексичної насиченості. Такі індикатори покращують інтерпретованість моделей і служать основою для візуалізації динаміки тем і настроїв у часі. Для часових рядів контенту вживаються згладжування ковзним вікном, сезонна декомпозиція та робастні метрики, стійкі до «спайків», спричинених віральними подіями або рейд-активністю.

З огляду на деревоподібну структуру коментарів Reddit, доцільними є графові уявлення дискусій, де вузли відповідають повідомленням, а ребра - відношенням відповіді. Такий підхід відкриває можливості для метрик центральності, виявлення поляризації та кластеризації за підгілками. У поєднанні з текстовими ембеддингами графові ознаки дозволяють оцінювати вплив користувачів і стабільність тем упродовж часу, а також підвищують якість агрегованих TL;DR.

Важливою складовою виробничих конвеєрів є керування потоками обробки. Для асинхронного AI-аналізу застосовуються черги завдань і механізми back-pressure, кешування проміжних артефактів та ідемпотентні повтори при помилках мережі або перевищенні лімітів API. Якість моделей контролюється через розщеплення вибірки, крос-валідацію та моніторинг зсуву даних; для багатомовних корпусів додається оцінка за підкорпусами, що запобігає завищенню метрик через домінування однієї мови.

У контексті локальних мовних моделей особливого значення набувають інструкційне донавчання і налаштування під задачі узагальнення, тональності та токсичності [29, 36]. Це дозволяє скористатися перевагами автономного розгортання (конфіденційність, передбачувана затримка) і водночас зберегти високу якість завдяки доменно-специфічним даним. Практика поєднання великих універсальних ембеддингів для пошуку

релевантних фрагментів із подальшою «короткою» генерацією резюме дає збалансований компроміс між витратами та точністю.

Зведено, сучасний арсенал методів охоплює надійні класичні техніки підготовки тексту і побудови ознак, а також трансформерні моделі для глибинного розуміння контексту. Для задач, поставлених у цьому проєкті, оптимальним є конвеєр, де очищений і нормалізований текст подається на контекстуальні ембеддинги, що живлять класифікацію тем і тональності, виявлення токсичності та виділення сутностей, тоді як узагальнення дискусій виконується абстрактивною моделлю після структурного стискання дерева коментарів. Така комбінація забезпечує високу якість інтерпретації, стійкість до шуму та масштабованість.

1.5 Огляд аналогів та архітектура застосунку

Розвиток технологій обробки природної мови та соціальної аналітики сприяв появі великої кількості сервісів, які здійснюють автоматизований збір і аналіз текстового контенту. До таких систем належать як комерційні продукти, орієнтовані на моніторинг соціальних мереж, так і дослідницькі інструменти, створені для академічних цілей. Огляд існуючих рішень дає змогу визначити рівень розвитку галузі, оцінити наявні технічні підходи та виявити недоліки, які потребують удосконалення.

Одним із найвідоміших комерційних рішень є платформа Brandwatch, яка надає можливість моніторингу соціальних мереж, новинних ресурсів і блогів із використанням алгоритмів машинного навчання. Система забезпечує класифікацію тем, аналіз тональності та виявлення трендів у соціальних дискусіях. Проте її функціонал орієнтований переважно на корпоративних клієнтів і потребує платної підписки, що обмежує доступність для індивідуальних дослідників і розробників. Крім того, вона не надає відкритого API для користувацького розширення чи глибокої інтеграції з іншими аналітичними модулями.

Іншим прикладом є аналітична система Crimson Hexagon (нині Brandwatch Consumer Research), яка використовує методи штучного інтелекту для визначення емоційного фону публікацій у соціальних мережах. Система надає зручні засоби візуалізації та тематичної класифікації контенту, але її використання обмежене закритим програмним середовищем і неможливістю локального розгортання. Для академічних досліджень та гнучкої адаптації під специфічні потреби така архітектура є недостатньо гнучкою.

Серед відкритих дослідницьких рішень варто виокремити платформу Google BigQuery + Reddit Dataset, яка дозволяє отримувати великі обсяги історичних даних Reddit для статистичного аналізу. Цей підхід підходить для масових досліджень, проте не забезпечує автоматичного збору актуального контенту, попередньої текстової обробки чи інтерпретації результатів. Користувачеві необхідно самостійно реалізовувати механізми фільтрації, лематизації та тематичної класифікації, що ускладнює практичне використання без значної технічної підготовки.

Варто також відзначити онлайн-сервіси MonkeyLearn, MeaningCloud та Hugging Face Spaces, які надають можливість аналізу текстів через API із використанням попередньо навчених моделей. Вони виконують класифікацію тем і визначення тональності, проте не підтримують глибоку інтеграцію з Reddit, не зберігають історію користувацьких запитів і не надають розширених інструментів візуалізації результатів. Крім того, більшість безкоштовних тарифів мають суворі обмеження на кількість запитів і обсяг оброблюваних даних.

Деякі дослідницькі проєкти, наприклад Reddit Sentiment Analysis Dashboard (GitHub, open-source), реалізують базові можливості збору даних Reddit і визначення емоційного забарвлення текстів за допомогою моделей на кшталт VADER або BERT [19]. Такі рішення демонструють перспективність застосування технологій обробки природної мови, однак здебільшого не мають масштабованої архітектури, не реалізують повний цикл обробки даних

(від авторизації користувача до візуалізації) і не підтримують персоналізовану роботу з результатами.

На відміну від наведених аналогів відображених у таблиці 1.1, розроблювана система поєднує гнучкість відкритих технологій і зручність готового користувацького інтерфейсу. Її особливістю є використання локальної мовної моделі chatgpt-oss, що дозволяє здійснювати AI-аналіз без передачі даних на сторонні сервери, зберігаючи конфіденційність інформації [7, 8, 34]. Система реалізує повний цикл обробки – від збору даних через офіційний Reddit API (OAuth2) до формування аналітичних звітів, узагальнень (TL;DR) і збереження результатів у Firebase Firestore.

Таким чином, проведений огляд показує, що попри наявність великої кількості інструментів для соціальної аналітики, жоден із них не поєднує у собі локальну AI-обробку, персоналізоване збереження даних і гнучку архітектуру для розширення під різні джерела контенту. Це підтверджує доцільність і новизну створення власної системи інтелектуального аналізу контенту Reddit, орієнтованої на відкритість, адаптивність і доступність для дослідників.

Таблиця 1.1

Порівняння наведених аналогів

Критерій	Brandwatch	Crimson Hexagon	Google BigQuery + Reddit Dataset	MonkeyLearn / MeaningCloud	Reddit Sentiment Analysis Dashboard	Розроблювана система
Тип рішення	Комерційна платформа	Комерційна система з AI-аналітикою	Аналітичний набір даних (Big Data)	API-сервіси NLP	Відкрите програмне забезпечення (open-source)	Вебзастосунок з локальною AI-моделлю
Підтримка Reddit API	Ні (опосередковано через соцмережі)	Ні	Так (обмежено, історичні дані)	Ні	Так (через офіційний Reddit API)	Так (повна інтеграція з Reddit API OAuth2)

Продовження табл. 1.1

Можливості AI / NLP аналізу	Класифікація тем, тональність, тренди	Емоційний аналіз, семантичні моделі	Лише статистичний аналіз	Класифікація, тональність, ключові слова	VADER / BERT - базовий аналіз тональності	Повний AI-аналіз (тональність, теми, TL;DR, токсичність)
Візуалізація результатів	Так (корпоративна аналітика)	Так (дашборди, графіки)	Так, але технічна (SQL / BI)	Обмежена (через API)	Проста (HTML / графіки)	Інтерактивна панель (React + Chart.js / Recharts)
Персоналізація користувача	Так, корпоративні акаунти	Так, у межах підписки	Ні	Ні	Обмежена	Так, через Firebase Authentication
Збереження історії аналізів	Так (у корпоративному акаунті)	Так (у межах сервісу)	Ні	Ні	Ні	Так (Firebase Firestore, за користувачем і темою)
Можливість локального розгортання	Ні	Ні	Так (залежить від середовища)	Ні	Так	Так (Node.js + локальна LLM chatgpt-oss)
Основні недоліки	Закрите API, висока вартість	Обмежена гнучкість, платна ліцензія	Відсутність AI-обробки, складність використання	Обмеження запитів, немає збереження даних	Немає персоналізації, обмежений функціонал	Потребує налаштування середовища, локальної моделі

Архітектура вебзастосунку побудована за багаторівневим принципом і орієнтована на повний цикл оброблення контенту: від ініціації запиту користувачем до збереження аналітичних результатів і їх візуалізації. Логічно система поділяється на клієнтський рівень, серверний рівень оркестрації, аналітичний рівень із локальною мовною моделлю та рівень даних. Така

декомпозиція мінімізує зв'язність між компонентами, підвищує надійність і дозволяє масштабувати кожен шар незалежно від інших.

Клієнтський рівень реалізований як односторінковий інтерфейс на основі React. Він забезпечує автентифікацію користувача, формування параметрів аналізу (вибір сабреддів, хештегів або конкретних URL дискусій), керування історією попередніх запитів і взаємодію з аналітичною панеллю. Обмін даними з сервером здійснюється через REST-інтерфейс, а стан інтерфейсу синхронізується з результатами, що зберігаються у хмарній базі даних.

Серверний рівень побудований на Node.js з використанням Express. Його призначення полягає у прийманні клієнтських запитів, керуванні OAuth2-взаємодією з офіційним Reddit API, нормуванні частоти запитів, кешуванні і валідації вхідних параметрів. Сервер здійснює оркестрацію конвеєра обробки: передає отримані дані в модуль попередньої обробки, ініціює виконання інтелектуального аналізу в локальній моделі, формує підсумкові структури результатів і записує їх до сховища. Для ізоляції довгих операцій використовується асинхронна постановка задач у чергу, що усуває блокування запитів і підтримує прогнозовану затримку інтерфейсу.

Аналітичний рівень поєднує модуль попередньої обробки текстів і локальний AI-двигун. На етапі попередньої обробки виконуються очищення від HTML і зайвих символів, мововизначення, токенізація та лематизація, видалення стоп-слів, фільтрація дублікатів і спаму. Далі нормалізований корпус надходить до локальної мовної моделі chatgpt-oss [39], яка здійснює класифікацію тем, оцінювання тональності, узагальнення дискусій, виділення ключових фраз і сутностей та виявлення токсичності. Результати агрегуються у вигляді компактних записів зі збереженням основних метаданих, що забезпечує відтворюваність і можливість повторного аналізу.

Рівень даних реалізовано у Firebase Firestore як документно-орієнтоване сховище з прив'язкою записів до ідентифікатора користувача, теми запиту та часових міток. Автентифікація користувачів виконується через Firebase

Authentication, що забезпечує безпечне зберігання сесій і контроль доступу на рівні правил бази даних. Для прискорення повторних запитів застосовується кеш проміжних результатів на сервері; у розгортаннях із підвищеним навантаженням цей кеш доцільно винести в окремий ін-меморі сервіс. Обмін між компонентами відбувається через чітко визначені контракти у форматі JSON, що спрощує тестування, логування та діагностику.

Потік даних має конвеєрну структуру. Користувач формує запит і проходить автентифікацію, після чого сервер виконує збір контенту в межах політики rate-limit, нормалізує дані, передає їх на інтелектуальну обробку, зберігає результати та повертає посилання на готові об'єкти для візуалізації. Аналітична панель відображає агреговані показники, розподіли та короткі резюме, а також надає доступ до деталей окремих постів і гілок коментарів. Така логіка підтримує відмовостійкість (завдяки ідемпотентним операціям і повторним спробам), горизонтальне масштабування (через stateless-бекенд і черги задач) та еволюцію моделі AI без зміни зовнішніх інтерфейсів.

Висновки до розділу 1

У межах першого розділу здійснено всебічний теоретико-аналітичний огляд проблематики інтелектуальної обробки соціального контенту та сформовано концептуальне підґрунтя для розроблення вебзастосунку Reddit Content Analyzer. Встановлено, що сучасні інформаційні потоки, зокрема у середовищах типу Reddit, характеризуються високим рівнем динамічності, неструктурованості та семантичної варіативності, що унеможлиблює їх ефективну інтерпретацію традиційними аналітичними методами. Це актуалізує потребу у створенні спеціалізованих систем, здатних поєднувати алгоритми попередньої обробки тексту, статистичні NLP-показники та генеративні можливості великих мовних моделей.

Розглянуті методи нормалізації, токенизації, лематизації, аналізу лексичної різноманітності, кластеризації тем та оцінки тональності дозволили

визначити оптимальну структуру аналітичного конвеєра, що забезпечує якісну підготовку даних до високорівневої інтерпретації. Детальне вивчення сучасних трансформерних моделей і технологій абстрактного узагальнення дало змогу окреслити можливість побудови пояснювального шару, здатного не лише відображати фактичні індикатори, а й формувати аналітичні висновки та змістовні резюме дискусій.

Проведений огляд аналогів, серед яких комерційні платформи (Brandwatch, Crimson Hexagon), прикладні API-рішення (MonkeyLearn, MeaningCloud) та відкриті проєкти (Reddit Sentiment Dashboard), засвідчив відсутність систем, які б повністю реалізовували весь цикл аналізу - від збору контенту до генерації інтерпретованих висновків із підтримкою персоналізації, локального розгортання та автономної LLM-інтеграції. Таким чином, обґрунтовано наукову та практичну доцільність створення нового застосунку, який об'єднує повноцінний бекенд, механізми AI-аналітики та інтерактивний інтерфейс користувача.

Окремо доведено, що вибір технологічного стеку (Node.js + Express, React, Firebase, локальна модель chatgpt-oss або openai/gpt-oss-20b) забезпечує гнучкість, модульність та масштабованість системи, дозволяючи адаптувати її не лише до Reddit, а й до інших соціальних джерел. Визначені апаратні вимоги підтвердили реалізованість системи як у локальному середовищі дослідника, так і в інфраструктурі розгортання для виробничого використання.

Таким чином, у результаті виконання першого розділу сформовано чіткий науково-методичний фундамент проєкту: визначено проблему, окреслено алгоритмічну базу, проаналізовано існуючі інструменти, обґрунтовано вибір технологій і сформульовано концептуальну модель майбутньої системи. Це створює передумови для переходу до проєктування архітектури, декомпозиції модулів і реалізації ключових компонентів, які будуть розглянуті у наступному розділі.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ КОНТЕНТУ

2.1 Моделювання функціональних вимог та сценаріїв взаємодії

У цьому підрозділі визначено ключові вимоги до системи та змодельовано сценарії взаємодії користувача із програмним комплексом. Процес проєктування базується на детальному описі варіантів використання (Use Cases) та функціональних специфікацій, що дозволяє формалізувати логіку роботи застосунку, визначити послідовність дій користувача та очікувані реакції системи.

Розроблені моделі слугують основою для побудови діаграм варіантів використання та діаграм активності, які візуалізують потоки даних і переходи станів системи - від моменту автентифікації до отримання результатів інтелектуального аналізу. Такий підхід забезпечує цілісне бачення функціональності перед етапом безпосередньої програмної реалізації.

Основним актором у системі є користувач, який має на меті провести тематичний і емоційний аналіз контенту Reddit. Після запуску застосунку користувач взаємодіє з інтерфейсом авторизації, де здійснює вхід до системи через електронну пошту або за допомогою Google OAuth. Після успішної автентифікації система надає доступ до персоналізованого середовища роботи, де зберігаються попередні аналізи, налаштування та історія запитів. Якщо користувач вводить некоректні облікові дані, система повідомляє про помилку, пропонуючи пройти повторну спробу входу або зареєструвати новий обліковий запис.

Після авторизації користувач переходить до етапу налаштування параметрів аналізу. На цій сторінці він визначає джерело даних - набір сабреддів або конкретне посилання на тред Reddit, задає часовий діапазон збору (наприклад, за останні 24 години, 7 днів або 30 днів) і встановлює обмеження кількості постів. Система дозволяє попередньо перевірити

доступність даних і зберегти обрані налаштування у вигляді пресета для подальшого використання. Таким чином, користувач може повторювати типові запити, не вводячи параметри щоразу вручну.

Після налаштування параметрів запускається процес збору даних. Система звертається до офіційного Reddit API, отримує обрані пости й коментарі, нормалізує отримані поля, очищує текст і зберігає дані у базі Firestore. Після завершення збору користувач отримує повідомлення про готовність даних до аналізу. У разі помилок запиту система повідомляє про проблему, зокрема про перевищення лімітів API або відсутність даних за обраними критеріями.

