НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____Олександр КОВАЛЬ
«_____»_____202_р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці» спеціальності 121 Інженерія програмного забезпечення на тему: «Комплексний аналіз настроїв тексту за допомогою штучного інтелекту»

Виконав (-ла):
студент (-ка) IV курсу, групи ТВ_з11
Шарабура Еліна Дмитрівна_____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник:
проф. каф. ІПЗЕ, д.т.н., доц., Мусієнко А. П _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:
доц. каф. ЦТЕ, к.ф.-м., доц., Донець А. Г. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
(підпис)

«____»_____202_р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Шарабурі Еліні Дмитрівні_____
(прізвище, ім'я, по батькові)

1.Тема роботи «Комплексний аналіз настроїв тексту за допомогою штучного інтелекту» _____
керівник роботи Мусієнко Андрій Петрович, д.т.н., доц.___ _____ _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "__"_____202_року №___

2. Строк подання студентом роботи ___9 червня 2025р._____ _____

3. Вихідні дані до роботи: мова програмування Python, середовище розробки Visual Studio Code, база даних Firebase Firestore (NoSQL), фреймворк Flask, клієнтська частина HTML, CSS, JavaScript, трансформер-моделі від Hugging Face_____

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити): провести аналіз існуючих систем для аналізу тексту з використанням штучного інтелекту; розробити власну вебсистему для комплексного аналізу настроїв тексту з використанням штучного інтелекту; розробити систему обробки, передачі даних на сервер та аналізу даних за допомогою інтелектуальних моделей; провести тестування та оцінку ефективності вебсистеми.

5. Перелік ілюстративного матеріалу
6. Дата видачі завдання «31» жовтня 2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	30.10.2024	виконано
2	Дослідження предметної області	31.10.2024 – 01.11.2024	виконано
3	Постановка вимог до проєктування системи	02.11.2024 – 01.12.2024	виконано
4	Дослідження існуючих рішень	02.12.2024 – 12.01.2025р.	виконано
5	Розробка програмного продукту	13.01.2025 – 11.05.2025	виконано
6	Тестування	12.05.2025 – 15.05.2025	виконано
7	Захист програмного продукту	12.05.2025 – 15.05.2025	виконано
8	Оформлення дипломної роботи	19.05.2025 – 01.06.2025	виконано
9	Передзахист	02.06.2025 – 06.06.2025	виконано
10	Захист	16.06.2025 – 27.06.2025	виконано

Студент _____ Еліна ШАРАБУРА
(підпис) (ім’я, прізвище)

Керівник роботи _____ Андрій МУСІЄНКО
(підпис) (ім’я, прізвище)

РЕФЕРАТ

Метою роботи є розробка інтелектуальної вебсистеми для комплексного аналізу настроїв тексту. Для досягнення поставленої мети виконано такі завдання: розбір проблем пов’язаних з аналізом тексту, проектування та розробка вебсервісу, включаючи фронтенд та бекенд, реалізація взаємодії серверу з моделями штучного інтелекту, тестування сервісу.

Практичне значення одержаних результатів полягає в отриманні вебсервісу, що покращує ефективність аналізу тексту за настроєм та токсичністю і класифікує його за тематикою. В результаті створено необхідну документацію та отримано готовий вебсервіс. Робота містить 56 сторінок, 28 рисунків, 2 додатки, 15 посилань.

Ключові слова: вебсервіс, інтелектуальні системи, штучний інтелект, аналіз настроїв, аналіз тексту, класифікація, обробка природної мови.

ABSTRACT

The aim of this work is to develop an intelligent web system for comprehensive sentiment analysis of text. To achieve this goal, the following tasks were accomplished: analysis of issues related to text processing, design and development of a web service including both frontend and backend, implementation of server interaction with artificial intelligence models, and testing of the service.

The practical value of the obtained results lies in the creation of a web service that enhances the efficiency of sentiment and toxicity analysis and classifies text by topic. As a result, the necessary documentation was prepared, and a fully functional web service was developed. The work contains 56 pages, 28 figures, 2 appendices, and 15 references.

Keywords: web service, intelligent systems, artificial intelligence, sentiment analysis, text analysis, classification, natural language processing.

ЗМІСТ

ВСТУП

Отримуючи кожного дня гігабайти інформації людина не здатна детально проаналізувати весь потік контенту, який споживає. Крім того, мозок має обмеження на миттєву та об’єктивну обробку великих обсягів даних. Тут на допомогу приходить автоматизація цього процесу. Автоматизувати такий процес можна за допомогою створення інтелектуальної системи для аналізу текстової інформації. Це буде швидко та без суб’єктивного сприйняття. Оскільки це актуально, то компанії інтегрують такі системи у свої робочі процеси, а саме фільтрацію токсичного та негативного контенту, класифікацію запитів користувачів за тематикою та їх поведінкою. Вбудовувати таку систему у процеси можна у вебсайти, мобільні застосунки, використовувати локально або по API. Також доступна кастомізація під багатомовність, тональності, форми подачі та рівні чутливості.

Ця робота присвячена розробці інтелектуального вебсервісу, який дозволяє отримати детальний аналіз настріїв текстової інформації за кількома параметрами: визначення тональності, токсичності з можливістю зміни її рівня, тематичної приналежності, а також автоматичне узагальнення змісту тексту. У системі реалізовано базовий функціонал аналізу без авторизації, та розширений функціонал для зареєстрованих користувачів, включно з історією аналізів, обробкою текстових та аудіо файлів та можливістю перегенерувати текст з іншим рівнем токсичності.

Метою проєкту є створення функціонального, інтуїтивного та технічно якісного вебінструменту, який поєднує сучасні технології штучного інтелекту, зокрема моделі типу трансформерів, з професійним інтерфейсом користувача.

До основних завдань належать:

– аналіз існуючих рішень та вибір відповідної архітектури системи;

– реалізація серверної частини на Flask із підключенням попередньо завантажених моделей від Hugging Face Transformers;

– створення адаптивного вебінтерфейсу з інтерактивними елементами;

– забезпечення функціональності збереження результатів, завантаження файлів і трансформації тексту за допомогою OpenAI API.

У результаті реалізовано повнофункціональну інтелектуальну програмну вебсистему, яка демонструє можливості глибокого аналізу текстових даних.

1 ПОСТАНОВКА ЗАДАЧІ ПОБУДОВИ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ВЕБАНАЛІЗУ ТЕКСТІВ

1.1 Поняття, мета та значення аналізу текстової інформації

Текстовою інформацією вважається представлений набір символів, який має смислове навантаження та підлягає логічному, емоційному або тематичному аналізу. Становить більшу частину комунікацій в інтернеті включаючи соціальні мережі, відгуки, новини, електронні пошти.

У новітніх інформаційних системах використовується як основний спосіб передачі змісту між машиною та користувачем, для збереження та аналізу інформації. Завдяки аналізу текстової інформації можна виявити емоційну тональність, наміри та потреби користувачів, саме тому це широко застосовується у сферах підтримки клієнтів, реклами, штучного інтелекту, наукової діяльності.

Аналізом текстової інформації є процес у ході якого автоматизовано або неавтоматизовано можна виділити зміст, структуру, емоційне забарвлення, тему та інші характеристики. За умови, якщо такий процес відбувається автоматизовано, а саме за участі комп'ютерної системи, то його реалізація відбувається за допомогою методів обробки природної мови (NLP – Natural Language Processing). Natural Language Processing – напрям штучного інтелекту, який дозволяє інтерпретувати тексти написані людиною. Автоматизований аналіз дозволяє системам автоматично генерувати рішення та реакції без втручання людини у такий процес. Оскільки обсяги інформації онлайн щоденно збільшують свої масштаби у геометричній прогресії, внаслідок масової діджиталізації, то виникає потреба в інтеграції автоматизованих підходів до її обробки. Системи повинні обробляти тисячі повідомлень щосекунди, тому автоматизований підхід не лише доцільне спрощення, а необхідне рішення для більшості сучасних застосунків та сервісів. З огляду на потребу в глибокому автоматизованому аналізу текстової інформації є

необхідність виділити характеристики за якими такі тексти варто класифікувати. Зокрема, їхню емоційну складову, тематику, та рівень токсичності, саме ці параметри вважаються ключовими в обробці текстової інформації.