Наступний етап - проведення аналізу. Коли користувач натискає кнопку запуску, застосунок виконує послідовно кілька шарів обробки. Спочатку реалізується NLP-аналіз, який передбачає лематизацію, видалення стоп-слів, підрахунок частот n-грам, виявлення колокацій, оцінку лексичної різноманітності, побудову часових рядів і розподілів за мовами. Отримані статистичні показники стають основою для подальшої генеративної інтерпретації. Другий шар обробки здійснюється локальною мовною моделлю через LM Studio. Система формує запит із ключовими NLP-показниками й прикладами сирих текстів, передає його в локальний API моделі openai/gpt-oss-20b і отримує розгорнутий текстовий звіт, який містить коротке резюме, основні висновки, спостереження щодо емоційної тональності, ризиків або упередженості та зазначення можливих обмежень аналізу.

Завершальний етап сценарію передбачає перегляд результатів у аналітичній панелі. На екрані користувач бачить інтерактивні графіки, таблиці, частотні діаграми та статистичні показники, отримані на етапі NLP. Нижче відображається текстовий блок із позначкою “AI-generated Insights by LM Studio”, де наведено сформульовані III-висновки. Користувач може експортувати результати у зручному форматі - CSV, JSON або PDF - або повторно запустити лише AI-аналіз, не виконуючи повторне збирання даних. Усі дії користувача й системні події - запуск аналізу, генерація, експорт,

помилки або повторні спроби - фіксуються у логах, що дозволяє відстежувати історію роботи й підвищує відмовостійкість системи.

Таким чином, варіанти використання застосунку охоплюють повний життєвий цикл взаємодії користувача із системою: від авторизації та налаштування параметрів аналізу до збору, обробки, інтерпретації й представлення результатів. Це забезпечує логічну послідовність роботи, зрозумілий інтерфейс і можливість повторного використання збережених даних для подальших досліджень. На рисунку 2.1 зображено діаграму варіантів використання застосунку.

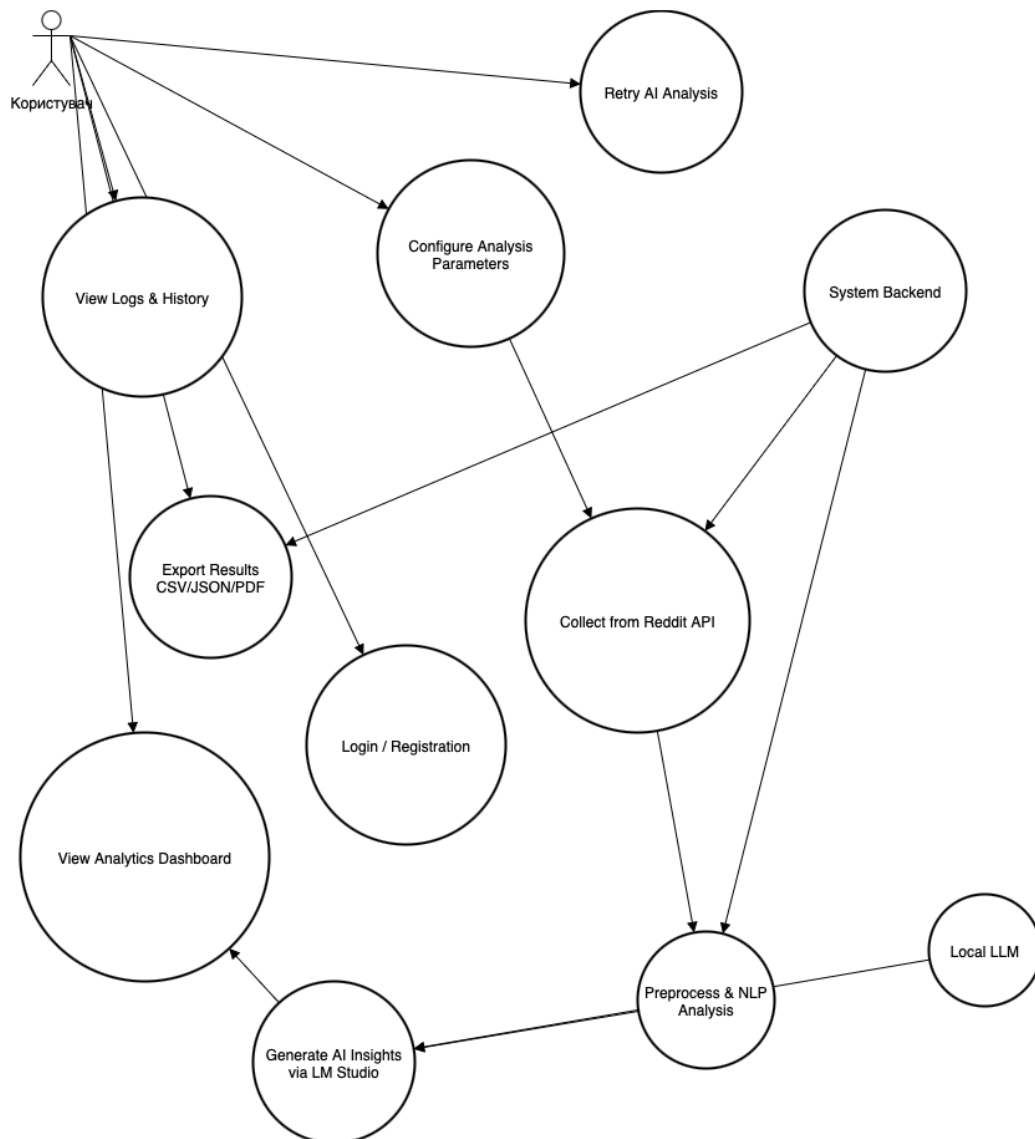


Рисунок 2.1 – Діаграма варіантів використання застосунку

Також в таблиці 2.1 наведено описову таблицю всіх варіантів використання. Цей перелік деталізує специфікації кожного прецеденту, визначаючи ключових акторів та очікувані результати виконання. Опис охоплює як інтерактивні дії користувача (налаштування, перегляд, експорт), так і фонові процеси бекенду, пов'язані зі збором даних через API, NLP-обробкою та генерацією інсайтів локальною мовною моделлю. Такий структурований підхід дозволяє чітко розмежувати зони відповідальності між клієнтською частиною, сервером та зовнішніми модулями, гарантуючи повноту реалізації заявленого функціоналу.

Таблиця 2.1

Опис варіантів використання

ID	Use Case	Actor	Короткий опис / Результат
UC1	Login/ Registration	User	Користувач входить через email/пароль або Google OAuth; отримує доступ до приватних даних і історії аналізів.
UC2	Configure Analysis Parameters	User	Задає сабреддіти або URL треду, часовий діапазон і ліміт; може зробити Preview та зберегти пресет.
UC3	Collect from Reddit API	System Backend	Сервіс збирає пости/коментарі через офіційний Reddit API (OAuth2), нормалізує та зберігає сирі дані.
UC4	Preprocess & NLP Analysis	System Backend	Очищення тексту, мова, токени/леми, антиспам; підрахунок n-грам, колокацій, лексичного різноманіття, часових рядів.
UC5	Generate AI Insights via LM Studio	Local LLM	Формування промпту з NLP-зведенням + уривками сирого тексту; LM Studio (openai/gpt-oss-20b) генерує структуровані інсайти.
UC6	View Analytics Dashboard	User	Переглядає дашборд: зверху - результати NLP (графіки/таблиці), нижче - AI-insights; застосовує фільтри.

UC7	Export Results (CSV/JSON/PDF)	User	Експортує елементи аналізу, агрегати та звіт (PDF) для документування або подальшої обробки.
UC8	Retry AI Analysis	User	Повторно запускає лише AI-етап без повторного збору/процесінгу даних у разі збою або оновлення моделі.
UC9	View Logs & History	User	Переглядає історію аналізів і основні логи подій; може повторно відкрити або відтворити минулі запуски.

Функціональні вимоги описують поведінку системи з точки зору користувача, а також визначають очікувані реакції програми на конкретні дії. У межах цієї кваліфікаційної роботи вони охоплюють як користувацький інтерфейс (frontend), так і серверну частину (backend), а також інтеграцію із зовнішніми сервісами - Reddit API, Firebase, Redis і локальною мовною моделлю LM Studio.

Система повинна реалізовувати можливість автентифікації користувачів через Firebase Authentication, підтримуючи вхід за електронною поштою, паролем або через Google OAuth. Після авторизації користувач має отримувати доступ до персоналізованого середовища, у якому зберігаються його історія аналізів, налаштування й пресети параметрів. Усі події входу та виходу з системи мають логуватися у Firestore.

Ключовою функціональністю є налаштування параметрів аналізу. Користувач має мати змогу вказати один або кілька сабреддів чи конкретне посилання на тред Reddit, визначити часовий діапазон збору даних, ліміт кількості постів і коментарів, а також попередньо переглянути доступність інформації через функцію «Preview». Додатково система повинна надавати можливість збереження вибраних параметрів у вигляді пресетів для подальшого використання.

Після задання параметрів користувач може ініціювати процес збору даних. Backend системи повинен звертатися до офіційного Reddit API із використанням механізму OAuth2, обробляти rate limits, виконувати повторні спроби при помилках і зберігати отриману інформацію у структурованому форматі у Firebase Firestore. На цьому етапі дані проходять попередню обробку - очищення від HTML, емодзі, посилань і символів, виявлення мови, токенізацію, лематизацію та видалення стоп-слів. Система також має фільтрувати дублікати та спам, позначаючи причини фільтрації у метаданих.

Подальший етап функціональності охоплює реалізацію модулів NLP-аналітики. Програма повинна розраховувати частотність n-грам (від 1 до 3), знаходити колокації (за показниками PMI і t-score), обчислювати показники лексичного різноманіття (TTR, MATTR, MTLT, Herdan C, Sichel S, ентропію), а також формувати часові розподіли активності користувачів і тональності текстів. Ці результати мають бути збережені у Firestore та кешовані у Redis з обмеженим часом життя для оптимізації повторних запитів.

Додатковим компонентом є модуль генеративного аналізу за допомогою локальної мовної моделі openai/gpt-oss-20b, яка виконується через LM Studio за адресою <http://127.0.0.1:1234>. Система повинна формувати вхідний запит до LLM, який містить метадані аналізу, короткі підсумки NLP-результатів і уривки сирих текстів. Отримана від моделі відповідь має бути у форматі JSON та включати згенероване резюме, основні інсайти, загальний огляд емоційної тональності, можливі ризики чи упередження та зазначення обмежень аналізу. У випадку помилки система повинна позначати статус LLM як «error» та надавати користувачу можливість повторного запуску лише AI-етапу.

На клієнтському рівні передбачено реалізацію аналітичної панелі. Користувач має бачити результати NLP-аналізу у вигляді інтерактивних графіків, таблиць і діаграм (частотність n-грам, розподіл мов, активність користувачів, показники лексичного різноманіття). Нижче на тій самій сторінці відображається блок AI-інтерпретації, який містить текстові висновки, сформульовані мовною моделлю. Інтерфейс має бути приємним

візуально, виконаним у сучасному мінімалістичному стилі з акцентами синього кольору, із плавними переходами й адаптивним дизайном.

Функціональні вимоги також охоплюють можливість експорту результатів у форматах CSV, JSON і PDF. Експортований PDF-звіт повинен містити заголовок, опис параметрів аналізу, графічні елементи з результатами NLP-обробки, а також текстову частину, сформовану мовною моделлю. Усі етапи аналізу, генерації, експорту, а також усі помилки мають бути зафіксовані у логах системи, що зберігаються локально й у Firestore.

Підсумовуючи, функціональні вимоги визначають комплексну поведінку системи від моменту входу користувача до отримання повного звіту про аналіз Reddit-контенту. Вони охоплюють взаємодію між користувачем, бекендом, базою даних, локальною мовною моделлю й інтерфейсом візуалізації, забезпечуючи повноцінну автоматизовану аналітичну платформу.

Діаграма активності відображає послідовність дій, що виконуються під час взаємодії користувача із системою Reddit Content Analyzer, починаючи з авторизації та закінчуючи отриманням результатів аналізу. Ця діаграма демонструє логіку обробки запиту - як переходить управління між користувачем, клієнтським інтерфейсом, сервером і локальною мовною моделлю. Вона показує, що система працює у чіткій послідовності етапів: введення параметрів → збір даних → попередня обробка → NLP-аналіз → генерація AI-висновків → візуалізація результатів → експорт.

Після запуску застосунку користувач спочатку проходить автентифікацію через Firebase Auth. Якщо вхід успішний, він переходить до сторінки налаштування параметрів аналізу, де вводить сабреддіти, посилання або хештеги, визначає часовий діапазон і ліміт кількості постів. Після натискання кнопки «Start Analysis» система ініціює процес збору даних із Reddit API. Отримані пости передаються на етап попередньої обробки, де вони очищуються, лематизуються, визначається мова й видаляються дублікати. Потім система виконує NLP-аналіз: підраховує частотність лексем, генерує n-

грамні статистики, визначає емоційне забарвлення текстів, а також показники лексичної різноманітності.

Далі результати цього аналізу разом із частиною сирих текстів передаються локальній LLM-моделі через LM Studio, яка формує аналітичне резюме. Коли відповідь отримана, система відображає результати на панелі аналітики: у верхній частині - статистичні та графічні показники NLP, у нижній - текстові висновки, сформовані ШІ. Після перегляду користувач може експортувати результати у потрібному форматі (CSV, JSON, PDF) або повторно запустити лише AI-етап у разі помилки.

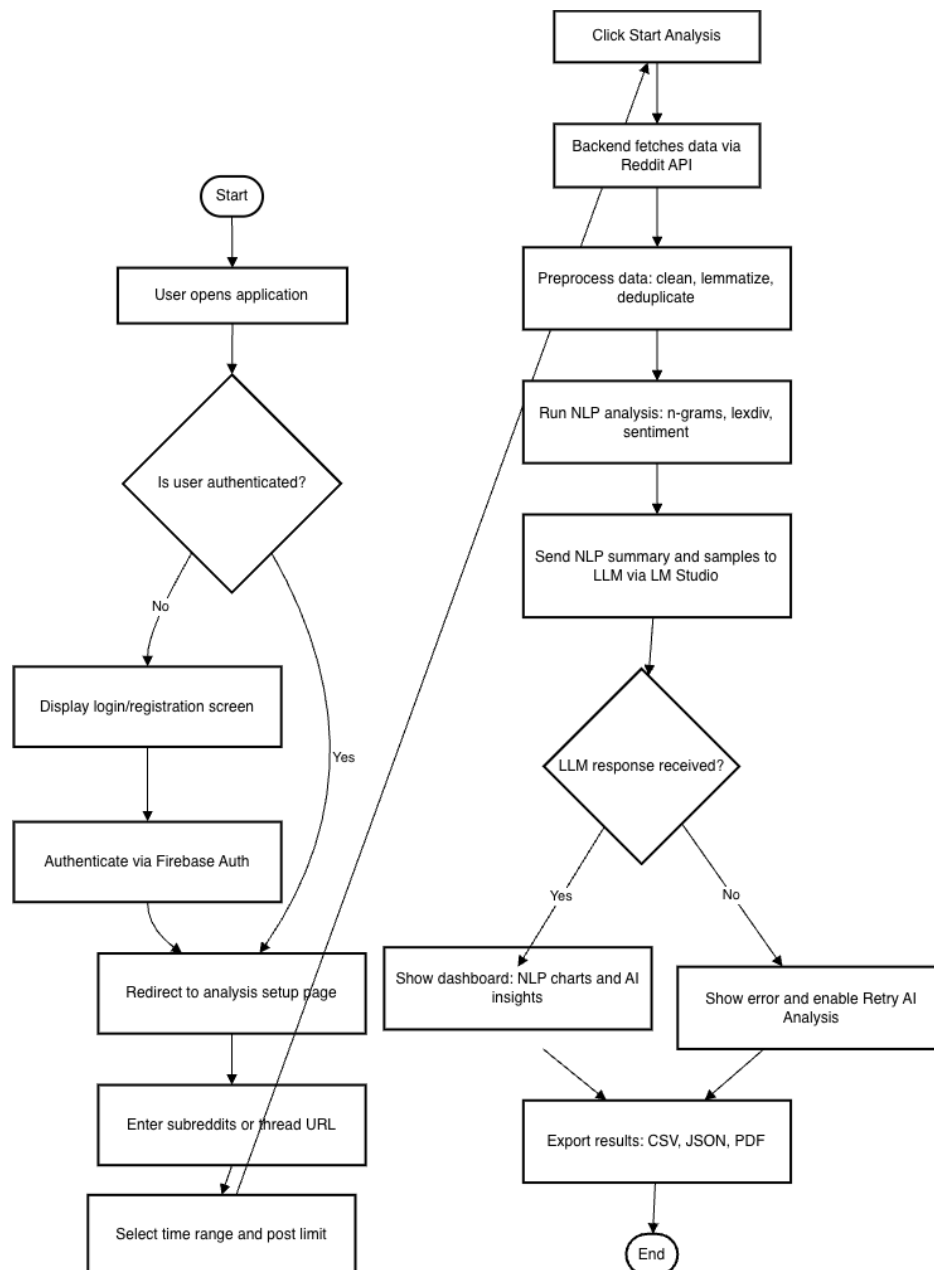


Рисунок 2.2 – Діаграма активності застосунку

Діаграма активності, відображена на рисунку 2.2 показує логіку роботи системи у вигляді потоку активностей: спочатку виконується перевірка автентифікації, потім користувач налаштовує параметри аналізу, після чого відбувається виклик Reddit API і послідовна обробка даних. У момент, коли локальна модель завершує генерацію, система переходить до візуалізації результатів.