1.2 Класифікація текстів за емоційністю, тематикою та рівнем токсичності

Потреба класифікації тексту виникає через значне зростання обсягів інформації та необхідністю її структурованого опрацювання для подальшого використання. У теперішньому інформаційному середовищі текст виробляється масово – у вигляді коментарів, повідомлень, відгуків, твітів, дописів, новинних публікацій, постів тощо. Щоб системи могли ефективно відреагувати на такий масивний потік даних текстова інформація має бути класифікованою за відповідними ознаками.

Класифікація за емоційністю

Аналіз емоційності тексту включає у себе визначення тону отриманого повідомлення, а саме чи є воно позитивним, негативним або нейтральним. Також, системи з глибоким аналізом мають змогу виділяти рівень певного тону. Такий підхід до розпізнавання надає дозвіл на оцінку ставлення автора до певної теми або об’єкта. Аналіз з визначенням емоційності широкого використання набуває у сферах де потрібна оцінка відгуків, настроїв користувачів, а також визначення емоційного тону у зверненні клієнтів до служби підтримки. Саме це дозволяє автоматизувати процеси та оперативно реагувати на критичні ситуації.

Класифікація за тематикою

Тематичний аналіз тексту означає встановлення його змістовної належності до певної галузи знань, або сфери діяльності. У багатьох випадках текст містить ключові слова та фрази, які впливають на визначення основної тематики.

Автоматичні системи класифікації тем використовують спеціально розроблені алгоритми для виявлення повторюваних шаблонів. Проте, в складніших випадках потрібне використання більш сучасних та глибинних моделей машинного навчання, які здатні аналізувати контекст та семантику. Практичного використання тематична класифікація набула у сортуванні новин, системах рекомендацій, підборі контенту за інтересам тощо. Це значно спрощує процес структурування контенту та підвищує релевантність видачі його.

Класифікація за токсичністю

Токсичність в тексті – це прояв агресії у мовній площині, заклик до насильства тощо. Такий контент буває спрямований на одну особу, або на групу людей за певними ознаками. Тобто виявлення токсичності в тексті це надзвичайно важлива частина в цифровій безпеці. Автоматизація розпізнавання токсичного контенту дає можливість зменшити ризик шкодити від агресивної комунікації в цифровому середовищі. Така класифікація отримала своє визнання та активно застосовується в сфері модерації та захисті користувачів від булінгу, а саме автоматична фільтрація коментарів, блокування користувачів за порушення норм, обмеження показу реклами.

Отже, емоційність, тематика та рівень токсичності – це три ключові характеристики, за якими найчастіше здійснюється класифікація текстової інформації у сучасних системах. Їхнє автоматичне виявлення дозволяє збільшити ефективність аналізу та покращити реакцію на вміст повідомлень, також вдосконалити адаптацію контенту під потреби користувача, що забезпечує захищене інформаційне середовище.

1.3 Формулювання цілей та завдань вебсервісу

У межах реалізації інтелектуальної системи аналізу настроїв тексту за допомогою штучного інтелекту обрано формат вебсервісу, оскільки це забезпечує

зручний доступ з будь-якого пристрою з доступом до інтернету. Він не потребує встановлення додаткового програмного забезпечення та має легку інтеграцію в інші системи. Саме такий підхід дозволяє організувати ефективну взаємодію з кінцевим користувачем.

В межах цієї роботи планується:

- проаналізувати сучасні рішення та підходи до автоматичного аналізу текстових даних;
- здійснити порівняння моделей та обґрунтувати вибір найдоцільнішого варіанта для розв’язання поставленої задачі;
- спроєктувати архітектуру програмного забезпечення та реалізувати його серверну частину;
- розробити функціонал класифікації текстів за визначеними ознаками;
- створити клієнтську частину системи з інтуїтивно зрозумілим інтерфейсом;
- реалізувати механізми збереження результатів роботи та доступу користувачів;
- забезпечити повноцінне тестування функціоналу системи та перевірку її працездатності.

Метою проєкту є розробка вебсервісу з головною функцією якою є автоматизований аналіз текстової інформації за допомогою штучного інтелекту за трьома критеріями: емоційністю, тематикою та рівнем токсичності, а також додатковий функціонал.

Основні завдання вебсервісу:

- класифікація тексту за емоційним забарвленням (тональністю): позитивна, негативна або нейтральна;
- визначення тематичної належності тексту з-поміж кількох основних категорій;

- виявлення токсичності у контексті, тобто образливих, агресивних або соціально неприйнятних висловлювань;
- генерація резюме (узагальнення змісту та витяг основної думки) в одному-двох реченнях, особливо актуально для довгих текстів;
- переформулювання текстів на основі отриманого рівня токсичності (можливість зменшення або підвищення);
- підтримка мультимовності шляхом використання моделей, які були навчені на багатомовних корпусах тексту;
- реалізація двох режимів використання: базового (без авторизації) і повного (після реєстрації або входу в систему);
- можливість аналізу файлів (.txt, .json, .csv, .pdf, .docx, .mp3, .wav, .m4a).
- збереження історії аналізів в базу даних з можливістю фільтрації та перегляду результатів, а також видалення результатів.
- функціонал для управління акаунтом зі сторони користувача

Таким чином, вебсервіс забезпечує гнучкий підхід до обробки природної мови та демонструє практичне застосування сучасних моделей трансформерної архітектури для вирішення реальних задач.

2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ТЕХНОЛОГІЙ

2.1 Технології класифікації тексту

Класифікацією тексту прийнято називати процес асоціювання вхідного тексту з певними категоріями, які можуть бути визначені заздалегідь або в процесі. Це інструмент для миттєвої та якісної обробки інформації, що у зв’язку з постійним зростанням обсягів текстових даних є необхідним. Методи класифікації тексту пройшли еволюцію від простих словникових правил до сучасних моделей глибокого навчання, що здатні до врахування контексту та значення слів у тексті, залежно від поставленої задачі та використання технологій.

Загальні підходи до класифікації

Правила/словники (rule-based methods) – метод, у якому є заздалегідь прописаний людиною набір правил для аналізу. Наприклад ключові слова, такі як “поганий”, “брудний”, “дурний”.

Статистичні методи – машина перевіряє, які слова з якою частотою використання зустрічаються в текстах певних категорій та на основі цієї інформації навчається розпізнаванню нових текстів на основі статистичних закономірностей. Прикладом таких алгоритмів є наївний байєс (найлегший і найпоширеніший), SVM (метод опорних векторів).

Методи на основі векторних подань – текст мусить бути перетвореним у набір чисел, які є векторами та дозволяють алгоритму знаходити схожість між словами та текстами.

Моделі глибокого навчання – нейронні мережі, котрі мають можливість обробити текст враховуючи семантичні моменти, тобто використання його в контексті. Наприклад LSTM – послідовні нейромережі, трансформери (BERT, GPT тощо) – найефективніші моделі.