Якщо відповідь від LLM не отримана, користувач може повторити запит. Завершальним кроком є експорт результатів або завершення роботи з інтерфейсом. Таким чином, діаграма демонструє не лише логічну послідовність виконання дій, а й циклічність процесу, коли аналіз може повторюватися багаторазово з різними параметрами.

2.2 Архітектура програмного комплексу та інфраструктура розгортання

Архітектура розробленої системи побудована за принципами багаторівневої клієнт–серверної моделі [38], що забезпечує чітке розмежування між інтерфейсом користувача, бізнес-логікою сервера та шаром даних. Така організація дозволяє незалежно масштабувати компоненти, підвищує надійність системи та спрощує інтеграцію із зовнішніми сервісами, такими як Reddit API та Firebase.

Інфраструктура розгортання реалізована з використанням технологій контейнеризації, що гарантує ізоляцію процесів, відтворюваність середовища виконання та стабільність роботи локальних AI-моделей [9, 27]. Фізична схема взаємодії компонентів виглядає наступним чином.

У користувача працює браузер, який завантажує статичні ресурси фронтенду (React SPA) з хостингу фронтенда або CDN і звертається до бекенду по HTTPS. На сервері застосунку одночасно працюють три ключові процеси: API-сервер на Node.js/Express (TS), LM Studio з локальною моделлю openai/gpt-oss-20b, а також Redis для кешу/черг. Той самий сервер зберігає

локальні файлові логи з ротацією. Хмарні сервіси Firebase надають ідентифікацію (Firebase Auth) та сховище даних (Firestore). Інтеграція з Reddit API відбувається лише з бекенду, з використанням OAuth2 (app-only). На рисунку 2.3 зображено діаграму розгортання застосунку. Клієнтська частина спілкується з фронтенд-хостингом і бекендом виключно по протоколу HTTPS через порт 443. Якщо у схемі використовується проксі-сервер (наприклад, Nginx), він термінує TLS-з'єднання, проксуює запити до API на внутрішній порт та видає статичні файли SPA.

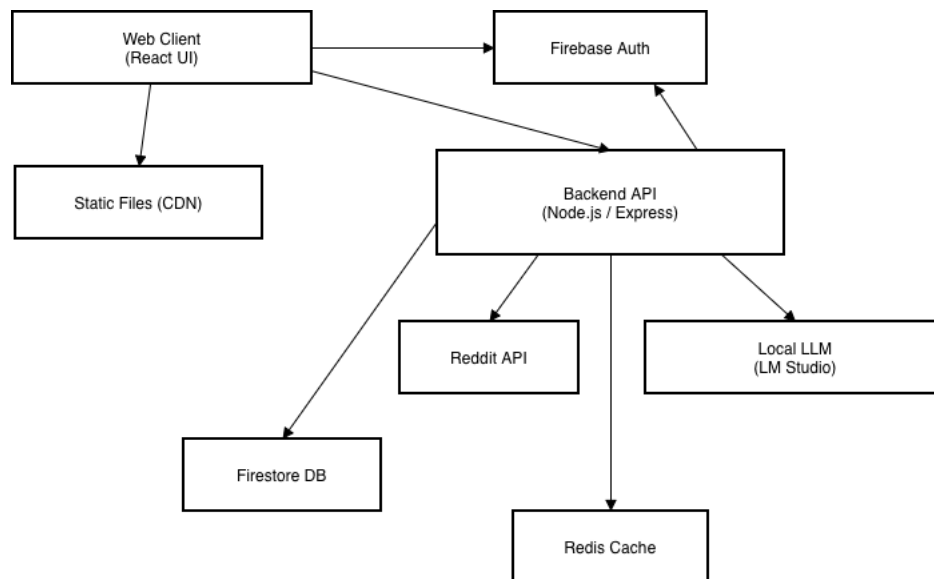


Рисунок 2.3 – Діаграма розгортання застосунку

Клієнтська частина спілкується з фронтенд-хостингом і бекендом виключно по HTTPS/443. Якщо використовується проксі-сервер (наприклад, Nginx), він термінує TLS і проксує запити до API (типово внутрішній порт 3000) і видає статичні файли SPA. LM Studio працює на локальному інтерфейсі за адресою `http://127.0.0.1:1234`; зовнішній доступ до цього порту закритий фаєрволем. Redis слухає 6379 лише на приватному інтерфейсі (localhost або VPC), без відкриття в інтернет. Бекенд здійснює вихідні HTTPS-з'єднання до Firebase Auth/Firestore і до Reddit API. Вхід на 80/tcp має робити редирект на 443/tcp (HTTP→HTTPS).

Після логіну браузер отримує ID-токен Firebase. Кожен приватний виклик до API супроводжується цим токеном у заголовку авторизації; бекенд

перевіряє підпис токена через Firebase Admin SDK. Параметри збору передаються на API, який звертається до Reddit, приймає і нормалізує пости/коментарі, зберігає “сирі” та підготовлені документи у Firestore. На етапі NLP виконується локальна обробка, результати кешуються в Redis та/або матеріалізуються в Firestore. Для генерації підсумків бекенд формує стислий контекст (агрегати NLP + уривки текстів) і звертається до LM Studio по локальному HTTP. Відповідь моделі у валідується як JSON і записується до Firestore. Браузер отримує зведені дані дашборда (спочатку NLP, нижче - AI-висновки) та, за потреби, ініціює експорт.

Всі приватні маршрути API захищено перевіркою Firebase ID-токена; в Firestore Rules передбачено ізоляцію даних на рівні `request.auth.uid`. На мережевому рівні дозволяються лише необхідні входи: 443/tcp з інтернету до проксі/API, 22/tcp для адміністрування (обмежений за IP), внутрішні порти 3000 (API), 1234 (LM Studio), 6379 (Redis) - доступні тільки локально або в межах приватної мережі. CORS обмежено доменом фронтенда. Секрети (креденшили Reddit, Firebase, ключі) зберігаються у змінних середовища або менеджері секретів; до репозиторію не потрапляють. TLS-сертифікати - через Let's Encrypt або керовану CDN; політики безпеки браузера (CSP, HSTS) вмикаються на проксі.

Бекенд пише структуровані JSON-логи (Winston/Pino) на консоль і у файлову систему з ротацією (добові файли), а також дублює ключові події у Firestore (`users/{uid}/analyses/{analysisId}/logs`). Логується початок/завершення етапів (`reddit`, `preprocess`, `nlp`, `llm`, `export`), тривалість операцій, розмір промπτу, коди/повідомлення помилок. Для швидкої діагностики передбачено `/health` (жива/готовність), метрики часу відповіді і валідації токена. За потреби, логи можуть бути підхоплені агентом (Filebeat/Vector) і відправлені у віддалене сховище або стектипу ELK/Cloud Logging.

Redis використовується як кеш агрегатів NLP (TTL $\approx$ 15 хв) і як проста черга/пула задач для довгих операцій (наприклад, великий тред або повторний

AI-запит). API є безстанним, аутентифікація - токенова, тож горизонтальне масштабування можливе без “клейких” сесій. Критичні секції (rate limit Reddit, одночасний запис у Firestore) захищаються ідемпотентними ключами та експоненційним backoff. Генерацію PDF/CSV виконують окремі воркери або той самий процес, але в ізольованих чергах, щоб не блокувати запити користувача.

Стартова конфігурація - один App Server типу 4–8 vCPU / 16–32 GB RAM (через RAM виграє і LM Studio, і Node), SSD-сховище для логів/тимчасових файлів. Якщо навантаження зростає, додають другий інстанс API за балансером; Redis виноситься на керований вузол або кластер; LM Studio може залишатися локально на одному з інстансів (sticky “AI-запити” через інтернал балансер) або масштабуватися окремо (по одній моделі на вузол). Firestore масштабується керовано. Важливо вчасно обмежувати розмір промптів і батчів NLP, щоб не створювати “пилу” затримок.

Репозиторій на GitHub інтегрується з GitHub Actions: збірка та лінт фронту й бекенду, юніт/інтеграційні тести, перевірка типів, далі - деплой. Фронтенд публікується на статичному хостингу або за CDN; бекенд - як контейнер, що розгортається на сервері або оркестраторі (Docker Compose/Kubernetes). Конфігурації середовищ dev/stage/prod рознесено по .env/секрет-менеджеру; чутливі значення у секретах, а не в коді. Стратегії оновлення - rolling або blue-green з перевіркою /health.

Firestore має вбудовані SLA і резервування; для аудиту корисно періодично експортувати знімки колекцій у хмарне сховище. Redis можна налаштувати на AOF/RDB, але як кеш/чергу він критичним не є - при рестарті система відновлюється. PDF/CSV можна відтворити повторною генерацією. Для LM Studio бажано мати процедуру “теплого перезапуску” моделі і моніторинг використання RAM/VRAM. Дизайн допускає безпечне деградування: якщо LLM недоступна, користувач бачить NLP-результати і може натиснути Retry AI пізніше.

LM Studio ізольовано на localhost:1234 або приватному інтерфейсі; зовнішніх входів до нього немає. На API діє жорстка валідація даних і ліміти розміру запитів/відповідей. У промпт не потрапляють персональні дані користувача (окрім публічного контенту Reddit і агрегатів NLP). Логи не містять сирих приватних фрагментів понад норму - за потреби застосовується маскування.

Типовий цикл: кілька секунд на збір і нормалізацію (залежно від ліміту), секунди-десятки секунд на NLP-агрегації (за рахунок кеша повторні перегляди швидкі), і до кількох секунд-десятиків на відповідь LLM. Інтерфейс показує прогрес етапів; дашборд відображає NLP одразу після готовності, а AI Insights - коли модель завершить генерацію. Експорт запускається асинхронно з повідомленням про готовність.

Схема відображає ці самі ролі та зв'язки: Browser завантажує SPA і звертається до API через HTTPS; API працює поруч із LM Studio, Redis, локальними логами; Firebase і Reddit - зовнішні інтеграції; вся приватна логіка та ключі - на бекенді. Межі довіри проходять між браузером і сервером (TLS + JWT), між сервером і публічними хмарами (HTTPS) та всередині приватної мережі (локальні порти для LM Studio/Redis).

В таблиці 2.2 наведено опис сутностей діаграми розгортання.

Таблиця 2.2

Опис сутностей діаграми розгортання

№	Вузол	Призначення	Порт / Протокол	Доступ
1	Web Browser	Клієнт користувача, інтерфейс React	HTTPS 443	Публічний
2	Frontend Hosting / CDN	Статичні ресурси застосунку (HTML, JS, CSS)	HTTPS 443	Публічний

3	Node.js / Express API	Основна логіка, інтеграції, обробка NLP/LLM, логування	HTTP 3000 (за проксі)	Внутрішній / через HTTPS
4	Redis	Кеш і черги завдань	TCP 6379	Локальний або приватна мережа
5	LM Studio	Локальна LLM (openai/gpt- oss-20b) для AI-висновків	HTTP 1234	Тільки localhost
6	Firebase Auth	Автентифікація користувачів	HTTPS 443	Зовнішній
7	Firestore Database	Збереження даних аналізів, логів, користувачів	HTTPS 443	Зовнішній
8	Reddit API	Збір контенту (OAuth2)	HTTPS 443	Зовнішній
9	Логи	Структуровані JSON-логи (Winston/Pino) з ротацією	Локальні файли	Внутрішній доступ

Архітектура програмного забезпечення реалізована за принципами багаторівневої клієнт–серверної системи та забезпечує повний цикл аналітичної обробки даних з Reddit - від автентифікації користувача до генерації узагальнених AI-висновків. Програмний комплекс складається з фронтенд-частини, реалізованої на React із використанням TypeScript і системи збирання Vite, та бекенд-частини на Node.js / Express.js, що відповідає за взаємодію з Reddit API, виконання NLP-аналізу, роботу з локальною LLM-моделлю та формування аналітичних звітів.

Клієнтська частина виконує роль інтерфейсу користувача, забезпечуючи аутентифікацію через Firebase Authentication, введення параметрів аналізу, перегляд результатів і експорт звітів. Архітектура фронтенду поділена на сторінки автентифікації (Login, Register, Reset), сторінки аналітики (Setup,

Dashboard) і контекст керування станом автентифікації (AuthContext), що забезпечує безперервну взаємодію з Firebase SDK.

Серверна частина організована у вигляді модульної системи з чітким розмежуванням обов'язків між контролерами, проміжними обробниками (middleware) та сервісами. Контролери відповідають за маршрутизацію запитів, тоді як middleware реалізують перевірку токенів Firebase, логування та обробку помилок. Основна бізнес-логіка зосереджена у сервісах: Reddit Service виконує авторизацію через OAuth2 та збір публікацій; Preprocessing Service очищує текст, визначає мову, видаляє спам і дублікати; NLP Service виконує частотний аналіз, обчислення лексичного різноманіття, n-грам і евристичний аналіз тональності; LLM Service забезпечує інтеграцію з локальним модулем LM Studio, де модель openai/gpt-oss-20b генерує узагальнення та інтерпретації; Export Service формує підсумкові звіти у форматах CSV, JSON і PDF.

У центрі архітектури розташований Pipeline Service, який координує послідовність виконання всіх етапів: валідацію введених параметрів, збір даних із Reddit, попередню обробку, NLP-аналіз, звернення до LLM і збереження результатів у Firebase Firestore. Для кешування проміжних результатів використовується Redis, що зменшує затримки при повторному аналізі тих самих тем. Логування виконано через Winston, а схема даних перевіряється за допомогою Zod.

Передача даних здійснюється через REST API у форматі JSON. Усі приватні маршрути захищені токенами Firebase, що перевіряються middleware. На рівні клієнта забезпечено асинхронну комунікацію з API, а результати відображаються у вигляді інтерактивних графіків, створених із використанням бібліотеки Recharts.

З точки зору розгортання система підтримує контейнеризацію через Docker Compose, де окремими сервісами виступають вебінтерфейс, API та Redis. Усі зовнішні сервіси - Firebase, Reddit API, LM Studio - взаємодіють із сервером через захищені канали HTTPS. Такий підхід забезпечує

масштабованість, розширюваність і безпечне виконання операцій у межах однієї інфраструктури

2.3 Програмна реалізація компонентів аналізу та обробки даних

Реалізація функціональності системи базується на архітектурі «клієнт–сервер» і поділяється на два основні компоненти: клієнтський інтерфейс (React + TypeScript) і серверну частину (Node.js + Express + TypeScript). Кожна частина має чітко визначену роль у загальному процесі аналізу контенту Reddit - клієнт відповідає за взаємодію з користувачем, а сервер обробляє дані, виконує аналітичні обчислення, керує логікою обробки тексту та інтеграцією з локальною моделлю LM Studio.

Робота системи починається з процесу аутентифікації користувача. Для цього використовується бібліотека Firebase Authentication, яка дозволяє авторизуватися як за допомогою пари логін/пароль, так і через Google OAuth. На стороні клієнта виклик аутентифікації виконується через офіційний SDK Firebase, який після успішного входу повертає токен аутентифікації (ID Token). Цей токен передається в кожному запиті на бекенд у заголовок Authorization.

На сервері створюється спеціальне проміжне програмне забезпечення (middleware), яке перевіряє цей токен перед виконанням будь-яких приватних запитів. У коді реалізовано функцію, яка отримує заголовок запиту, виділяє з нього токен і надсилає на перевірку до Firebase Admin SDK. Якщо перевірка проходить успішно, до об'єкта запиту додається інформація про користувача (req.user), і керування передається наступному обробнику. Якщо ж токен відсутній або недійсний, запит завершується з кодом помилки 401 або 403. Такий підхід дозволяє ізолювати рівень безпеки від основної логіки застосунку - усі захищені маршрути проходять через цей фільтр, а це означає, що ні один запит без чинного токена не зможе викликати внутрішні функції аналізу чи запису у Firestore.