Сучасні технологічні рішення значно спрощують роботу з класифікацією тексту. Актуальне широке використання спеціальних програмних засобів та фреймворків, які забезпечують реалізацію вище згаданих підходів. Як для простих статичних моделей, так і більш складних моделей глибокого навчання. Завдяки інтеграції таких засобів у процес розробки ефективність виконання задач значно збільшується. Прикладами слугують системи автоматичної модерації, пошуку, чат-боти тощо. Вибір моделі для системи аналізу є ключовим етапом, оскільки саме ефективність класифікації залежить від неї.

2.2 Огляд і порівняння мовних моделей

В межах розробки інтелектуальної системи для комплексного аналізу настроїв тексту, включаючи емоційність, тематику, токсичність та інтеграцію додаткового функціоналу, ключовим моментом є вибір мовної моделі. Оскільки саме модель відповідає за забезпечення глибокого та ефективного розуміння контексту та тонких емоційних моментів, прихованих емоцій, потенційно шкідливого змісту та контенту. У цьому проєкті розглянуто підходи на базі відкритих моделей, а також через інтеграцію з API. Узагальнивши структури використаних в цьому проєкті моделей можна отримати:

XLM-RoBERTa – трансформерна модель, яка включає у себе використання певної множини мов, тобто багатомовності та є вдосконаленням версії RoBERTa. Саме після розширення версії ця модель набула функціоналу з використанням розширеного списку мов. Нова версія була навчена на масивних текстових даних близько ста мов. Оновлений варіант з багатомовністю дозволяє розробку всеохопних інтелектуальних систем класифікації тексту різними мовами, яка були включені у вибірку для навчання моделі.

Архітектура Transformer дала ґрунт для XLM-RoBERTa. Точність в даній моделі, яка отримана за допомогою використання методу маскування слів у тексті (masked language modeling), тобто урахування контексту по обидва боки від слова, є

основним пунктом для вибору цієї моделі для аналізу настроїв. Саме завдяки аналізу контексту в ліву та праву сторону від основного слова можна отримати коректний результат. Крім того, для підвищення точності результату при аналізі настроїв можна використовувати спеціалізовані версії цієї моделі для конкретних задач – таких як аналіз токсичності, тематика тощо. Попри це, для досягнення максимальної точності можна адаптувати модель до конкретної задачі – провести донавчання.

GPT (моделі випущені OpenAI) – Generative Pre-trained Transformer, тобто модель трансформерна, проте автогенеративного типу, що дозволяє передбачати наступне слова після основного з врахуванням використовуваного контексту. Модель з автоагрегативною структурою уваги (causal/self-attention), саме ця структура може ефективно опрацювати великі об’єми даних. Для користування цією моделлю потрібно оплатити тариф користування (рис. 2.1).

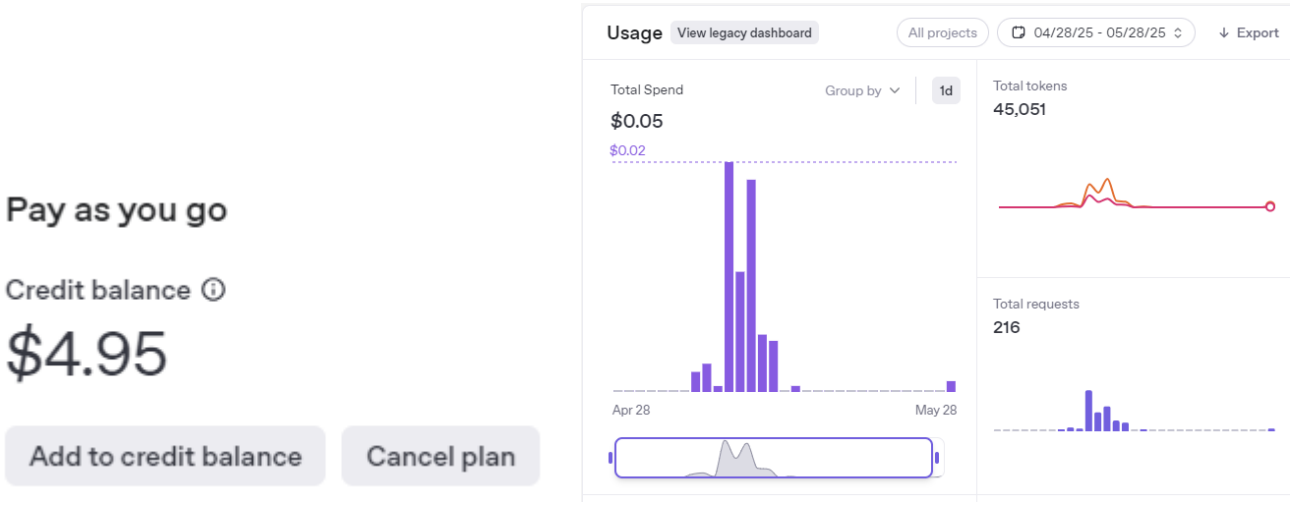


Рисунок 2.1 – Сторінка тарифу для користування GPT API

Проте, основна модель в проєкті є спеціалізовані версії XLM-RoBERTa, а GPT модель як інтелектуальний інструмент для розширення функціоналу. Модель працює через API OpenAI, це означає що:

- вона не запускається локально і усі обчислення відбуваються на стороні OpenAI;
- доступ до неї вимагає з’єднання з інтернетом і API-ключа з оплаченим тарифом користування;

2.3 Порівняльний аналіз рішень та вибір підходу

В ході розробки інтелектуального вебсервісу для аналізу текстової інформації постала задача обрати найоптимальніший підхід для реалізації функціоналу аналізу настроїв тексту. Найголовнішим при цьому було отримати точність, ефективність, швидкодію та багатомовність. За результатами пошуків, порівняльного аналізу та тестуванням моделей було обрано інтегрувати в систему комбінацію підходів:

1. Використання готових навчених моделей, створених на базі архітектури XLM-RoBERTa та адаптованих під конкретні задачі інтелектуальної системи. Для кожної з трьох класифікаційних задач – застосування окремої попередньо натренованої моделі, що спеціалізується сама на потрібних типах даних, наприклад, твіти, коментарі, пости в соціальних мережах, чати тощо.
2. Інтеграція сучасної мовної моделі через API, зокрема GPT від OpenAI, яка дозволяє обробляти запити генеративного характеру, зокрема в даному проєкті переформулювання тексту, генерацію резюме або визначення основного змісту. Використовуючи інтеграцію через API локальне розгортання моделі не потрібне, саме такий підхід дозволяє не утворювати додаткові обчислювальні навантаження на сервер. GPT модель навчено на масивних корпусах текстової інформації, що включають вебсторінки, книги, наукові статті, діалоги, інструкції та програмний код.

Пріоритетні критерії для порівняння та вибору підходів та моделей є: висока точність класифікації, ефективність роботи, багатомовність, прозорість моделі, ресурсовикористання.

3 СЕРЕДОВИЩЕ ТА ЗАСОБИ РОЗРОБКИ

3.1 Мова програмування Python

Python сучасна мова програмування високого рівня, яка вперше представлена у 1991 році. Основною ідеєю її створення стало забезпечення максимальної зрозумілості та чистоти у написанні коду. Саме тому Python вирізняється простим, лаконічним синтаксисом, що значно полегшує та оптимізує процес розробки.

Python має величезну спільноту користувачів та активно використовується та розвивається. Сьогодні набув широкого застосування в таких сферах, як аналітика даних, штучний інтелект, автоматизація процесів, веброзробка, кібербезпека тощо. Завдяки потужним бібліотекам і фреймворкам, а також сучасним інструментам для роботи з нейронними мережами, наприклад, Transformers, Python став універсальним інструментом для реалізації будь-яких ідей.