Після автентифікації користувач потрапляє до інтерфейсу налаштування аналізу. Цей інтерфейс створено у вигляді функціонального компонента React, який керує своїм станом через React Hooks (useState) та використовує асинхронну функцію для виклику API. Поля введення (input, select) дозволяють користувачу задати сабреддіти або конкретний URL, часовий діапазон і кількість постів, які потрібно проаналізувати. Після натискання кнопки “Start Analysis” функція startAnalysis() формує JSON-об’єкт із параметрами користувача і надсилає його методом POST на маршрут /api/analysis/run. Сам компонент не містить бізнес-логіки - він лише збирає дані і делегує обробку серверу. Такий розподіл обов’язків відповідає принципам «чистої архітектури» та спрощує підтримку коду.

На серверному боці обробка запиту /api/analysis/run ініціює багатоетапний конвеєр (pipeline), що координує взаємодію між модулями збору даних, лінгвістичного аналізу та генеративного штучного інтелекту.

Процес збору та нормалізації даних розпочинається з валідації вхідних параметрів (список сабреддітів, часовий діапазон, ліміт публікацій). Функція виконує авторизований запит до Reddit API за протоколом OAuth2, використовуючи токен, отриманий при ініціалізації сервера. Отримані дані, що надходять у вигляді складних JSON-структур, проходять етап нормалізації.

З масиву відповідей виділяються ключові сутності (заголовок title, текст selftext, автор author, метрики залученості), які приводяться до уніфікованого інтерфейсу. Це забезпечує незалежність внутрішньої логіки системи від можливих змін у форматі відповіді зовнішнього API.

Етап попередньої обробки є критично важливим для якості подальшого аналізу. У розробленій системі він реалізований функцією яка працює як композиція фільтрів. Формально процес очищення вхідного документа можна представити такою формулою:

$$D_{clean} = f_{stop} \left(f_{lemma} \left(f_{token} \left(f_{norm} (D_{raw}) \right) \right) \right) \quad (2.1)$$

Функція нормалізації f_{norm} використовує регулярні вирази для видалення HTML-тегів, URL-адрес та емодзі. Паралельно з очищенням виконується детекція спаму та токсичності. Алгоритм класифікує повідомлення як спам, якщо виконується одна з умов: Кількість гіперпосилань у тексті перевищує поріг ($Nlinks > 3$); Виявлено аномальну частоту повторення слів (repetition flood); Текст містить обценну лексику (перевірка за словником).

Ідентифікація мови здійснюється бібліотекою `franc` на основі триграмного аналізу. Токенізація та видалення стоп-слів виконується з використанням пакетів `natural` та `stopword`.

Модуль лінгвістичного аналізу обчислює кількісні показники тексту. Для глибокої оцінки словникового запасу в системі реалізовано комплекс метрик лексичної різноманітності. У таблиці 2.3 відображені моделі, які використовуються у модулі NLP-аналізу.

Таблиця 2.3

Кількісні показники та метрики лінгвістичного аналізу

Назва метрики	Формула	Опис метрики
Type-Token Ratio (TTR)	$TTR = \frac{V}{N}$	Базовий показник лексичного різноманіття, що визначає відношення кількості унікальних лексем (V) до загальної кількості слів (N).
Moving-Average TTR (MATTR)	$MATTR = \frac{1}{M} \sum_{i=1}^M \frac{V_i}{L}$	Метод ковзного вікна для оцінки різноманіття незалежно від довжини тексту. L - розмір вікна (50), M - кількість вікон, V_i - унікальні лексеми у вікні.
MTLD (Measure of Textual Lexical Diversity)	$MTLD = \frac{N}{F}$	Метрика для довгих текстів. Визначається як відношення кількості токенів N до кількості факторів F, сформованих при падінні TTR нижче порогу 0.72.
Ентропія Шеннона (Shannon Entropy)	$H(T) = - \sum_{i=1}^V p_i \log_2 p_i$	Показник інформаційної щільності та невизначеності тексту. p_i - відносна частота появи i-го унікального слова.

Поточкова взаємна інформація (PMI)	$PMI(w_1, w_2) = \log_2 \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$	Міра асоціації для виявлення колокацій. Оцінює, наскільки ймовірність спільної появи слів вища за ймовірність їх незалежної появи.
Т-критерій (T-score)	$T = \frac{f(w_1, w_2) - \frac{f(w_1)f(w_2)}{N}}{\sqrt{f(w_1, w_2)}}$	Статистичний критерій для підтвердження значущості колокації. Враховує частоти появи слів $f(w)$ та знижує вагу рідкісних випадкових комбінацій.

Завершальним етапом є генеративний аналіз, система агрегує отримані NLP-показники та приклади сирих текстів у JSON-пакет, який надсилається до локального сервера LM Studio (адреса <http://127.0.0.1:1234>). Для взаємодії використовується модель openai/gpt-oss-20b. Промпт формується з використанням рольової моделі: System Role: Задає контекст («You are an AI analyst...») та вимагає відповідь виключно у форматі JSON; User Role: Передає структуровані дані аналізу.

Для забезпечення стабільності відповіді параметр варіативності (temperature) встановлено на рівні 0.2. Отримана відповідь проходить валідацію та парсинг; у разі помилки генерації система фіксує статус error, дозволяючи користувачу повторити запит до AI-модуля без необхідності повторного збору даних.

На рівні інтерфейсу компонент Dashboard забезпечує візуалізацію отриманих даних. Використання бібліотеки Recharts дозволяє будувати інтерактивні графіки розподілу мов, активності користувачів та частотності n-грам. Блок «AI Insights» оновлюється асинхронно, щойно сервер завершує генерацію висновків, що забезпечує чутливість інтерфейсу (non-blocking UI).

Для документування результатів реалізовано модуль експорту (exportService), який дозволяє генерувати звіти у форматах CSV, JSON та PDF. Генерація PDF-файлів виконується бібліотекою pdfkit на стороні сервера, що

дозволяє створювати деталізовані звіти з графікою та текстом без навантаження на браузер клієнта.

Розроблена архітектура забезпечує повний цикл обробки даних - від збору до інтелектуальної інтерпретації. Модульність коду дозволяє в майбутньому замінити Reddit API на інше джерело даних або змінити модель LLM без необхідності переписання основної бізнес-логіки. Використання кешування, валідації даних та асинхронної обробки робить систему стійкою до навантажень та зручною для користувача.

Висновки до розділу 2

Програмна реалізація компонентів системи підтвердила ефективність обраної сервіс-орієнтованої архітектури, забезпечивши створення гнучкого та масштабованого інструменту для автоматизованого аналізу соціального контенту. Ключовим досягненням етапу розробки стала успішна інтеграція різномірних технологічних стеків у єдиний аналітичний конвеєр, де чітко розмежовано зони відповідальності між клієнтським інтерфейсом, серверною бізнес-логікою та модулями штучного інтелекту. Реалізація механізмів автентифікації через проміжне програмне забезпечення та валідацію токенів гарантувала високий рівень безпеки та ізоляцію даних користувачів, що є критичною вимогою для сучасних вебзастосунків.

Особливу увагу в процесі реалізації було приділено математичному апарату лінгвістичного аналізу. Програмна інтерпретація алгоритмів попередньої обробки тексту та обчислення складних метрик, таких як MATTR, MTLD, ентропія Шеннона та показники взаємної інформації (PMI), дозволила перейти від поверхневого статистичного огляду до глибокої оцінки семантичної структури та лексичного багатства дискусій. Це забезпечило об'єктивність кількісних показників, які слугують фундаментом для подальшої генеративної обробки.

Важливим результатом стала практична реалізація гібридного підходу до аналізу даних, що поєднує детерміновані алгоритми NLP з когнітивними можливостями локальної великої мовної моделі. Розроблена схема динамічного формування промптів та валідації JSON-відповідей забезпечила стабільність роботи генеративного модуля, мінімізуючи ризики галюцинацій моделі. У поєднанні з оптимізаційними рішеннями, такими як кешування результатів у Redis та асинхронна візуалізація даних на дашборді, створена система демонструє високу продуктивність та стійкість до навантажень. Таким чином, програмна реалізація повністю відповідає сформульованим функціональним вимогам та створює надійну технічну базу для проведення експериментальних досліджень і перевірки ефективності системи, що буде здійснено у наступному розділі роботи.

РОЗДІЛ 3. ТЕСТУВАННЯ ТА РОБОТА З СИСТЕМОЮ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ КОНТЕНТУ

3.1 Опис реалізації автоматичного тестування

Для забезпечення надійності та відтворюваності результатів системи було впроваджено автоматичне тестування ключових модулів, зокрема механізмів попередньої обробки та фільтрації контенту. Використання юніт-тестів дозволяє верифікувати коректність роботи алгоритмів на контрольованих прикладах і гарантує стабільність функцій під час подальших змін у кодовій базі.

Даний тест перевіряє коректність виявлення спам-контенту та ненормативної лексики в модулі попередньої обробки. Використовуючи штучно створений набір текстів, тест імітує реальні сценарії токсичних публікацій і перевіряє, чи позначає система такі записи як спам, а також чи додає відповідну причину profanity. Позитивний результат доводить, що алгоритм виявлення спаму на основі ключових слів і евристичних правил функціонує стабільно та запобігає подальшій участі небажаного контенту в NLP-аналізі.

```
describe('preprocessDocuments', () => {  
  it('should detect spam content', () => {  
    const spamPosts = [{ text: 'This is spam with fuck shit bitch words' }];  
    const result = preprocessDocuments(spamPosts);  
    expect(result[0].is_spam).toBe(true);  
    expect(result[0].spam_reasons).toContain('profanity');  
  });  
});
```

Цей тест оцінює працездатність функції експорту результатів аналізу у форматі PDF. Він створює тимчасовий набір параметрів аналізу, викликає функцію exportToPDF і перевіряє, що повертається шлях до сформованого файлу. Тест також контролює відсутність помилок введення-виведення та коректне завершення потоку запису. Така перевірка гарантує цілісність процесу генерації звітів і підтверджує, що користувач може зберігати

результати досліджень у форматі, придатному для подальшої публікації або документування.

```
describe('exportToPDF', () => {  
  it('should generate PDF with analysis data', async () => {  
    const result = await exportToPDF(analysisId, params);  
    expect(result).toBe('/tmp/analysis-test.pdf');  
  });  
});
```

Фрагмент демонструє перевірку проміжного шару автентифікації, який забезпечує валідацію токенів Firebase ID. Тест створює підроблений об'єкт запиту з коректним токеном, імітує успішне розшифрування за допомогою `mockVerifyIdToken` та перевіряє, що в запиті з'являється `uid`, а управління передається наступному обробнику (`next`). Така перевірка підтверджує, що система правильно інтегрована з `Firebase Auth`, коректно розпізнає авторизованих користувачів і блокує неавторизовані запити, що критично для безпеки сервісу.

```
describe('verifyFirebase middleware', () => {  
  it('should verify valid Firebase token and call next', async () => {  
    mockVerifyIdToken.mockResolvedValueOnce({ uid: 'test-user-123' });  
    mockReq.headers = { authorization: 'Bearer valid-token-123' };  
    await verifyFirebase(mockReq, mockRes, mockNext);  
    expect(mockReq.uid).toBe('test-user-123');  
    expect(mockNext).toHaveBeenCalledTimes(1);  
  });  
});
```

Даний тест спрямований на перевірку функції очищення вихідного тексту, що надходить від мовної моделі `LM Studio`. Метою є вилучення валідного JSON із маркованих `markdown`-блоків, які часто присутні у відповідях моделей типу `GPT`. Тест перевіряє, що функція правильно виділяє вміст між трьома зворотними апострофами, і що результат не містить службових символів. Такий механізм забезпечує надійну десеріалізацію відповідей `LLM` та запобігає виникненню помилок при подальшому парсингу чи збереженні результатів у `Firestore`.

```
describe('sanitizeLLMContent', () => {
  it('extracts JSON from fenced code blocks', () => {
    const content = "Here is the result:\n```\njson\n{\n  \"foo\": \"bar\"\n}\n```\n";
    expect(sanitizeLLMContent(content)).toBe('{"foo": "bar"}');
  });
});
```

Цей тест забезпечує перевірку модуля NLP, який обчислює показники лексичного різноманіття. Задано невеликий набір токенів із повтореннями, а функція `lexicalMetrics` повинна розрахувати коефіцієнт TTR (тип-токен рейт). Тест контролює, що значення відповідає очікуваному математичному результату 0.75, що свідчить про правильну реалізацію підрахунку унікальних лексем. Перевірка дозволяє гарантувати точність кількісних показників мовного аналізу, необхідних для порівняння різних текстових корпусів у процесі дослідження Reddit-контенту.

```
describe('lexicalMetrics', () => {
  it('calculates lexical diversity metrics', () => {
    const tokens = ['hello', 'world', 'hello', 'analysis'];
    const metrics = lexicalMetrics(tokens);
    expect(metrics.TTR).toBeCloseTo(0.75, 2);
  });
});
```

Фрагмент демонструє тестування утиліти `sanitizeForFirestore`, яка очищує об'єкти від `undefined` перед передачею до бази Firestore. Тест перевіряє, що функція коректно видаляє порожні властивості як на верхньому, так і на вкладеному рівнях, залишаючи лише визначені значення. Такий підхід гарантує узгодженість структури даних у хмарному сховищі та запобігає збереженню некоректних документів, що може призводити до помилок десеріалізації або зайвих витрат на зберігання.

```
describe('sanitizeForFirestore', () => {
  it('removes undefined properties from objects', () => {
    const input = { keep: 'value', remove: undefined, nested: { alsoKeep: 123, alsoRemove: undefined } };
  });
});
```

```

    const result = sanitizeForFirestore(input);
    expect(result).toEqual({ keep: 'value', nested: { alsoKeep: 123 } });
  });
});

```

Заключний приклад ілюструє інтеграційне тестування всього аналітичного конвеєра застосунку - від отримання даних з Reddit до формування відповіді LLM. Усі зовнішні сервіси (Firebase, Reddit API, LM Studio) імітуються, що дозволяє повністю контролювати сценарій виконання. Тест перевіряє, що конвеєр завершується без помилок, має статус `done` та коректно формує NLP і LLM результати.

```

describe('pipeline integration', () => {
  it('runs end-to-end with mocked dependencies', async () => {
    const params = { subreddits: 'testing', timeRange: '24h', limit: 5 };
    const result = await pipeline.runPipeline('uid', params);
    expect(result.status).toBe('done');
    expect(result.llm_status).toBe('ok');
  });
});

```

Такий підхід гарантує цілісність системи, зменшує ризики регресій і підтверджує взаємну узгодженість усіх компонентів під час автоматизованого тестування.

3.2 Розгортання застосунку

Розгортання вебзастосунку реалізовано із використанням контейнеризації на основі Docker та оркестрації через Docker Compose. Такий підхід забезпечує ізольоване виконання окремих сервісів, повторюваність середовища та скорочення часу на налаштування програмного комплексу на різних інфраструктурних платформах. До складу розгортання входять три основні компоненти: серверна частина (API), клієнтський інтерфейс (Web) та допоміжний сервіс Redis, який використовується для кешування і керування проміжними даними аналізу.

Архітектурна модель передбачає багатостадійне збирання контейнерів, де процес інсталяції залежностей, компіляції TypeScript-коду та формування продуктивного образу розділено на окремі етапи. Це дозволяє істотно зменшити розмір кінцевого контейнера та підвищити безпеку за рахунок вилучення розробницьких модулів. Для фронтенд-компонента після збирання застосунку у середовищі Node.js статичні ресурси передаються до Nginx, що забезпечує оптимальне обслуговування клієнтських запитів та підтримку маршрутизації односторінкового інтерфейсу.

Мережеві з'єднання між контейнерами здійснюються у межах внутрішньої bridge-мережі, створеної Docker Compose, що усуває потребу у ручному налаштуванні IP-адрес і дозволяє сервісам звертатися один до одного за DNS-іменами. Для зовнішнього доступу порт серверної частини (4000) і клієнтського інтерфейсу (8080) публікуються на хост-систему. Для інтеграції з локальною мовною моделлю (LM Studio) передбачено використання host.docker.internal (для Windows/Mac) або IP-адреси хост-машини (172.17.0.1 для Linux), що забезпечує можливість звернення контейнера до зовнішнього AI-сервера.

Налаштування параметрів розгортання здійснюється через файл середовища (.env), де задаються ключі для Reddit API, Firebase, URL-адреса сервера мовної моделі, а також змінні, пов'язані з портами та конфігурацією логування. Такий підхід виключає жорстке кодування секретних даних у репозиторії та дозволяє швидко перемикатися між середовищами розробки, тестування та експлуатації.

Процес локального розгортання передбачає два можливих сценарії: ручний запуск через Node.js або повноцінне контейнерне розгортання. У другому випадку достатньо виконати команду `docker compose up --build`, після чого автоматично запускаються всі необхідні сервіси, включаючи Redis, API та вебінтерфейс. Контейнери налаштовані з політикою `restart: unless-stopped`, що забезпечує автоматичне відновлення у разі системного збою або перезапуску хосту.

Особливу увагу приділено інтеграції із зовнішніми сервісами. Firebase використовується для автентифікації та збереження даних, а Reddit API - для отримання первинного контенту. Усі взаємодії із цими сервісами відбуваються на серверному рівні, що виключає ризики витоку ключів у браузерному середовищі. Локальна мовна модель LM Studio використовується через REST-інтерфейс, що дозволяє виконувати AI-аналіз без доступу до хмарних сервісів, забезпечуючи автономність та конфіденційність.

Для продуктивного середовища передбачено можливість масштабування серверного компонента та винесення Redis у керований кластер. Запис логів здійснюється у файли з ротацією та дублюванням у Firestore, що полегшує моніторинг і аудит. Задіяні механізми кешування та TTL для NLP-даних зменшують навантаження та скорочують час повторних обчислень.

Таким чином, архітектура розгортання поєднує стандарти контейнеризованих систем із вимогами безпеки та продуктивності, забезпечуючи надійне виконання всіх етапів - від збору інформації до генерації інтерпретованих результатів. Обрана схема розгортання дозволяє як запуск у середовищі дослідника, так і повноцінне використання у виробничій інфраструктурі

3.3 Мануальне тестування

Метою мануального тестування є підтвердження правильності інтеграції фронтенд-, бекенд- і зовнішніх сервісів, а також перевірка стабільності системи під час взаємодії з користувачем, без залучення автоматизованих інструментів тестування.

Процес тестування виконувався за принципом наскрізного проходження типових сценаріїв роботи користувача. Спершу було перевірено процедуру автентифікації через Firebase. Під час входу з коректними обліковими даними система коректно генерувала ID-токен, який використовувався для доступу до

захищених маршрутів API. У разі введення хибного пароля чи відсутності токена користувач отримував повідомлення про помилку з відповідним статус-кодом 401. Це підтвердило правильність реалізації middleware-рівня перевірки автентифікації та стійкість системи до несанкціонованого доступу.

Далі проводилась перевірка сценарію налаштування параметрів аналізу. Користувач вводив сабреддіт, кількість постів і часовий проміжок, після чого натискав кнопку «Start Analysis». Перевірено, що система коректно надсилала HTTP-запит до бекенду, валідувала параметри за допомогою бібліотеки Zod і відображала повідомлення про початок обробки. Було протестовано як стандартні, так і граничні значення параметрів (мінімальні, надмірно великі, некоректні типи), що дозволило впевнитися в надійності механізму обробки помилок та правильності повідомлень у інтерфейсі.

На наступному етапі тестувалася взаємодія з Reddit API. Для перевірки використовувалися реальні субреддіти різної активності. Система стабільно виконувала OAuth2-авторизацію, дотримувалася обмежень Reddit API та зберігала результати запитів у Firestore. Перевірено також поведінку у випадку відсутності мережевого з'єднання - застосунок коректно повідомляв користувача про помилку та пропонував повторити спробу.

Окрему увагу приділено тестуванню етапу попередньої обробки та NLP-аналізу. Для цього використовувалися тексти з різною структурою: короткі пости, великі обговорення, коментарі з емодзі та посиланнями. Під час перевірки підтверджено, що функції очищення тексту видаляють HTML-теги, спеціальні символи та емодзі, визначають мову публікації й видаляють дублікати. Результати NLP-аналізу (частотність слів, n-грами, показники лексичного різноманіття) правильно відображалися у вигляді інтерактивних графіків.

Останнім етапом було тестування інтеграції з локальною мовною моделлю LM Studio. Під час запуску AI-аналізу система надсилала запит із параметрами попереднього NLP-результату, отримувала відповідь у форматі JSON і відображала згенеровані підсумки на дашборді. У разі, коли LM Studio

була вимкнена, система правильно обробляла помилку підключення та пропонувала повторити запит.

Мануальне тестування також охопило перевірку експорту аналітичних даних. Було протестовано формування звітів у форматах CSV, JSON та PDF, причому у всіх випадках результати містили правильні дані, метадані аналізу та часові мітки. Перевірено сумісність з різними браузерами (Chrome, Firefox, Edge) і операційними системами (Windows, macOS, Linux).

Отримані результати підтвердили стабільну роботу всіх функціональних компонентів системи. Жодних критичних збоїв чи втрати даних не зафіксовано. Мануальне тестування показало, що відповідає заявленим функціональним вимогам, забезпечує надійність, коректну взаємодію компонентів і зручність користувацької взаємодії. В таблиці 3.1 наведено опис та статус перевірки тест кейсів.

Таблиця 3.1

Опис та статус перевірки тест кейсів

Назва тест-кейсу	Опис перевірки	Результат
Перевірка автентифікації через Firebase	Вхід із коректними даними - успішний; при помилкових - статус-код 401 і повідомлення про помилку	Пройдено
Перевірка middleware перевірки токена	Система блокує несанкціонований доступ без токена	Пройдено
Налаштування параметрів аналізу	HTTP-запит надсилається, параметри валідуються через Zod, повідомлення про початок обробки відображається	Пройдено
Перевірка граничних значень параметрів	Мінімальні, надмірні й некоректні дані обробляються, повідомлення про помилки коректні	Пройдено

Інтеграція з Reddit API	OAuth2-авторизація стабільна, дотримано лімітів, результати збережено у Firestore	Пройдено
Відсутність мережевого з'єднання	Користувач отримує повідомлення про помилку й можливість повторити спробу	Пройдено
Попередня обробка тексту	Очищення HTML, спецсимволів, емодзі, виявлення мови та видалення дублікатів	Пройдено
NLP-аналіз текстів	Результати частотності, n-грам і лексичного різноманіття коректно відображаються на графіках	Пройдено
Інтеграція з LM Studio	Запит формується, відповідь у JSON, підсумки відображаються на дашборді	Пройдено
Обробка вимкненої LM Studio	Система показує повідомлення про помилку підключення й пропонує повторити запит	Пройдено
Експорт аналітичних даних	Формування CSV, JSON, PDF із правильними даними, метаданими та часовими мітками	Пройдено
Перевірка кросбраузерної сумісності	Робота стабільна у Chrome, Firefox, Edge на Windows, macOS, Linux	Пройдено

3.4 Демонстрація роботи та практичне використання програмного комплексу

Після успішного розгортання компонентів системи та підтвердження їхньої стабільності етапом тестування, доцільно розглянути сценарій практичної взаємодії користувача з вебзастосунком Reddit Content Analyzer.

Інтерфейс системи спроектовано з дотриманням принципів мінімалізму та інтуїтивної зрозумілості, що дозволяє досліднику зосередитися на налаштуванні параметрів аналізу та інтерпретації результатів, не відволікаючись на технічні аспекти роботи бекенду.

Робота із застосунком розпочинається зі сторінки привітання, яка відображена на рисунку 3.1 і одночасно виконує роль шлюзу безпеки. Оскільки система передбачає збереження історії запитів та персональних налаштувань, доступ до функціоналу можливий лише для авторизованих користувачів. Це забезпечує конфіденційність даних та розмежування доступу між різними дослідниками.

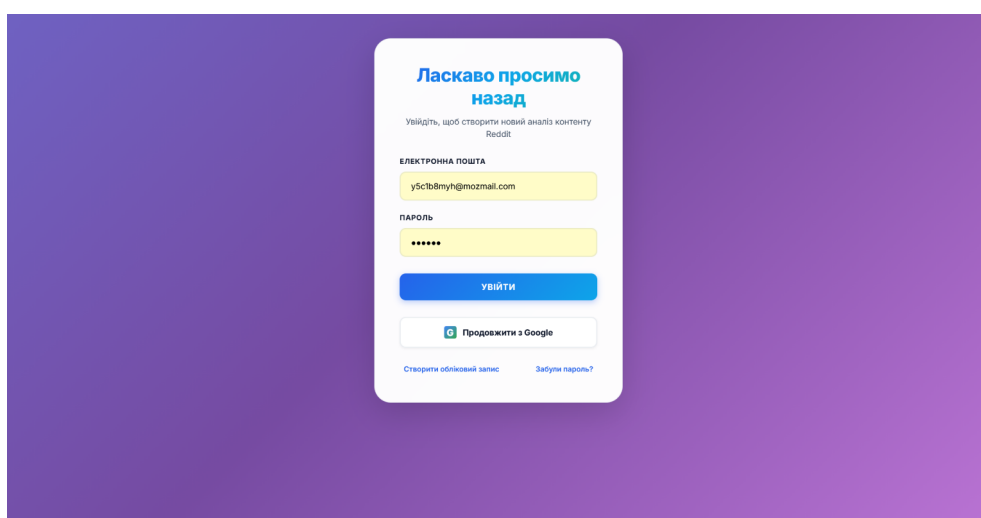


Рисунок 3.1 - Сторінка автентифікації користувача

При першому вході в систему користувачеві пропонується форма авторизації, реалізація якої базується на сервісі Firebase Authentication, що гарантує надійний захист облікових даних. Інтерфейс підтримує два основні сценарії входу, перший з яких передбачає класичну реєстрацію або автентифікацію через Email та пароль. У цьому режимі користувач може створити новий акаунт, скориставшись посиланням «Don't have an account? Sign Up», причому під час реєстрації система автоматично перевіряє валідність введеної електронної пошти та складність пароля. Альтернативним варіантом є швидкий вхід через Google за протоколом OAuth 2.0, коли

натискання кнопки «Continue with Google» дозволяє авторизуватися в системі за допомогою існуючого Google-акаунту без необхідності створення окремого пароля для застосунку.

Після успішної авторизації система перенаправляє користувача до основного робочого простору - сторінки конфігурації параметрів аналізу. Цей інтерфейс спроектовано з метою надання максимальної гнучкості у виборі джерел даних та глибини дослідження.

На сторінці налаштувань (рис. 3.2) користувачеві доступні декілька ключових полів введення. Поле «Сабреддіти / Хештеги» дозволяє задати тематичні спільноти для моніторингу (наприклад, technology, startups), вказуючи їх через кому. Альтернативно, за допомогою поля «Посилання на обговорення Reddit», дослідник може вставити пряму URL-адресу конкретного треду для його детального розбору.

Рисунок 3.2 - Сторінка налаштувань аналізу

Для керування обсягом вибірки передбачено випадаючий список «Часовий діапазон» (на зображенні обрано «Останні 7 днів») та поле «Обмеження», де задається кількість публікацій для обробки (в діапазоні від 1

до 100). Це дозволяє балансувати між швидкістю роботи системи та репрезентативністю даних, а також уникати перевищення лімітів API Reddit.

Важливою особливістю інтерфейсу є перемикач «Увімкнути інсайти штучного інтелекту (LM Studio)». Ця опція активує гібридний режим роботи: якщо прапорець встановлено, після завершення детерміністичного NLP-аналізу система додатково звернеться до локальної мовної моделі для генерації змістовних узагальнень та висновків.

У нижній частині форми розміщено елементи керування процесом: кнопка «Запустити аналіз» ініціює збір та обробку даних, а кнопка «Зберегти шаблон» дозволяє зафіксувати поточні параметри для їх швидкого використання в майбутніх сесіях.

Після завершення процесу збору та обробки даних система автоматично генерує інтерактивну панель керування (рис. 3.3), яка відображає ключові метрики досліджуваного контенту.

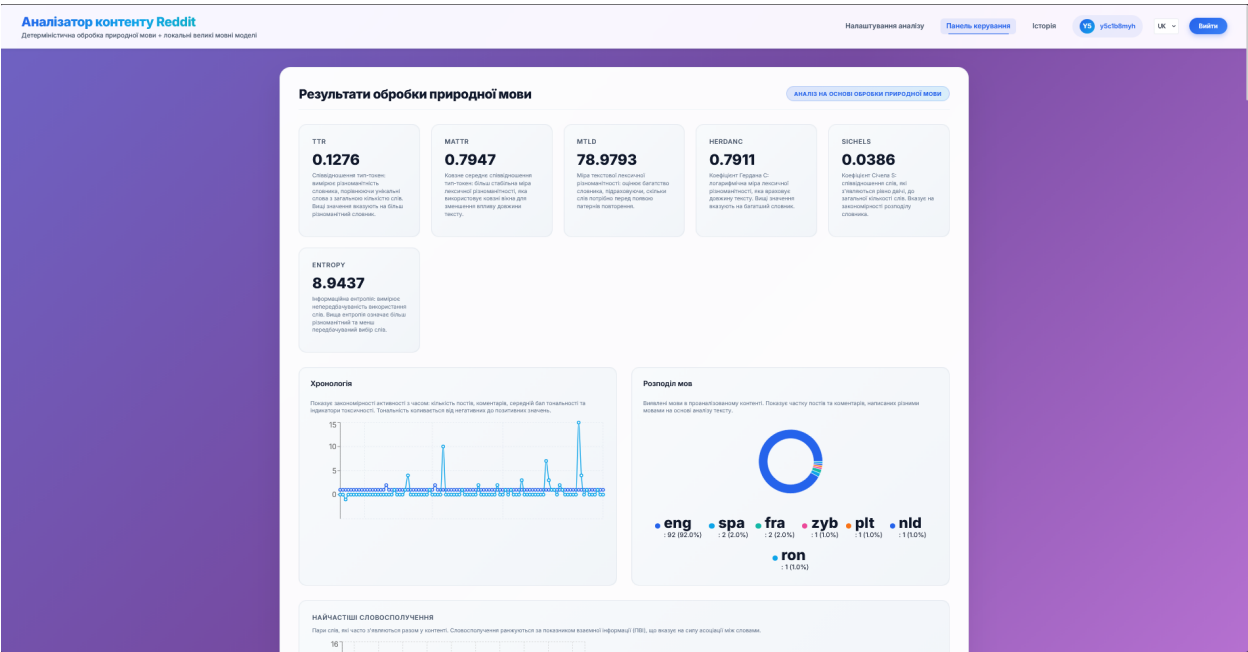


Рисунок 3.3 - Результат обробки природної мови

У верхній частині сторінки («Результати обробки природної мови») користувачеві представлено картки з кількісними показниками лексичної різноманітності. Зокрема, розраховуються коефіцієнти TTR (Type-Token

Ratio), MATTR (Moving-Average TTR) та індекс ентропії, що дозволяє швидко оцінити багатство словникового запасу та інформаційну насиченість дискусії.

Нижче розташовано блок графічної візуалізації, що включає хронологію активності та розподіл мов. Графік «Хронологія» демонструє динаміку публікації постів і коментарів у часі, накладену на криву зміни середньої тональності. Поруч знаходиться діаграма «Розподіл мов», яка візуалізує відсоткове співвідношення контенту різними мовами (наприклад, англійською, іспанською чи французькою), що є важливим для мультикультурного аналізу спільнот Reddit.

Для глибшого розуміння контексту система пропонує блок «Найчастіші словосполучення» (рис. 3.4). Ця діаграма відображає біграми (пари слів), ранжовані за показником взаємної інформації (PMI), що дозволяє виявити стійкі тематичні конструкції, характерні для обраної ніші. Наприклад, у технічних обговореннях можуть виділятися такі пари, як "noise cancellation" або "battery life".

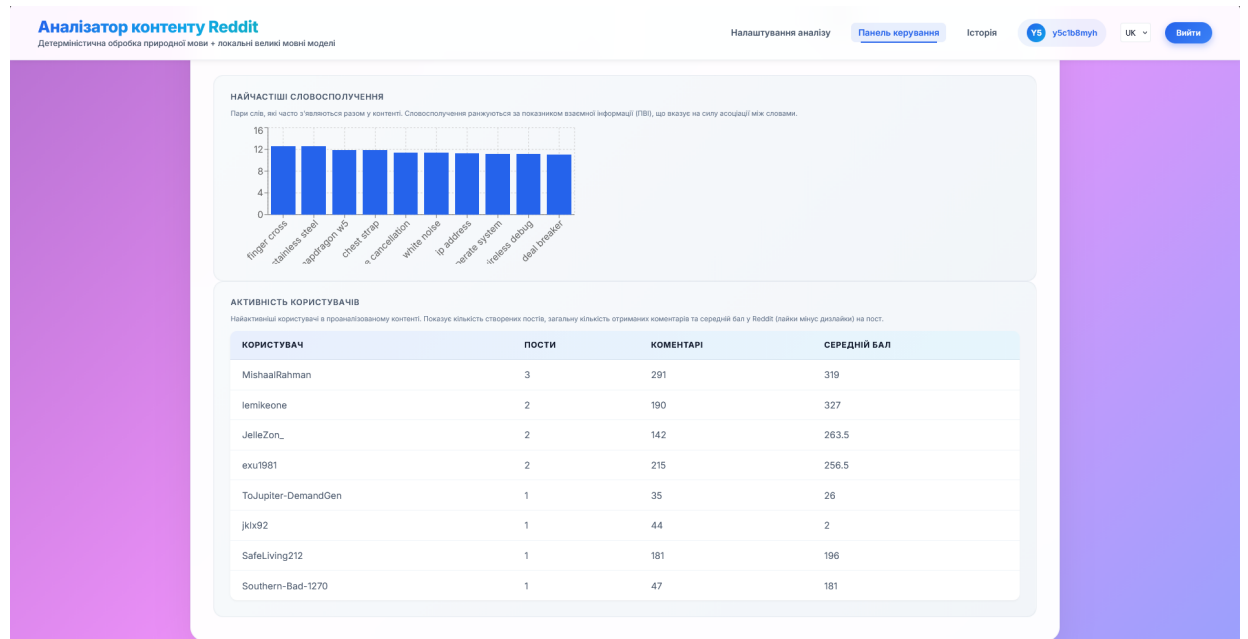


Рисунок 3.4 - Секція найчастіших словосполучень та активності користувачів

У цьому ж секторі розміщено таблицю «Активність користувачів» (рис. 3.4), яка ідентифікує найбільш залучених учасників дискусії. Таблиця містить

дані про кількість створених постів, коментарів та середній рейтинг (score), що дозволяє досліднику виокремлювати лідерів думок або виявляти потенційних ботів у межах досліджуваного сабреддита.