Для реалізації даного програмного продукту було прийнято рішення використовувати саме мову Python, оскільки вона ідеально підходить для створення гнучких, функціональних і масштабованих рішень. Її популярність у сучасній розробці пояснюється не лише зручністю, а й широким спектром інструментів, які доступні для різних галузей, зокрема веброзробки, штучного інтелекту, обробки даних, машинного навчання та взаємодії з API.

Python також надає потужні можливості для інтеграції з технологіями, які використовуються у цьому проєкті. Зокрема, мова забезпечує повну сумісність з фреймворком Flask, платформою Firebase, бібліотекою Transformers та API від OpenAI. Це дозволяє без додатково задіяного функціоналу поєднувати всі компоненти системи в єдину архітектуру.

Крім того, Python має зручні засоби для роботи з HTTP-запитами, авторизацією користувачів, обробкою JSON-даних і керуванням середовищем

через .env-файли, що робить його підходящим для реалізації як серверної логіки, так і інтеграції зі сторонніми сервісами.

У результаті, мову Python було обрано як основну мову програмування проєкту через її ефективну сумісність з тими інструментами, які використовуються для розробки. Вона найкраще підходить для інтеграції з бібліотеками штучного інтелекту, роботи з API моделей, а також для взаємодії з обраними фреймворками та хмарними сервісами. Усі обробки запитів користувача, взаємодія з моделями штучного інтелекту, а також авторизація та збереження даних здійснюються саме за допомогою Python-коду.

Структура проєкту була організована у вигляді модульного застосунку, де кожен компонент виконує чітко визначену функцію. Наприклад, окремі модулі відповідають за маршрутизацію запитів у Flask, інші – за обробку відповідей від моделей. Такий підхід підвищує зручність розробки та ефективність тестування окремих частин системи.

Python був використаний і для роботи з конфігураційними файлами .env, що містять ключі API та інші чутливі дані. Це дозволило гнучко налаштовувати середовище виконання та забезпечити безпеку при роботі з зовнішніми сервісами.

3.2 Фреймворк Flask

Фреймворк Flask для створення вебсистем мовою Python, надає розробнику базову структуру для будування серверної частини застосунку. У реалізації системи Flask використовується для організації обробки HTTP-запитів, створення маршрутів та передачі даних між клієнтом і сервером. Саме через цей фреймворк здійснюється взаємодія з зовнішніми сервісами, такими як API моделі GPT від OpenAI та трансформерів від Hugging Face, зокрема XLM-RoBERTa, а також з платформою Firebase.

Застосування Flask дозволяє легко реалізувати REST API, які є основним механізмом обміну даними між фронтендом, написаним на JavaScript та бекендом. Для зберігання конфіденційних налаштувань, таких як API-ключі, використовується бібліотека python-dotenv, яка інтегрується з Flask і дає змогу керувати середовищем за допомогою .env файлів.

3.3 Інтеграція моделей Hugging Face Transformers та API GPT

У межах цього проєкту реалізовано інтеграцію двох типів мовних моделей: моделей GPT від OpenAI через офіційний API та попередньо навчених трансформерів з платформи Hugging Face, які завантажуються локально для виконання окремих завдань.

Основна логіка генерації тексту реалізується через API OpenAI, зокрема з використанням моделі GPT-3.5-turbo. Запити до моделі надсилаються у форматі JSON через HTTP-з'єднання з авторизацією за допомогою ключа, що зберігається в .env файлі. Такий підхід дозволяє отримувати швидкі та якісні текстові відповіді, генеровані нейромережею, відповідно до заданих параметрів (системні повідомлення, промпти, контекст).

Для автоматизованого аналізу вхідного тексту також були використані кілька відкритих моделей трансформерів з Hugging Face типу XLM-RoBERTa. Вони були завантажені для локального використання через бібліотеку transformers, що дозволило підвищити швидкість обробки без звернення до зовнішнього API. Зокрема:

1. textdetox/xlmr-large-toxicity-classifier-v2 – для визначення рівня токсичності повідомлень;
2. cardiffnlp/tweet-topic-21-multi – для тематичної класифікації вмісту на основі багатомовних твітів;

3. cardiffnlp/twitter-xlm-roberta-base-sentiment – для аналізу тональності тексту, зокрема виявлення позитивних, негативних або нейтральних емоцій.

Завдяки поєднанню моделей Hugging Face та GPT API вдалося побудувати систему, яка спочатку проводить попередню обробку, оцінює настрій текст за кількома критеріями (емоційність, тема, токсичність), а потім звертається до генеративної моделі GPT для змістовного резюме, за потреби перегенерації тексту зі зміною рівня токсичності.

3.4 Використання Firebase для автентифікації та бази даних

В інтелектуальну систему було інтегровано хмарну платформу Firebase як зручне рішення для організації автентифікації користувачів та зберігання даних у реальному часі. Firebase надає набір вже готових сервісів, які легко інтегруються у веб-додатки та забезпечують надійність і масштабованість без необхідності самостійного налаштування серверної інфраструктури.

Для авторизації користувачів застосовується Firebase Authentication (рис. 3.1). Цей сервіс дозволяє налаштувати вхід через email і пароль, а також, за потреби, через сторонні сервіси (рис. 3.2), такі як Google, Facebook чи GitHub.

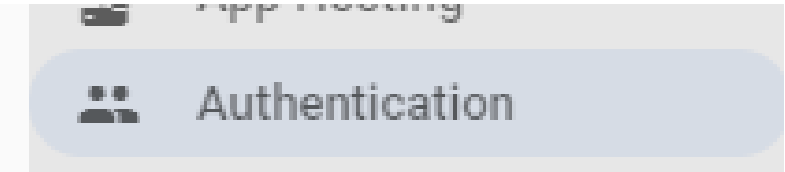


Рисунок 3.1 – Вкладка авторизації в Firebase Authentication

У даному проєкті використано стандартну email/пароль авторизацію.

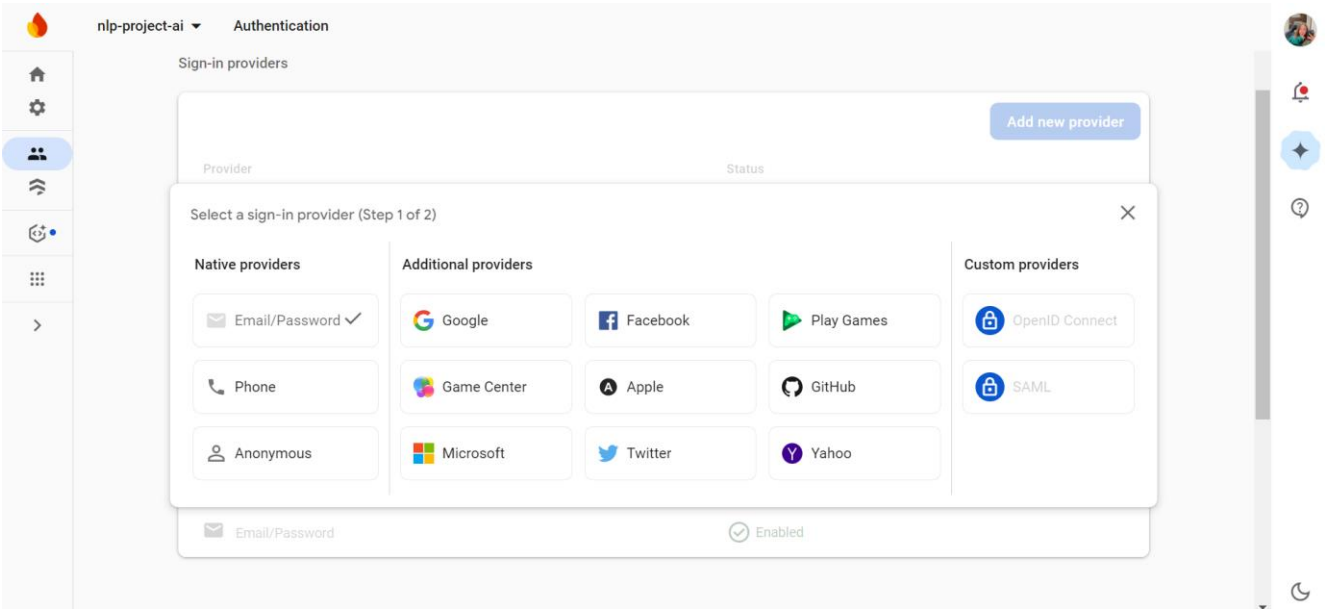


Рисунок 3.2 – Варіанти авторизації в Firebase Authentication

Реєстрація, вхід і вихід користувача реалізовані через вбудовані функції Firebase SDK, які взаємодіють із клієнтською частиною застосунку.