Ключовою особливістю системи є інтеграція з локальною мовною моделлю (LM Studio), результати роботи якої виводяться в окремий блок «Інсайти, згенеровані штучним інтелектом» (рис. 3.5). Замість сухих цифр, система надає структуроване текстове резюме дискусії, виділяє ключові інсайти (наприклад, чутливість користувачів до ціни або проблеми з апаратним забезпеченням) та описує виявлені ризики й упередження.

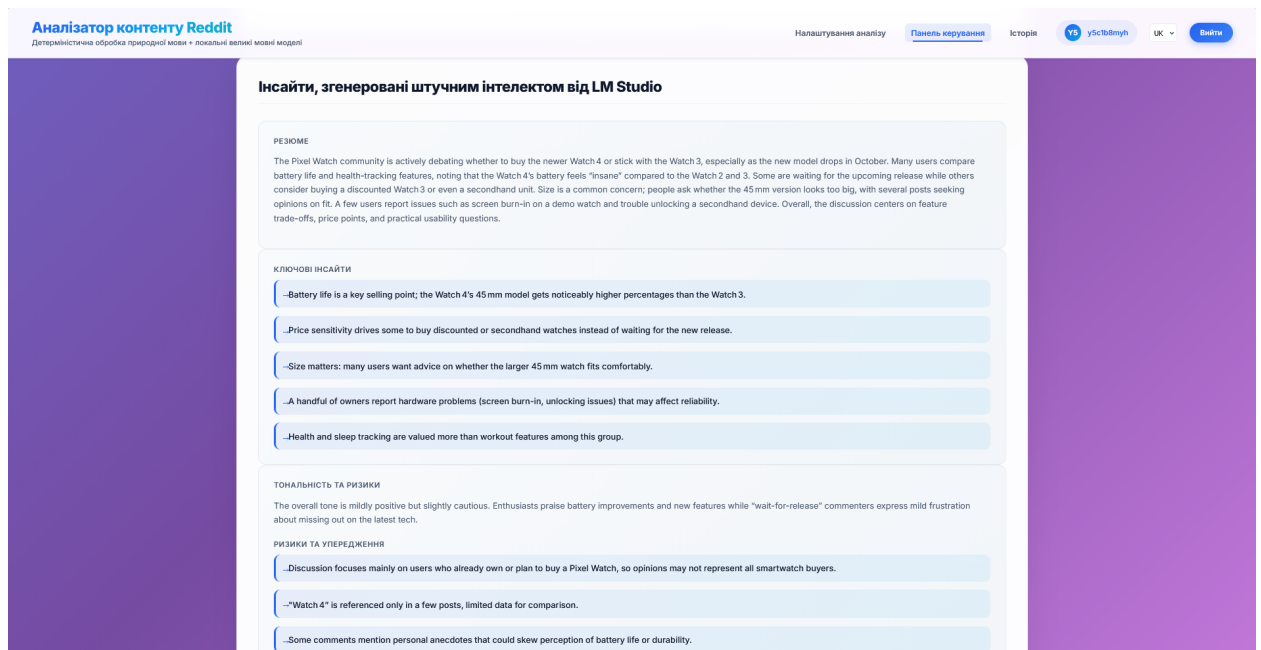


Рисунок 3.5 - Секція інсайтів згенерованих штучним інтелектом

Завершує аналітичний огляд секція «Покращений аналіз тональності» (рис. 3.6), яка поєднує евристичні методи з алгоритмом VADER. Користувач бачить кругову діаграму розподілу настроїв (позитивний, нейтральний, негативний) та порівняльну таблицю точності методів. Також система відображає список конкретних документів із присвоєними їм балами тональності, що дозволяє верифікувати автоматичну оцінку на реальних прикладах тексту.

Окремим важливим елементом є віджет середнього балу тональності, який агрегує загальний емоційний фон вибірки в діапазоні від -1 (різко негативний) до +1 (різко позитивний). Блок порівняння методів наочно демонструє переваги VADER над простим словниковим підходом, показуючи різницю в оцінці складних або неоднозначних висловлювань через зіставлення метрик обох алгоритмів.

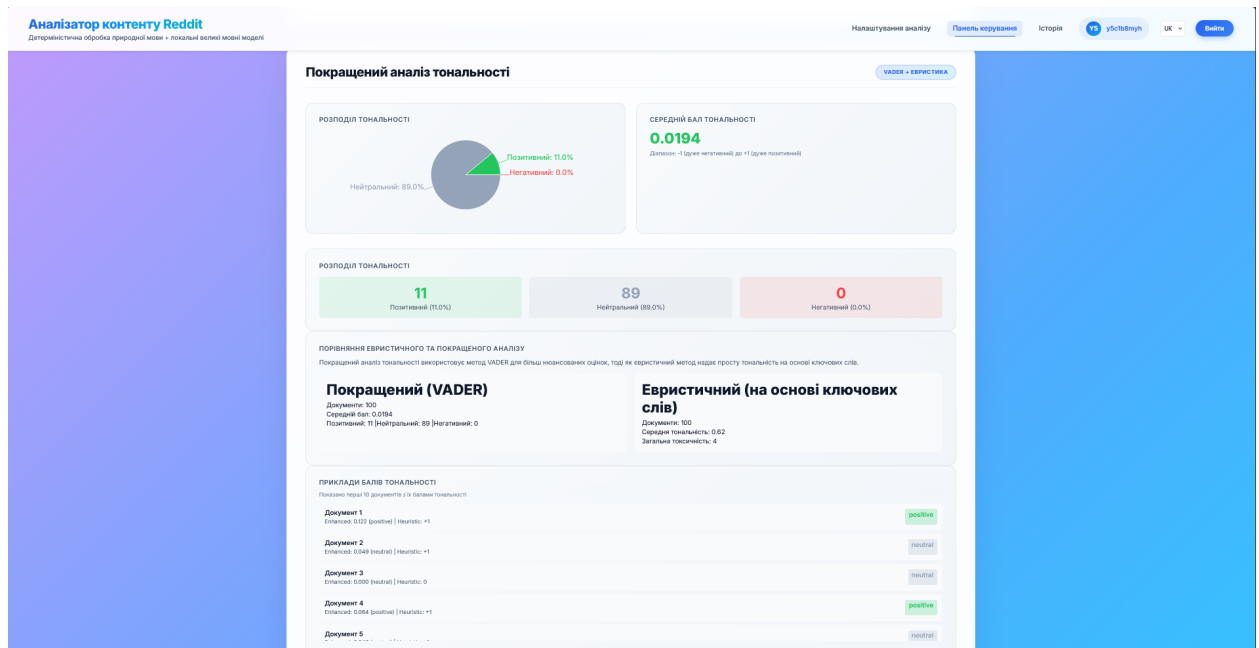


Рисунок 3.6 - Результати генеративного аналізу та оцінки тональності тексту

У найнижчій частині інтерфейсу (або на окремих вкладках, залежно від конфігурації) реалізовано інструменти для глибокого статистичного аналізу: тематичне моделювання (Topic Modeling) на основі LDA та виявлення аномалій (Trend Detection). Система автоматично підсвічує часові мітки, де зафіксовано нетипові сплески активності або різкі зміни тональності. Також доступна теплова карта активності (Activity Heatmap), яка дозволяє візуально оцінити інтенсивність дискусій у розрізі годин доби та днів тижня.

Модуль тематичного моделювання не лише розбиває масив текстів на кластери, але й надає детальну статистику щодо їхньої ваги в загальному дискурсі (рис. 3.7). Для кожної виявленої теми система відображає відсоток покриття документів та список ключових слів із відповідними коефіцієнтами

значущості, що дозволяє швидко ідентифікувати предмет обговорення без необхідності читання всього масиву.

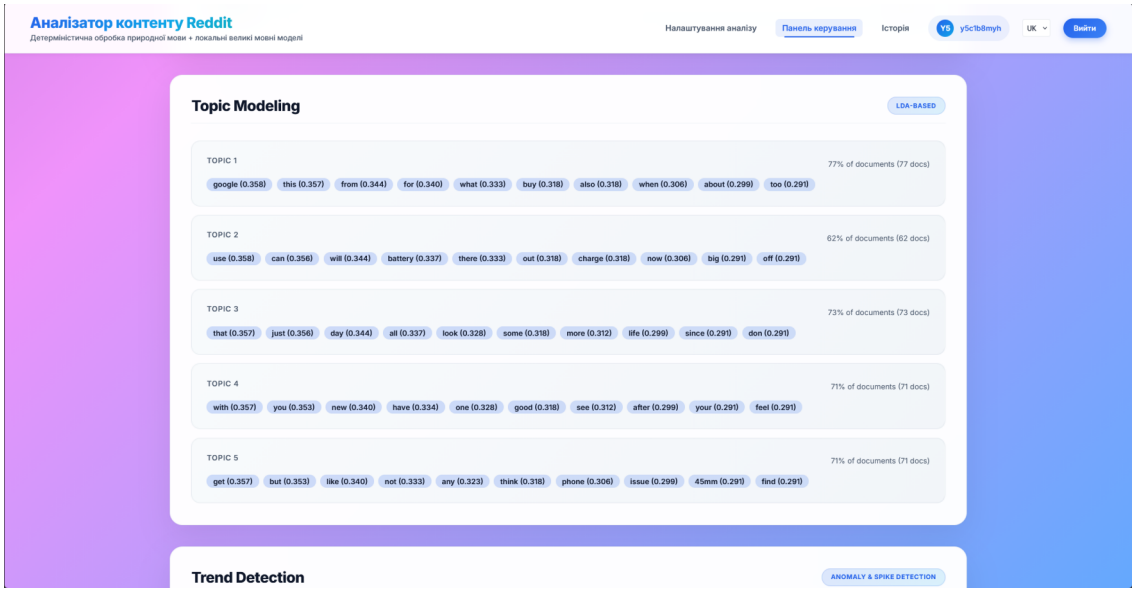


Рисунок 3.7 - Тематичне моделювання

Блок виявлення трендів (рис. 3.8) забезпечує автоматизований моніторинг статистичних відхилень, фіксуючи як аномалії (Anomalies Detected), так і різкі сплески активності (Activity Spikes).

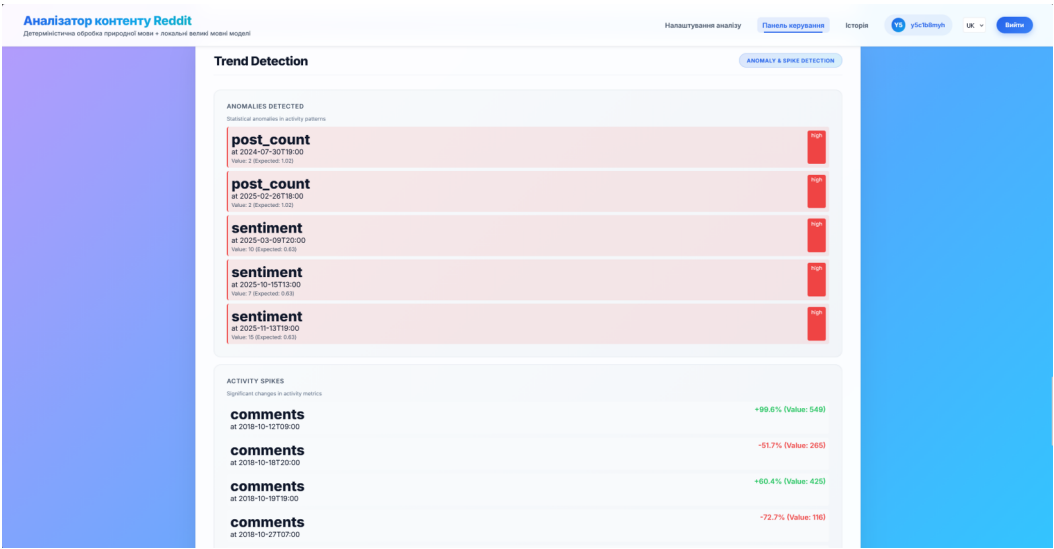


Рисунок 3.8 - Блок виявлення трендів

Для кожної зафіксованої події система надає детальний контекст: часову мітку, тип метрики (кількість постів, коментарів чи тональність), а також

порівняння фактичного значення з очікуваним (наприклад, Value: 10 (Expected: 0.63)), що дозволяє досліднику миттєво реагувати на віральні події або інформаційні атаки.

Нижню частину аналітичної панелі займає секція «Візуалізація частотності слів» (Word Frequency Visualization), яка реалізована у форматі хмари N-грам (рис. 3.9). Цей інструмент дозволяє миттєво ідентифікувати найуживанішу лексику в досліджуваному масиві даних, де розмір кожного слова прямо пропорційний частоті його використання. Такий візуальний підхід допомагає досліднику швидко виокремити домінуючі терміни, ключові об'єкти обговорення (наприклад, «pixel», «watch») та загальну тональність лексики, що формує семантичне ядро дискусії.

Завершує інтерфейс «Теплова карта активності» (Activity Heatmap), призначена для аналізу часових патернів поведінки спільноти (рис. 3.9).

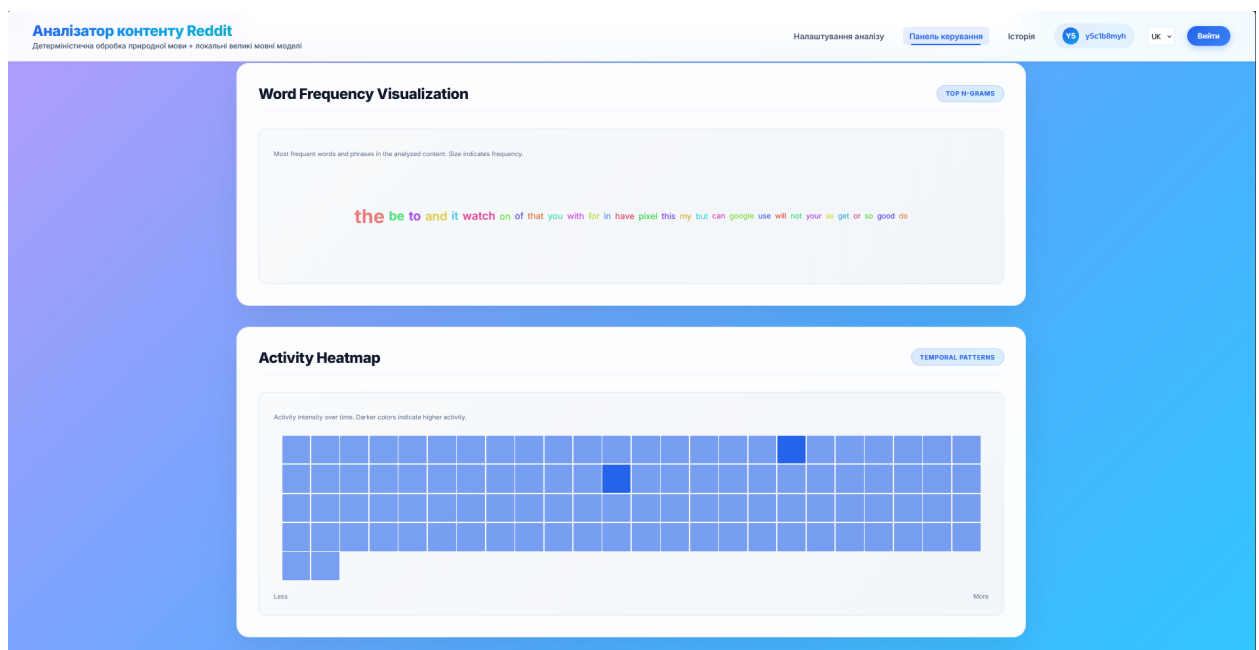


Рисунок 3.9 - Візуалізації частотності слів та теплової карти активності

Графік відображає інтенсивність дискусій у вигляді сітки, де насиченість кольору (від прозорого до темно-синього) вказує на рівень активності в конкретні години доби. Це дозволяє виявляти пікові періоди залученості

аудиторії, визначати оптимальний час для публікацій та розуміти циклічність обговорень у межах досліджуваного сабреддіта.