Інтерфейс Firebase забезпечує простий контроль за користувачами, дає змогу переглядати список зареєстрованих акаунтів та керувати ними у кілька кліків. Для захисту даних використовуються токени доступу, які автоматично оновлюються, забезпечуючи безпеку на рівні сесії.

Рисунок 3.3 – Вкладка хмарного сховища в Firebase Authentication

У якості бази даних (рис. 3.3) використовується Firebase Realtime Database – хмарна база даних, що дозволяє зберігати та миттєво оновлювати дані у форматі JSON.

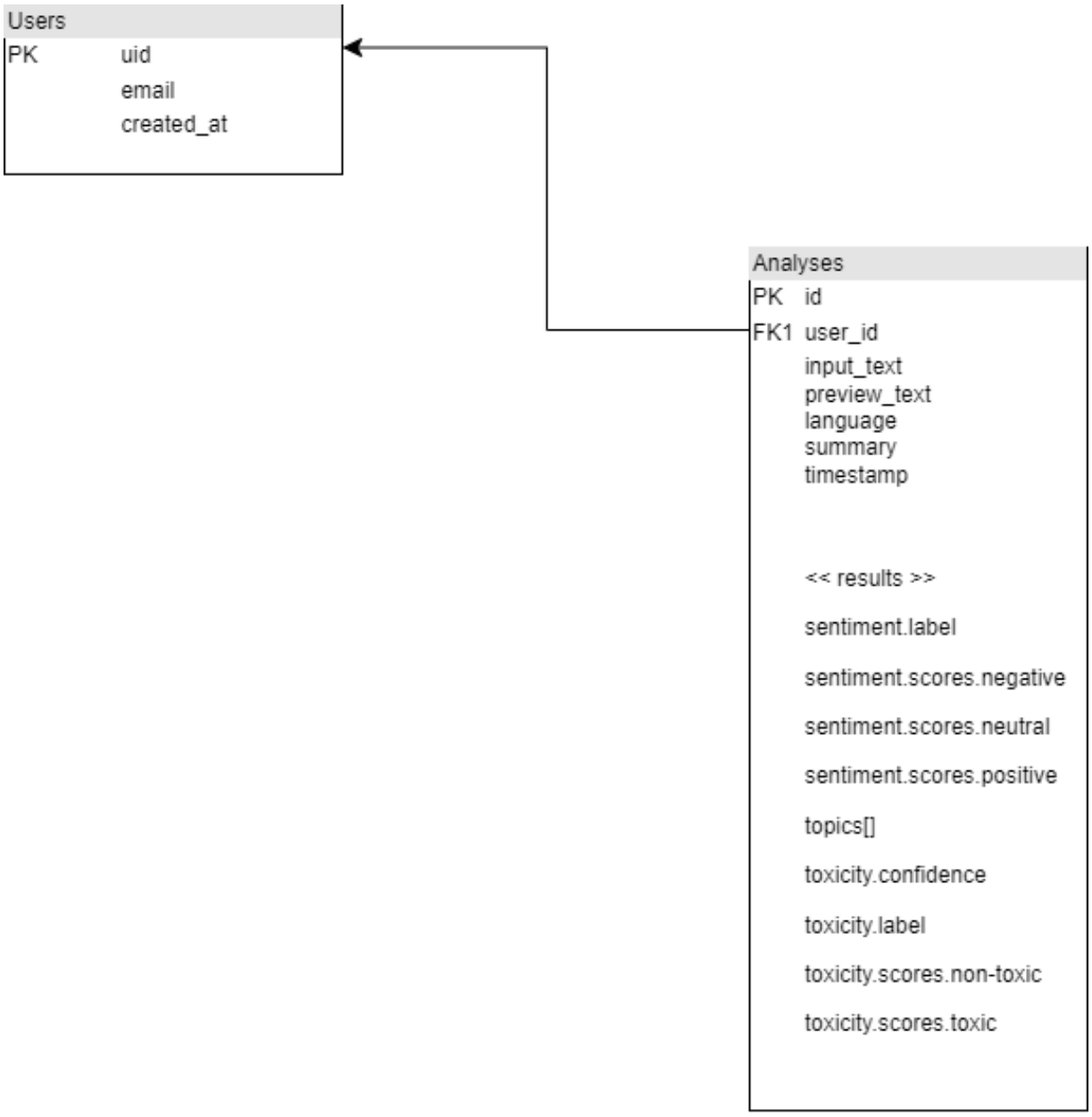


Рисунок 3.4 – Діаграма структури бази даних

Це особливо корисно для зберігання запитів користувачів, відповідей моделей, а також додаткової інформації, яка може бути потрібна для статистики або навчання системи в майбутньому. Структура бази даних зображена на діаграмі на рисунку 3.4.

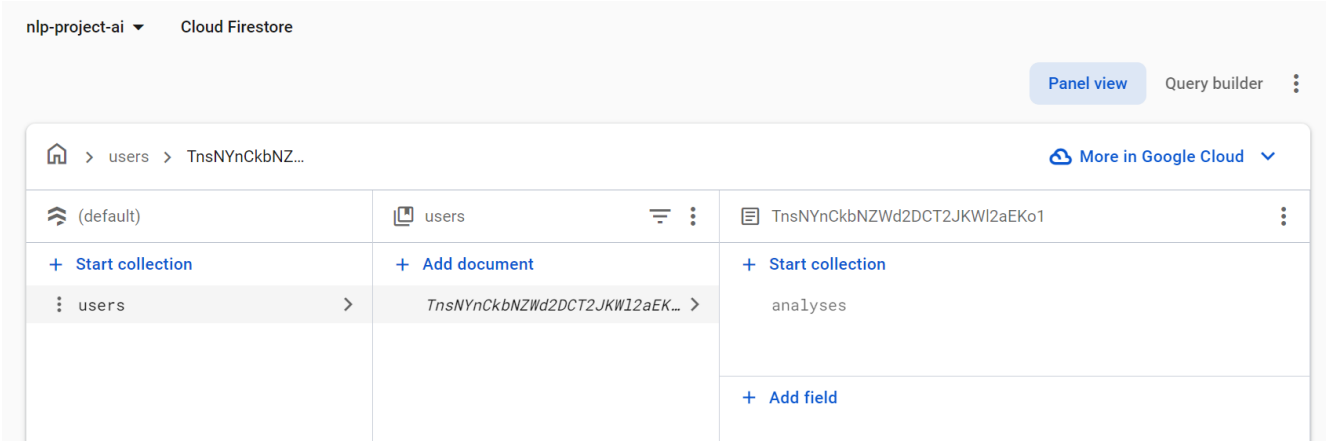


Рисунок 3.5 – Firebase Realtime Database зберігання даних відповідно до користувача

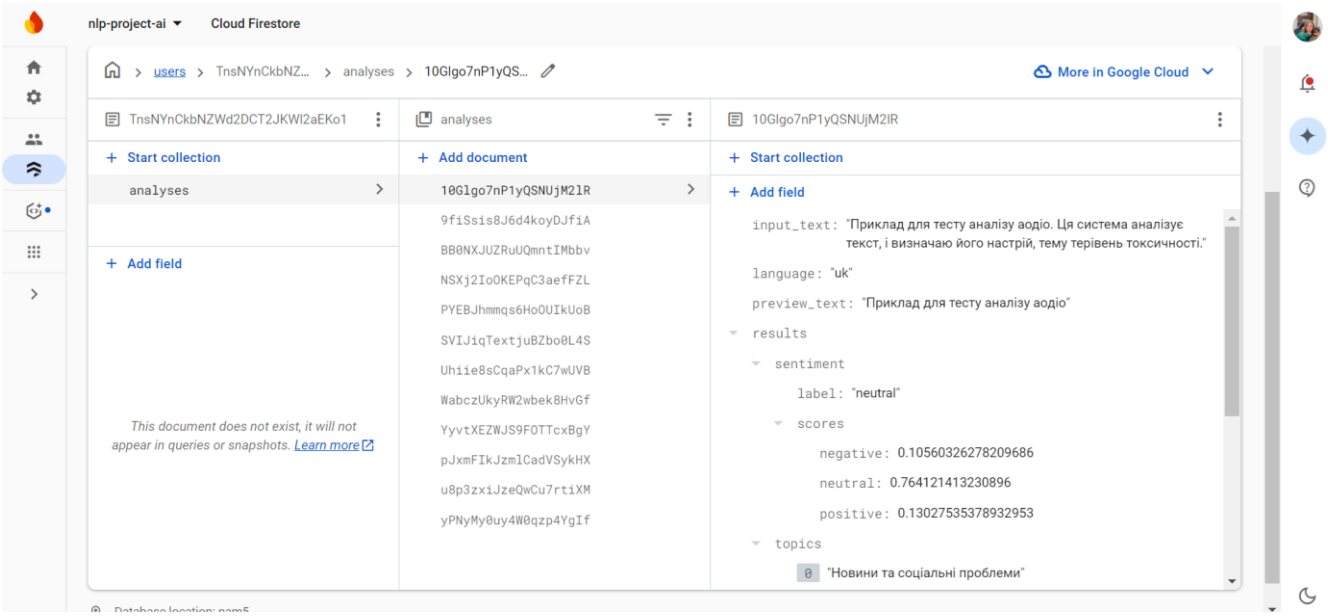


Рисунок 3.6 – Firebase Realtime Database приклад зберігання даних

Структура бази даних дозволяє зручно групувати інформацію за UID користувача (рис. 3.5, 3.6), що спрощує фільтрацію та обробку даних на сервері. Доступ до записів обмежений згідно з правилами безпеки Firebase Security Rules, які визначають, хто і які дані може читати або змінювати.

3.5 Розробка фронтенду: HTML, CSS, JavaScript

Фронтенд застосунку створений з використанням основних вебтехнологій – HTML, CSS та JavaScript, які забезпечують, інтуїтивно зрозумілий та функціональний інтерфейс. Застосунок працює динамічно без перезавантаження сторінки та активно взаємодіє з серверною частиною через API.

HTML використовується для побудови логічної структури сторінки: форма введення, кнопки, діаграми, блок результатів і повідомлення. CSS відповідає за стилізацію інтерфейсу – кольорову гаму, шрифти, відступи, індикацію рівня токсичності тощо.

JavaScript реалізує всю логіку взаємодії користувача з вебсервісом:

- відправлення тексту на аналіз через API,
- обробка відповіді з моделі,
- відображення результатів на сторінці,
- оновлення інтерфейсу на основі даних.

Також реалізована перевірка авторизації користувача через Firebase Auth і захищений доступ до серверної частини за допомогою токена.

Для наочності результатів аналізу застосовано популярні бібліотеки:

Chart.js – побудова "doughnut"-діаграми для візуалізації емоційного стану (позитивний, нейтральний, негативний);

ApexCharts – створення радіальної діаграми токсичності, яка змінює колір відповідно до рівня;

Колірна індикація підтримується через JavaScript, що допомагає швидко зчитувати результат навіть візуально.

Обробка файлів і генерація PDF

Інтерфейс підтримує аналіз аудіо та текстових файлів. Користувач може завантажити документ, який автоматично розпізнається та надсилається на сервер для аналізу.

Також реалізовано функцію експорту результатів у PDF за допомогою бібліотеки html2pdf.js. Це дозволяє зберігати результати аналізу локально, що зручно для подальшого використання.

Основні бібліотеки, що використовуються на фронтенді:

- Chart.js – побудова кругової діаграми;
- ApexCharts – радіальна шкала для токсичності;
- html2pdf.js – експорт сторінки у PDF;
- Firebase JS SDK – авторизація, токени, доступ до бази;
- fetch API – надсилання запитів до бекенду.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Архітектура застосунку

Архітектура застосунку побудована за принципом клієнт-серверної моделі, що зображено на діаграмі на рисунку 4.1 та 4.2, для відображення процесів обміну даними між компонентами системи.

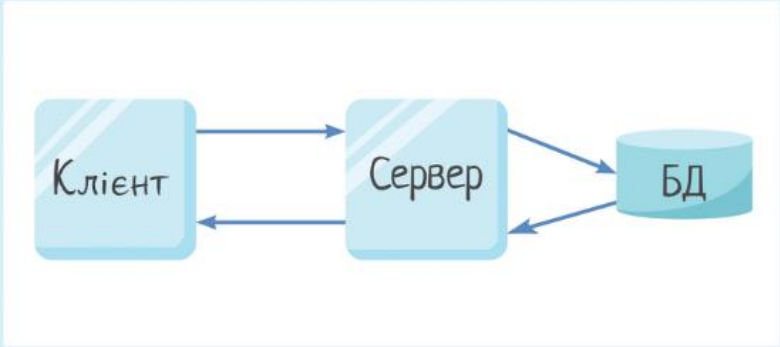


Рисунок 4.1 – Принцип клієнт-серверної архітектури

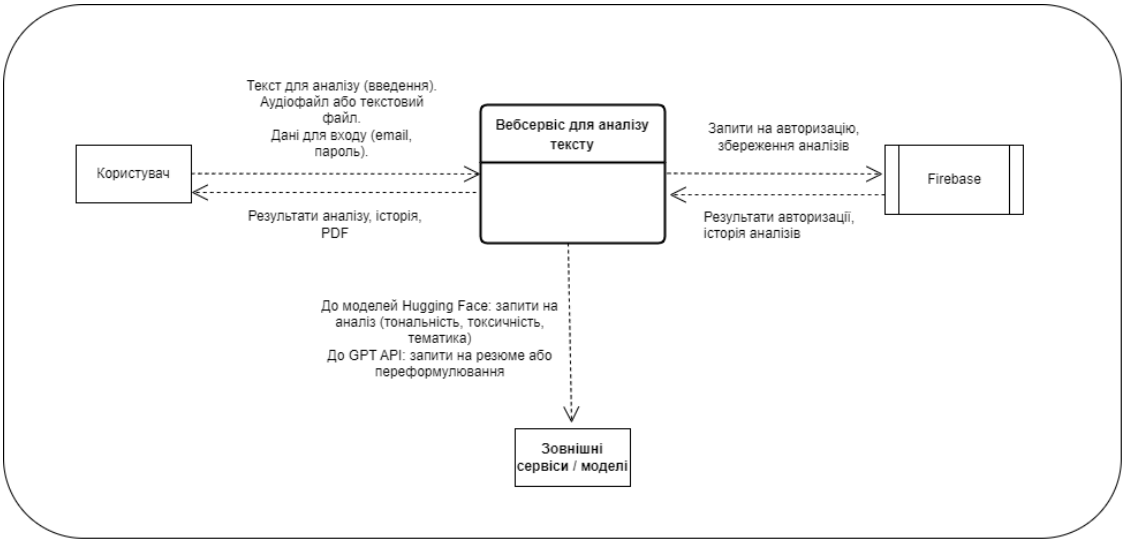


Рисунок 4.2 – DFD-рівень 0, потік даних у вебсистемі

В вебсервісі фронтенд відповідає за взаємодію з користувачем, а бекенд обробляє запити, виконує аналіз тексту, звертається до мовних моделей та керує базою

даних. Для зв’язку між компонентами використовується HTTP-протокол із передачою даних у форматі JSON.