Завершальним етапом взаємодії з системою є робота з архівом досліджень на сторінці «Історія аналізу» (рис. 3.10). Цей модуль дозволяє користувачеві переглядати, шукати та керувати результатами попередніх сесій, що є критично важливим для проведення лонгітюдних досліджень та порівняльного аналізу в часі. Інтерфейс сторінки обладнано гнучкою системою фільтрації, яка включає поля для введення пошукового запиту, назви сабреддіта, а також віджети для вибору календарного періоду («дд.мм.рррр») створення звіту. Користувач може ініціювати фільтрацію кнопкою «Пошук» або скинути параметри кнопкою «Очистити».

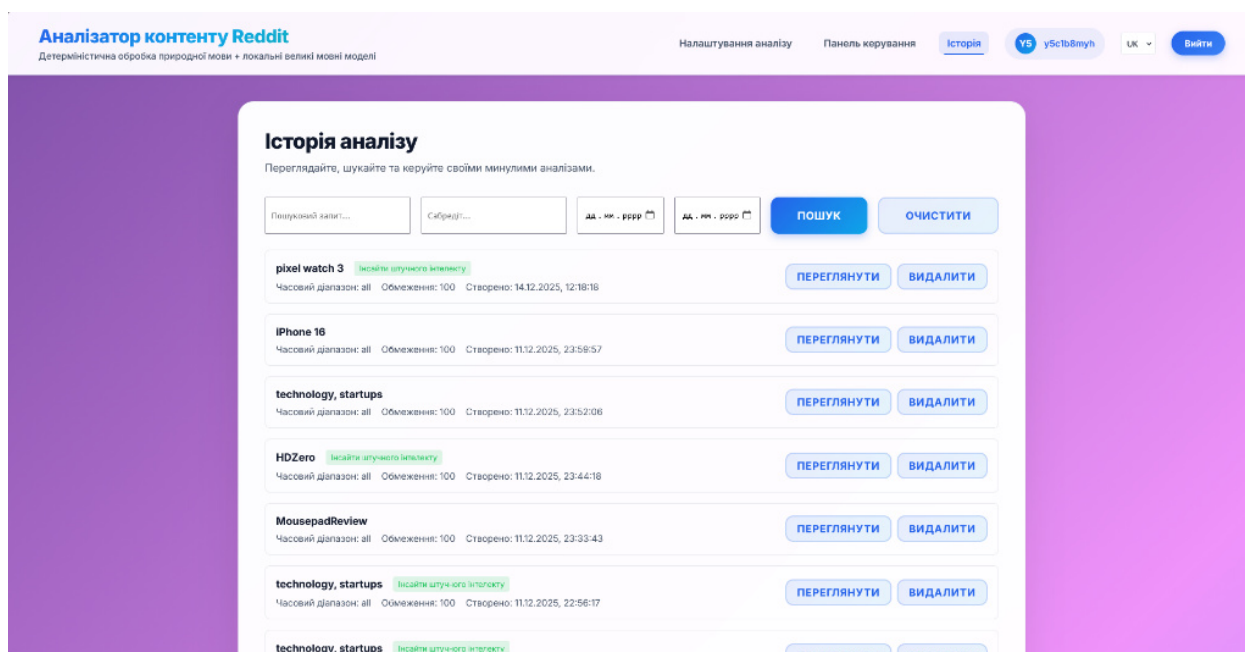


Рисунок 3.10 - Історія аналізу

Центральну частину екрана займає список збережених аналізів, де кожен запис представлено у вигляді інформаційної картки з ключовими метаданими: темою запиту (наприклад, «pixel watch 3» або «iPhone 16»), застосованими обмеженнями вибірки (наприклад, «Обмеження: 100») та точною часовою міткою створення. Особливістю інтерфейсу є наявність візуальних маркерів, таких як тег «Інсайти штучного інтелекту», що дозволяє миттєво відрізнити

звіти, згенеровані з використанням LLM, від виключно статистичних зрізів. Для кожного елемента історії доступні кнопки швидкої дії: «Переглянути», що відновлює деталізований дашборд із результатами, та «Видалити» для очищення застарілих даних з хмарного сховища.

Висновки до розділу 3

У межах третього розділу здійснено комплексну перевірку працездатності, тестування та демонстрацію практичного використання розробленої системи інтелектуального аналізу контенту Reddit.

Проведене автоматизоване тестування підтвердило коректність реалізації ключових алгоритмічних модулів: від фільтрації спаму й очищення текстів до обчислення складних лексичних метрик і генерації звітів. Верифікація роботи middleware-рівня автентифікації засвідчила надійну інтеграцію з Firebase та безпеку доступу до API, а інтеграційні тести конвеєра довели узгодженість передачі даних між усіма етапами обробки.

Мануальне тестування підтвердило, що застосунок успішно реалізує повний життєвий цикл взаємодії користувача із системою в різних браузерах та середовищах. Жодних критичних збоїв при виконанні сценаріїв авторизації, збору даних, NLP-обробки та експорту результатів зафіксовано не було.

Демонстрація практичного використання системи засвідчила високу інформативність та ергономічність розробленого інтерфейсу. Реалізовані на панелі керування (Dashboard) засоби візуалізації - теплові карти активності, хмари n-грам, графіки розподілу мов та тональності - дозволяють досліднику швидко інтерпретувати великі масиви даних. Особливу цінність продемонструвала інтеграція з локальною мовною моделлю LM Studio, яка забезпечує генерацію змістовних текстових інсайтів та резюме, доповнюючи статистичні показники якісним аналізом.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було сформовано цілісне уявлення про процес створення інтегрованої системи для аналізу вебконтенту соціальних платформ із використанням сучасних методів обробки природної мови та великих мовних моделей. У ході дослідження було обґрунтовано актуальність розробки програмного комплексу, здатного автоматично збирати, попередньо обробляти, інтерпретувати та візуалізувати дані з Reddit як представника динамічних і високоволіумних соціальних середовищ. Проведений теоретичний аналіз дозволив визначити ключові виклики, пов'язані з неструктурованістю даних, багатомовністю, проявами токсичності та необхідністю семантичного узагальнення дискусій.

Було досліджено та адаптовано інструментарій NLP, класифікаційні та генеративні моделі, що забезпечують автоматизоване визначення тематичних напрямів, емоційного забарвлення та лексичної різноманітності текстів. Особливу увагу приділено реалізації локального AI-компоненту на основі мовної моделі, що дало змогу досягти автономності системи, зберегти конфіденційність даних і забезпечити керованість процесу генерації висновків. Завдяки поєднанню статистичних методів з генеративною інтерпретацією вдалося сформуванати інструмент, здатний не лише агрегувати числові показники, а й відтворювати аналітичні узагальнення та інсайти.

Архітектурна модель застосунку була реалізована за принципами багаторівневості та модульності, що забезпечило чіткий розподіл відповідальності між клієнтською, серверною, аналітичною та дата-компонентами. Інтеграція з Reddit API, Firebase та локальною LLM була виконана у відповідності з вимогами безпеки, доступності та масштабованості. Запровадження механізмів кешування, логування та тестування сприяло підвищенню надійності системи та забезпечило можливість її подальшого розширення.

Здійснене автоматизоване та мануальне тестування підтвердило коректність реалізації ключових функціональних модулів, стабільність аналізу при різних сценаріях використання та здатність системи коректно поводитися у разі помилок зовнішніх сервісів. Одержані результати демонструють, що програмний комплекс готовий до використання в дослідницьких і прикладних задачах, зокрема для аналізу фінансових настроїв, інформаційних ризиків і соціальних трендів.

Підсумовуючи, виконана робота має як наукове, так і практичне значення. Вона підтвердила ефективність інтеграції великих мовних моделей у системи соціальної аналітики та окреслила перспективи подальшого розвитку у напрямках розширення джерел даних, удосконалення моделей прогнозування та впровадження адаптивних інтерфейсів для експертного аналізу. Створений прототип може слугувати основою для майбутніх досліджень у галузі цифрової гуманітаристики, фінансової аналітики та штучного інтелекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Башманівський М. О., Чижмотря О. В., Дмитренко І. А. Архітектура full-stack застосунку для інтелектуального аналізу Reddit з використанням локальних великих мовних моделей. Тези VIII Всеукраїнської науково-технічної конференції "Комп'ютерні технології: інновації, проблеми, рішення", 02-03 грудня 2025 року. Житомир: "Житомирська політехніка", 2025. (подано до друку).
2. Башманівський М. О., Чижмотря О. В., Дмитренко І. А. Гібридна методологія аналізу контенту Reddit: поєднання класичного NLP та генеративних моделей для узагальнення та виявлення тональності. Тези VIII Всеукраїнської науково-технічної конференції "Комп'ютерні технології: інновації, проблеми, рішення", 02-03 грудня 2025 року. Житомир: "Житомирська політехніка", 2025. (подано до друку).
3. Aggarwal C. C. Machine Learning for Text. Cham : Springer, 2018. 493 p. URL: <https://doi.org/10.1007/978-3-319-73531-3>.
4. Alammar J. Visualizing the Inner Workings of the Transformer. IEEE Transactions on Visualization and Computer Graphics. 2021. Vol. 27, No. 2. pp. 1364–1374. URL: <https://www.scribd.com/document/900179302/The-Illustrated-Transformer-Jay-Alammar-Visualizing-Machine-Learning-One-Concept-at-a-Time>.
5. Bahdanau D., Cho K., Bengio Y. Neural Machine Translation by Jointly Learning to Align and Translate. International Conference on Learning Representations (ICLR). 2015. URL: <https://doi.org/10.48550/arXiv.1409.0473>.
6. Baumgartner J., Zannettou S., Keegan B. The Pushshift Reddit Dataset. Proceedings of the International AAAI Conference on Weblogs and Social Media. 2020. Vol. 14. pp. 830–839. URL: <https://doi.org/10.1609/icwsm.v14i1.7347>.
7. Bommasani R. et al. On the Opportunities and Risks of Foundation Models. arXiv preprint arXiv:2108.07258. 2021. URL: <https://doi.org/10.48550/arXiv.2108.07258>.

8. Brown T. et al. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*. 2020. Vol. 33. pp. 1877–1901. URL: <https://doi.org/10.48550/arXiv.2005.14165>.
9. Bucchiarone A. et al. (eds.) *Microservices: Science and Engineering*. Cham : Springer, 2020. p. 409 URL: <https://doi.org/10.1007/978-3-030-31646-4>
10. Cambria E., Das D., Bandyopadhyay S., Feraco A. *A Practical Guide to Sentiment Analysis*. Cham : Springer, 2017. 190 p. URL: https://www.academia.edu/92830286/A_Practical_Guide_to_Sentiment_Analysis
11. Chen L., Wang Y., Liu W. *Sentiment Analysis in Social Media Data: A Survey*. *IEEE Access*. 2023. Vol. 11. pp. 12345–12360. URL: <https://doi.org/10.1007/s10115-018-1236-4>.
12. Devlin J., Chang M., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of NAACL-HLT*. 2019. pp. 4171–4186. URL: <https://aclanthology.org/N19-1423/>.
13. Dogra V., Verma A. A Comprehensive Survey on Text Summarization Approaches. *Arabian Journal for Science and Engineering*. 2022. Vol. 47. p. 1086. URL: <https://www.sciencedirect.com/science/article/abs/pii/S0957417420305030?via%3Dihub>.
14. Eisenstein J. *Introduction to Natural Language Processing*. Cambridge: MIT Press, 2019. 536 p. URL: <https://mitpress.mit.edu/9780262042840/introduction-to-natural-language-processing/>.
15. Elrom E. *React and Libraries: Your Complete Guide to the React Ecosystem*. Berkeley, CA : Apress, 2021. 368 p. URL: <https://doi.org/10.1007/978-1-4842-6696-0>.
16. Freeman A. *Pro React 18: Build Powerful Web Applications with React*. 5th ed. New York : Apress, 2024. 900 p. URL: <https://doi.org/10.1007/979-8-8688-0284-4>.
17. Goodfellow I., Bengio Y., Courville A. *Deep Learning*. Cambridge : MIT Press, 2016. 800 p. URL: <https://doi.org/10.1007/s10710-017-9314-z>.

18. Han X. Pre-trained models: Past, present and future. *AI Open*. 2021. Vol. 2. pp. 225–250. URL: <https://doi.org/10.1016/j.aiopen.2021.08.002>.
19. Hutto C. J., Gilbert E. VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. *Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media*. 2014. pp. 216–225. URL: <https://doi.org/10.1609/icwsm.v8i1.14550>.
20. Jurafsky D., Martin J. H. *Speech and Language Processing*. 3rd ed. draft. 2024. URL: <https://web.stanford.edu/~jurafsky/slp3/>.
21. Lewis M. et al. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020. pp. 7871–7880. URL: <https://doi.org/10.18653/v1/2020.acl-main.703>.
22. Liu B. *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. 2nd ed. Cambridge : Cambridge University Press, 2020. 450 p. URL: https://doi.org/10.1162/COLI_r_00259.
23. Liu Y. et al. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ACM Computing Surveys*. 2023. Vol. 55, No. 9. pp. 1–35. URL: <https://doi.org/10.1145/3560815>.
24. Luo R., Zhang J., Bo L. Investigation on Sentiment Analysis of Social Media Using Large Language Models. *Proceedings of the 2023 International Conference on Artificial Intelligence*. 2023. pp. 45–52. URL: <https://www.intechopen.com/chapters/88300>.
25. Manaswi N. K. *Deep Learning with Applications Using Python*. Berkeley, CA : Apress, 2018. 235 p. URL: <https://doi.org/10.1007/978-1-4842-3516-4>.
26. Mikolov T. et al. Efficient Estimation of Word Representations in Vector Space. *International Conference on Learning Representations (ICLR)*. 2013. URL: <https://doi.org/10.48550/arXiv.1301.3781>.

27. Minaee S. et al. Deep Learning-based Text Classification: A Comprehensive Review. *ACM Computing Surveys*. 2021. Vol. 54, No. 3. pp. 1–40. URL: <https://doi.org/10.1145/3439726>.
28. Otter D. W., Medina J. R., Kalita J. K. A Survey of the Usages of Deep Learning for Natural Language Processing. *IEEE Transactions on Neural Networks and Learning Systems*. 2021. Vol. 32, No. 2. pp. 604–624. URL: <https://doi.org/10.48550/arXiv.1807.10854>.
29. Ouyang L. et al. Training Language Models to Follow Instructions with Human Feedback. *Advances in Neural Information Processing Systems*. 2022. Vol. 35. p. 68. URL: <https://doi.org/10.48550/arXiv.2203.02155>.
30. Pennington J., Socher R., Manning C. GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014. pp. 1532–1543. URL: <https://doi.org/10.3115/v1/D14-1162>.
31. Proferes N., Jones N., Gilbert S., Fiesler C., Zimmer M. Studying Reddit: A Systematic Overview of Disciplines, Approaches, Methods, and Ethics. *Social Media + Society*. 2021. Vol. 7, No. 2. URL: <https://doi.org/10.1177/20563051211019004>.
32. Qiu X. et al. Pre-trained Models for Natural Language Processing: A Survey. *Science China Technological Sciences*. 2020. Vol. 63, No. 10. pp. 1872–1897. URL: <https://doi.org/10.48550/arXiv.2003.08271>.
33. Rogers A., Kovaleva O., Rumshisky A. A Primer in BERTology: What We Know About How BERT Works. *Transactions of the Association for Computational Linguistics*. 2020. Vol. 8. pp. 842–866. URL: https://doi.org/10.1162/tac1_a_00349.
34. Touvron H. et al. LLaMA: Open and Efficient Foundation Language Models. 2023. URL: <https://doi.org/10.48550/arXiv.2302.13971>.
35. Vaswani A. et al. Attention Is All You Need. *Advances in Neural Information Processing Systems*. 2017. Vol. 30. pp. 5998–6008. URL: <https://doi.org/10.48550/arXiv.1706.03762>.