Система складається з наступних основних блоків:

1. Фронтенд (HTML, CSS, JavaScript)

   Відповідає за збір тексту від користувача, візуалізацію результатів, авторизацію через Firebase та зручний інтерфейс. Реалізована підтримка роботи з файлами, генерація PDF та побудова діаграм для результатів аналізу.

2. Бекенд (Flask, Python)

   Обробляє всі запити, надсилає текст до моделей (локально або через API), виконує попередню обробку, chunking (розбиття на частини), агрегацію результатів, створення резюме, збереження інформації до Firebase. Сервер захищений авторизацією через токен Firebase.

3. Мовні моделі

   - Локальні: моделі з Hugging Face (textdetox/xlmr, cardiffnlp/tweet-topic, twitter-sentiment) – використовуються для попереднього аналізу.
   - Віддалені: GPT API (OpenAI) – використовується для генерації відповіді або переформулювання тексту.

4. База даних (Firebase Realtime Database)

Зберігає історію аналізів, попередні запити, ключові метадані. До бази даних можна звертатись лише після авторизації.

5. Сторонні бібліотеки

Chart.js, ApexCharts, html2pdf.js – відповідають за побудову графіків, збереження результатів у PDF, інтерактивність інтерфейсу.

4.2 Реалізація бекенду

Бекенд реалізовано з використанням мови програмування Python та фреймворку Flask. Основне завдання серверної частини – обробляти запити, виконувати попередній аналіз тексту, взаємодіяти з мовними моделями та забезпечувати зв’язок із базою даних.

Основні функції бекенду:

1. Обробка HTTP-запитів

   Через Flask реалізовано REST API з маршрутами типу POST /api/full-analysis, GET /api/analyses, POST /api/regenerate_text тощо.

   Дані передаються у форматі JSON, а відповіді формуються після обробки моделей.

2. Аутентифікація користувача

   Для захищеного доступу до бекенду використовується Firebase Authentication. Токен, отриманий на фронтенді, перевіряється сервером для ідентифікації користувача перед обробкою запиту.

3. Інтеграція з моделями

   Використано бібліотеку transformers для завантаження та використання моделей з Hugging Face.

   Додатково інтегровано GPT API від OpenAI – запити до моделі надсилаються через HTTPS з використанням API-ключа.

4. Обробка тексту

   Реалізовано метод chunking – розбиття великих текстів на частини для стабільної обробки.

Здійснюється агрегація результатів після аналізу окремих частин (середнє значення емоційності, токсичності, список тем).

5. Генерація резюме та переформулювання

Окремі ендпоінти відповідають за створення коротких підсумків (резюме) або зміну тону тексту (на менш/більш токсичний) за допомогою генерації тексту використовуючи GPT.

6. Збереження результатів

Результати аналізу зберігаються у Firebase Realtime Database.

Дані структуруються за унікальним ID користувача для подальшого відображення історії аналізів.

Технології та бібліотеки:

- Flask – обробка запитів;
- firebase-admin – перевірка токенів та робота з базою;
- Transformers – завантаження моделей Hugging Face;
- requests – надсилання запитів до GPT API;
- dotenv, json, uuid, datetime – допоміжні бібліотеки для обробки налаштувань, даних та логіки.

4.3 Реалізація фронтенду

Фронтенд частина застосунку реалізована на основі HTML, CSS та JavaScript із використанням сторонніх бібліотек для візуалізації даних та взаємодії з сервером, що зображено на діаграмі послідовностей на рисунку 4.3.

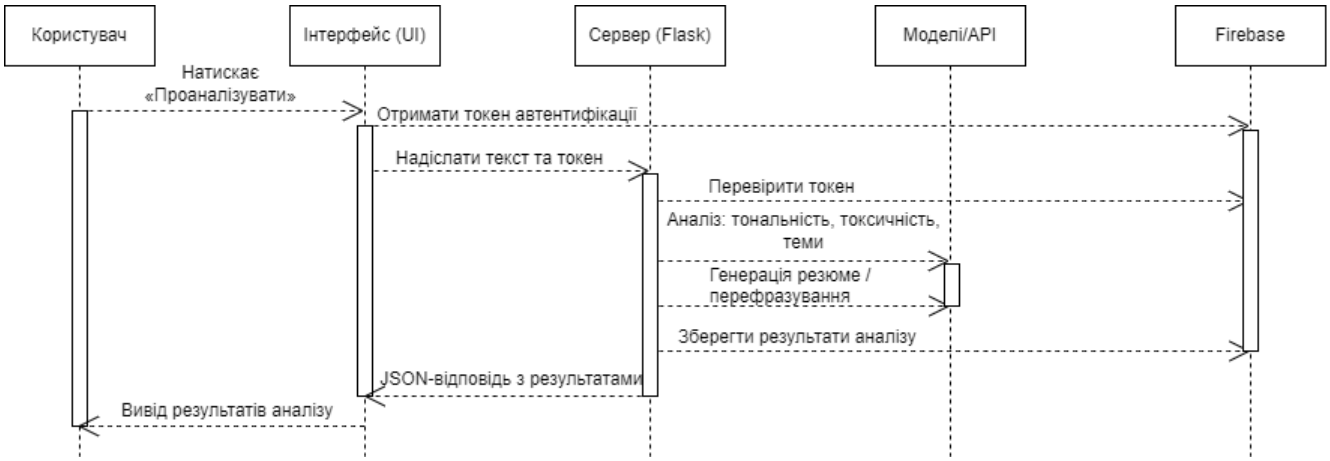


Рисунок 4.3 – Діаграма послідовностей взаємодії користувача через інтерфейс з вебсистемою під час повного аналізу тексту

Основна мета – надати користувачеві зручний інтерфейс для аналізу тексту, перегляду результатів і роботи з файлами.

Інтерфейс інтелектуального вебсервісу містить такі ключові елементи:

- поле для введення тексту або завантаження файлу;
- кнопки запуску аналізу, перегенерації та експорту в PDF;
- блок візуалізації результатів (токсичність, емоційність);
- історія попередніх запитів;
- повідомлення та підказки у випадку помилок або відсутності даних.

HTML використовується для побудови розмітки сторінки, а CSS відповідає за стилізацію та адаптивність. Завдяки правильній структурі елементів забезпечено сумісність з мобільними пристроями та різними браузерами.

JavaScript реалізує всі основні функції:

- отримання введеного тексту;
- перевірка авторизації користувача через Firebase;

- надсилання запиту на аналіз (/api/full-analysis);
- обробка та відображення результатів;
- генерація діаграм (тональність, токсичність);
- динамічне створення елементів DOM (наприклад, бейджі тем).

Асинхронні запити реалізовано через fetch, із використанням авторизаційного токена для доступу до захищених маршрутів. У випадку з помилками передбачено виведення інформативних повідомлень для користувача.