36. Wei J. et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Advances in Neural Information Processing Systems*. 2022. Vol. 35. p. 43. URL: <https://doi.org/10.48550/arXiv.2201.11903>.
37. Wolf T. et al. Transformers: State-of-the-Art Natural Language Processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 2020. pp. 38–45. URL: <https://doi.org/10.18653/v1/2020.emnlp-demos.6>.
38. Zambon G. *Practical Software Architecture: Moving from System Context to Deployment*. Berkeley, CA : Apress, 2021. 305 p. URL: <https://doi.org/10.1007/978-1-4842-6963-3>.
39. Zhao W. A Survey of Large Language Models. 2023. 144 p. URL: <https://doi.org/10.48550/arXiv.2303.18223>.
40. Zhu X., Goldberg A. B. *Introduction to Semi-Supervised Learning*. *Synthesis Lectures on Artificial Intelligence and Machine Learning*. 2009. Vol. 3, No. 1. pp. 1–130. URL: <https://doi.org/10.2200/S00196ED1V01Y200906AIM006>.

ДОДАТКИ

Код алгоритму попередньої обробки та нормалізації текстових даних

```

import { Response } from 'express';
import { analysisRequestSchema } from '../types/analysis';
import { AuthedRequest } from '../middleware/verifyFirebase';
import { runPipeline } from '../services/pipelineService';
import { callLLM } from '../services/llmService';
import { appendLogEntry, listAnalyses, searchAnalyses } from
'../services/firestoreService';
import { exportToCSV, exportToJSON, exportToPDF } from '../services/exportService';
import { structuredLog } from '../utils/logger';
import { firestore as getFirestore } from '../config/firebase';

export const startAnalysis = async (req: AuthedRequest, res: Response) => {
  const parse = analysisRequestSchema.safeParse(req.body);
  if (!parse.success) {
    return res.status(400).json({
      error: { code: 'VALIDATION_ERROR', message: parse.error.message },
    });
  }

  try {
    const response = await runPipeline(req.uid!, parse.data);
    structuredLog('info', 'Analysis pipeline response sent', {
      context: 'pipeline',
      code: 'analysis.response',
      uid: req.uid,
      meta: { analysisId: response.analysisId },
    });
    return res.json(response);
  } catch (error) {
    return res.status(500).json({
      error: { code: 'UPSTREAM_ERROR', message: (error as Error).message },
    });
  }
};

export const retryLLM = async (req: AuthedRequest, res: Response) => {
  const { analysisId } = req.body as { analysisId?: string };
  if (!analysisId) {
    return res.status(400).json({ error: { code: 'VALIDATION_ERROR', message:
'analysisId required' } });
  }
  const db = getFirestore();
  const ref = db.doc(`users/${req.uid}/analyses/${analysisId}`);
  const snapshot = await ref.get();
  if (!snapshot.exists) {

```

```

        return res.status(404).json({ error: { code: 'VALIDATION_ERROR', message:
'Analysis not found' } });
    }
    const data = snapshot.data();
    const preprocessed = data?.preprocessed?.docs ?? [];
    const raw = data?.raw?.posts ?? [];
    const nlp = data?.nlp;
    if (!nlp) {
        return res.status(400).json({ error: { code: 'UPSTREAM_ERROR', message: 'Missing
NLP results' } });
    }
    const { llm, status } = await callLLM(analysisId, nlp, preprocessed, raw);
    await ref.set(
        {
            llm,
            meta: { llm_status: status },
        },
        { merge: true }
    );
    return res.json({ analysisId, llm, llm_status: status });
};

export const getAnalysis = async (req: AuthedRequest, res: Response) => {
    const { id } = req.params;
    const db = getFirestore();
    const snapshot = await db.doc(`users/${req.uid}/analyses/${id}`).get();
    if (!snapshot.exists) {
        return res.status(404).json({ error: { code: 'VALIDATION_ERROR', message:
'Analysis not found' } });
    }
    return res.json(snapshot.data());
};

export const logUiEvent = async (req: AuthedRequest, res: Response) => {
    const { analysisId, code, message } = req.body as {
        analysisId: string;
        code: string;
        message?: string;
    };
    if (!analysisId || !code) {
        return res.status(400).json({ error: { code: 'VALIDATION_ERROR', message:
'analysisId and code required' } });
    }
    await appendLogEntry(req.uid!, analysisId, {
        ts: new Date().toISOString(),
        level: 'info',
        context: 'ui',
        code,
        msg: message ?? "",
    });
    return res.json({ status: 'logged' });
};

```

```

export const exportAnalysis = async (req: AuthedRequest, res: Response) => {
  const { id, format } = req.params as { id: string; format: 'csv' | 'json' | 'pdf' };
  const db = getFirestore();
  const snapshot = await db.doc(`users/${req.uid}/analyses/${id}`).get();
  if (!snapshot.exists) {
    return res.status(404).json({ error: { code: 'VALIDATION_ERROR', message:
'Analysis not found' } });
  }
  const data = snapshot.data() as any;
  if (!data?.nlp) {
    return res.status(400).json({ error: { code: 'UPSTREAM_ERROR', message: 'Missing
analysis results' } });
  }
  if (format === 'csv') {
    const csv = exportToCSV(id, data.nlp, data.llm ?? null);
    res.setHeader('Content-Type', 'text/csv');
    res.setHeader('Content-Disposition', `attachment; filename="analysis-${id}.csv"`);
    return res.send(csv);
  }
  if (format === 'json') {
    const json = exportToJSON(id, data);
    res.setHeader('Content-Type', 'application/json');
    return res.send(json);
  }
  if (format === 'pdf') {
    const pdfPath = await exportToPDF(id, { nlp: data.nlp, llm: data.llm ?? null });
    res.setHeader('Content-Type', 'application/pdf');
    res.setHeader('Content-Disposition', `attachment; filename="analysis-${id}.pdf"`);
    return res.sendFile(pdfPath);
  }
  return res.status(400).json({ error: { code: 'VALIDATION_ERROR', message:
'Unsupported format' } });
};

export const listAnalysisHistory = async (req: AuthedRequest, res: Response) => {
  try {
    const limit = parseInt(req.query.limit as string) || 20;
    const offset = parseInt(req.query.offset as string) || 0;
    const analyses = await listAnalyses(req.uid!, limit, offset);
    return res.json(analyses);
  } catch (error) {
    return res.status(500).json({
      error: { code: 'UPSTREAM_ERROR', message: (error as Error).message },
    });
  }
};

export const searchAnalysisHistory = async (req: AuthedRequest, res: Response) => {
  try {
    const { query, subreddit, dateFrom, dateTo } = req.query as {
      query?: string;

```

```

    subreddit?: string;
    dateFrom?: string;
    dateTo?: string;
  };
  const analyses = await searchAnalyses(req.uid!, {
    query,
    subreddit,
    dateFrom,
    dateTo,
  });
  return res.json(analyses);
} catch (error) {
  return res.status(500).json({
    error: { code: 'UPSTREAM_ERROR', message: (error as Error).message },
  });
}
};

export const deleteAnalysis = async (req: AuthedRequest, res: Response) => {
  const { id } = req.params;
  const db = getFirestore();
  const ref = db.doc(`users/${req.uid}/analyses/${id}`);
  const snapshot = await ref.get();
  if (!snapshot.exists) {
    return res.status(404).json({ error: { code: 'VALIDATION_ERROR', message:
'Analysis not found' } });
  }
  await ref.delete();
  structuredLog('info', 'Analysis deleted', {
    context: 'pipeline',
    code: 'analysis.delete',
    uid: req.uid,
    analysisId: id,
  });
  return res.json({ status: 'deleted' });
};

```

Код збору даних з офіційного Reddit API

```

import axios from 'axios';
import { structuredLog } from '../utils/logger';
import { env } from '../config/env';
import { RawRedditPost } from '../types/analysis';

interface RedditTokenResponse {
  access_token: string;
  token_type: string;
  expires_in: number;
  scope: string;
}

const redditOAuth = axios.create({
  baseURL: 'https://www.reddit.com/api/v1',
});

const redditApi = axios.create({
  baseURL: 'https://oauth.reddit.com',
});

let tokenCache: { token: string; expiresAt: number } | null = null;

export const convertTimeRange = (timeRange: '24h' | '7d' | '30d' | 'all'): {
  redditApiFormat: 'hour' | 'day' | 'week' | 'month' | 'year' | 'all';
  cutoffTimestamp: number | null;
} => {
  const now = Math.floor(Date.now() / 1000);

  switch (timeRange) {
    case '24h':
      return { redditApiFormat: 'day', cutoffTimestamp: now - 24 * 60 * 60 };
    case '7d':
      return { redditApiFormat: 'week', cutoffTimestamp: now - 7 * 24 * 60 * 60 };
    case '30d':
      return { redditApiFormat: 'month', cutoffTimestamp: now - 30 * 24 * 60 * 60 };
    case 'all':
      return { redditApiFormat: 'all', cutoffTimestamp: null };
    default:
      return { redditApiFormat: 'all', cutoffTimestamp: null };
  }
};

const ensureToken = async () => {
  const now = Date.now();
  if (tokenCache && tokenCache.expiresAt > now + 60_000) {
    return tokenCache.token;
  }

```

```

const credentials = `${env.redditClientId}:${env.redditSecret}`;
const authHeader = Buffer.from(credentials).toString('base64');

try {
  const response = await redditOAuth.post<RedditTokenResponse>(
    '/access_token',
    new URLSearchParams({
      grant_type: 'client_credentials',
    }).toString(),
    {
      headers: {
        Authorization: `Basic ${authHeader}`,
        'User-Agent': env.redditUserAgent,
        'Content-Type': 'application/x-www-form-urlencoded',
      },
    },
  );
  tokenCache = {
    token: response.data.access_token,
    expiresAt: now + response.data.expires_in * 1000,
  };
  structuredLog('info', 'Obtained Reddit API token', {
    context: 'reddit',
    code: 'reddit.token',
  });
  return tokenCache.token;
} catch (error) {
  structuredLog('error', 'Failed to obtain Reddit token', {
    context: 'reddit',
    code: 'reddit.token.error',
    meta: { message: (error as Error).message },
  });
  throw error;
}
};

export const fetchSubredditPosts = async (
  subreddit: string,
  limit: number,
  timeRange: '24h' | '7d' | '30d' | 'all' = 'all'
): Promise<RawRedditPost[]> => {
  const token = await ensureToken();
  try {
    const cleanSubreddit = subreddit.replace(/^\r\//, "").trim();
    const { redditApiFormat, cutoffTimestamp } = convertTimeRange(timeRange);

    const params: any = { limit };
    if (redditApiFormat !== 'all') {
      params.t = redditApiFormat;
    }
  }

```

```

const response = await redditApi.get(`/r/${cleanSubreddit}/hot`, {
  params,
  headers: {
    Authorization: `Bearer ${token}`,
    'User-Agent': env.redditUserAgent,
  },
});
structuredLog('info', 'Reddit posts fetched', {
  context: 'reddit',
  code: 'reddit.fetch.success',
  meta: { subreddit: cleanSubreddit, count: response.data?.data?.children?.length ?? 0 },
});
const posts = (response.data?.data?.children ?? []).
  .map((child: any) => child.data)
  .map((item: any): RawRedditPost => ({
    id: item.id,
    author: item.author,
    title: item.title,
    text: item.selftext ?? "",
    score: item.score,
    num_comments: item.num_comments,
    created_utc: item.created_utc,
    subreddit: item.subreddit,
    permalink: `https://reddit.com${item.permalink}`,
  })))
  .filter((post: RawRedditPost) => cutoffTimestamp === null || post.created_utc >=
cutoffTimestamp);

return posts;
} catch (error: any) {
  const status = error.response?.status;
  if (status === 404) {
    structuredLog('error', 'Subreddit not found', {
      context: 'reddit',
      code: 'reddit.subreddit.not_found',
      meta: { subreddit, message: `Subreddit "${subreddit}" does not exist or is
inaccessible` },
    });
    throw new Error(`Subreddit "${subreddit}" not found. It may not exist, be private, or
be banned.`);
  }
  if (status === 403) {
    structuredLog('error', 'Subreddit access forbidden', {
      context: 'reddit',
      code: 'reddit.subreddit.forbidden',
      meta: { subreddit, message: `Subreddit "${subreddit}" is forbidden. It may be private,
quarantined, or require authentication.` },
    });
    throw new Error(`Subreddit "${subreddit}" access forbidden. It may be private,
quarantined, or require authentication.`);
  }
}

```

```

    if (status === 429) {
      structuredLog('warn', 'Reddit rate limited', {
        context: 'reddit',
        code: 'reddit.rate_limited',
        meta: { subreddit },
      });
    }
    structuredLog('error', 'Reddit fetch error', {
      context: 'reddit',
      code: 'reddit.fetch.error',
      meta: { subreddit, message: error.message, status },
    });
    throw error;
  }
};

export const searchReddit = async (
  query: string,
  subreddit?: string,
  limit: number = 25,
  sort: 'relevance' | 'hot' | 'top' | 'new' | 'comments' = 'relevance',
  timeRange: '24h' | '7d' | '30d' | 'all' = 'all'
): Promise<RawRedditPost[]> => {
  const token = await ensureToken();
  try {
    const endpoint = subreddit ? ` /r/${subreddit}/search` : '/search';

    const { redditApiFormat } = convertTimeRange(timeRange);
    const params: any = {
      q: query,
      limit,
      sort,
      restrict_sr: subreddit ? 'true' : 'false',
    };

    if ((sort === 'top' || sort === 'hot') && redditApiFormat !== 'all') {
      params.t = redditApiFormat;
    }

    const response = await redditApi.get(endpoint, {
      params,
      headers: {
        Authorization: `Bearer ${token}`,
        'User-Agent': env.redditUserAgent,
      },
    });

    const { cutoffTimestamp } = convertTimeRange(timeRange);
    const posts = (response.data?.data?.children ?? [])
      .map((child: any) => child.data)
      .filter((item: any) => item && item.id)

```

```

.map((item: any): RawRedditPost => ({
  id: item.id,
  author: item.author,
  title: item.title,
  text: item.selftext ?? "",
  score: item.score,
  num_comments: item.num_comments,
  created_utc: item.created_utc,
  subreddit: item.subreddit,
  permalink: `https://reddit.com${item.permalink}`,
}))
.filter((post: RawRedditPost) => cutoffTimestamp === null || post.created_utc >=
cutoffTimestamp);

structuredLog('info', 'Reddit search completed', {
  context: 'reddit',
  code: 'reddit.search.success',
  meta: { query, subreddit: subreddit || 'all', count: posts.length },
});

return posts;
} catch (error: any) {
  const status = error.response?.status;
  if (status === 429) {
    structuredLog('warn', 'Reddit rate limited', {
      context: 'reddit',
      code: 'reddit.rate_limited',
      meta: { query, subreddit },
    });
  }
  structuredLog('error', 'Reddit search error', {
    context: 'reddit',
    code: 'reddit.search.error',
    meta: { query, subreddit, message: error.message },
  });
  throw error;
}
};

export const fetchThread = async (permalink: string): Promise<RawRedditPost[]> => {
  const token = await ensureToken();
  try {
    const response = await redditApi.get(permalink.replace('https://www.reddit.com', ''), {
      headers: {
        Authorization: `Bearer ${token}`,
        'User-Agent': env.redditUserAgent,
      },
    });
  }
  const posts: RawRedditPost[] = [];
  const [postListing, commentsListing] = response.data;
  const postData = postListing.data.children[0].data;
  posts.push({

```

```

    id: postData.id,
    author: postData.author,
    title: postData.title,
    text: postData.selftext ?? "",
    score: postData.score,
    num_comments: postData.num_comments,
    created_utc: postData.created_utc,
    subreddit: postData.subreddit,
    permalink: `https://reddit.com${postData.permalink}`,
  });

```

```

const flattenComments = (listing: any) => {
  if (!listing?.data?.children) return;
  listing.data.children.forEach((child: any) => {
    if (child.kind === 't1') {
      const data = child.data;
      posts.push({
        id: data.id,
        author: data.author,
        title: posts[0].title,
        text: data.body ?? "",
        score: data.score,
        num_comments: data.replies?.data?.children?.length ?? 0,
        created_utc: data.created_utc,
        subreddit: data.subreddit ?? posts[0].subreddit,
        permalink: `https://reddit.com${data.permalink}`,
      });
      flattenComments(data.replies);
    }
  });
};

```

```

flattenComments(commentsListing);

```

```

structuredLog('info', 'Reddit thread fetched', {
  context: 'reddit',
  code: 'reddit.fetch.success',
  meta: { permalink, count: posts.length },
});

```

```

return posts;
} catch (error: any) {
  structuredLog('error', 'Failed to fetch thread', {
    context: 'reddit',
    code: 'reddit.fetch.error',
    meta: { permalink, message: error.message },
  });
  throw error;
}
};

```