Для графічного представлення результатів використовуються:

- Chart.js – побудова кругової діаграми для тональності (позитивна, нейтральна, негативна);
- ApexCharts – створення радіальної діаграми для рівня токсичності з динамічною зміною кольору;
- бейджі тем – побудовані вручну через JavaScript.

Ці інструменти дозволяють подати результати аналізу у візуальній формі.

Додаткові функції такі як:

Генерація PDF – реалізована через бібліотеку html2pdf.js, яка дозволяє користувачу зберегти результати аналізу у вигляді документа.

Аналіз файлів – підтримується завантаження аудіо або текстових файлів, оброблений текст автоматично вставляється у поле для аналізу.

Перегенерація – можливість змінити стиль тексту (зробити його більш або менш токсичним) через окрему кнопку.

4.4 Інтеграція моделей та API

Інтелектуальні можливості системи реалізовані через інтеграцію локальних моделей Hugging Face на базі XLM-RoBERTa, а також генеративного GPT API від OpenAI. Далі описано технічні особливості підключення.

Локальні моделі від Hugging Face (Transformers):

Щоб забезпечити автономність, стабільність підключення та уникнути залежності від інтернету, усі необхідні моделі з платформи Hugging Face були заздалегідь завантажені та збережені локально у директорії проєкту. Це дозволяє запускати основний функціонал аналізу тексту без потреби у зовнішніх підключеннях під час продакшн-деплою.

Таке рішення дозволяє:

- уникнути затримок при завантаженні моделей в реальному часі;
- мінімізувати ймовірність помилок типу ConnectionError, Timeout, RateLimitExceeded;
- запускати систему в офлайн-середовищі, включно з локальним сервером чи Docker-контейнером;

З технічного боку моделі завантажувалися за допомогою бібліотеки transformers та зберігались у форматі, який дозволяє подальше локальне ініціалізування:

У проєкті локально використовується три основні моделі:

1. textdetox/xlmr-large-toxicity-classifier-v2 – визначення токсичності;
2. cardiffnlp/twitter-xlm-roberta-base-sentiment – аналіз емоційності;
3. cardiffnlp/tweet-topic-21-multi – класифікація за темами.

Ці моделі запускаються миттєво при старті сервера, без онлайн-доступу, що значно спрощує деплой на будь-який сервер, у тому числі із закритим доступом до зовнішніх ресурсів.

Інтеграція GPT API від OpenAI

На відміну від локальних моделей Hugging Face, для генеративних завдань – таких як створення резюме тексту або переформулювання з урахуванням токсичності – використовується віддалений API від OpenAI, зокрема модель gpt-3.5-turbo.

Використання GPT API обґрунтоване тим, що подібні великі мовні моделі (Large Language Models, LLMs) є надзвичайно ресурсомісткими та потребують потужних серверних ресурів (GPU-обладнання або доступу до спеціалізованих inference-серверів), тому в межах цього проєкту обрано хмарний варіант через API, що значно спрощує реалізацію без втрати якості.

Як реалізовано технічно:

1. Отримання API-ключа

Ключ генерується через особистий акаунт OpenAI (рис.4.4) й зберігається у .env файлі (рис. 4.5):

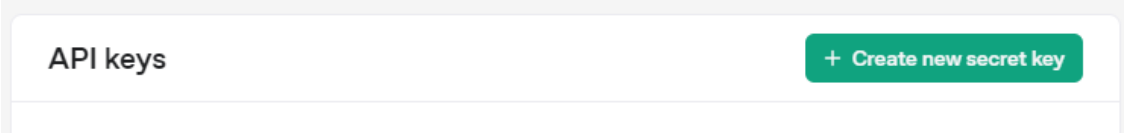


Рисунок 4.4 – Генерація API-ключа

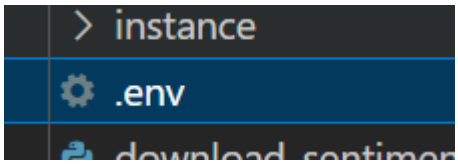


Рис 4.5 – .env файл

2. Налаштування та відправка запиту

Для взаємодії використовується офіційна бібліотека openai.

3. Запит до моделі

Формується запит у форматі ChatCompletion, де текст подається як частина діалогу.

4. Обробка та передача відповіді

Отриманий текст обробляється на сервері та передається клієнту через API. Якщо запит виконується з помилкою (наприклад, при перевищенні ліміту або відсутності інтернету), реалізовано обробку винятків з відповідним повідомленням для користувача.

Чому вибрано використання API, а не локальний доступ до моделі: моделі GPT не є доступними для завантаження локально, саме без комерційної ліцензії. Розгортання великої мовної моделі (Large Language Model, LLM) на сервері вимагало б значних обчислювальних ресурсів (від 16+ GB VRAM). OpenAI API забезпечує ефективність моделей та гнучкість у використанні через API, можна легко масштабувати запити та адаптувати логіку без оновлення інфраструктури.

У підсумку, система використовує локальні моделі для швидкого й автономного аналізу, а GPT API – для більш складних генеративних завдань, де потрібен глибший контекст і варіативність формулювань. Такий гібридний підхід дозволяє досягти балансу між якістю, швидкістю, ефективністю й незалежністю від зовнішніх сервісів.

4.5 Обробка тексту: Chunking та агрегація результатів

Оскільки деякі мовні моделі мають обмеження щодо максимальної кількості токенів або символів на вхідну обробку за один запит, то виникає необхідність у попередньому розбитті великих текстів на менші фрагменти. Цей процес називається chunking, українською це називається розбиття тексту на частини, що є важливою частиною системи.

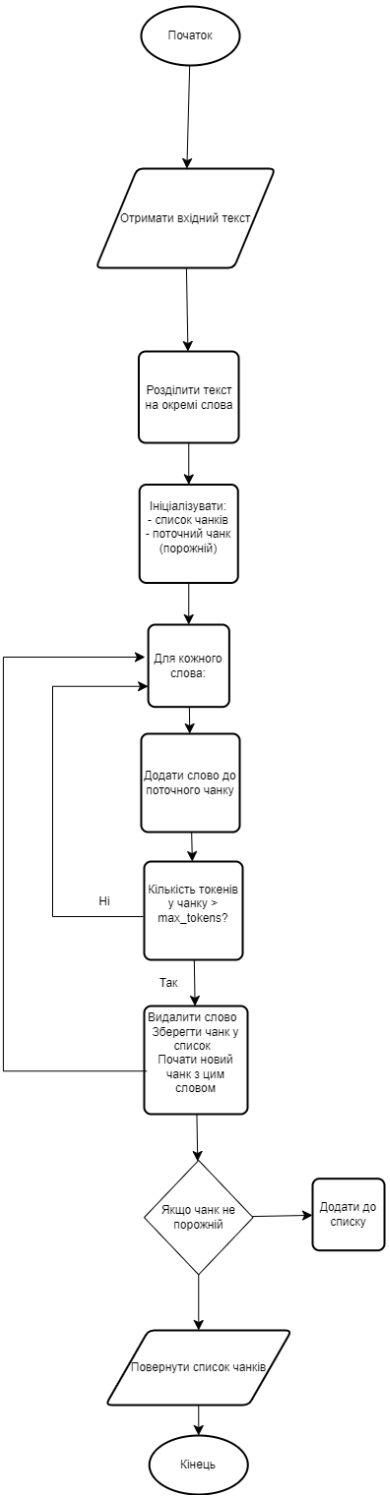


Рисунок 4.6 – Flowchart діаграма алгоритму розбиття тексту на частини

Алгоритм модуля поділу тексту на частини зображено на діаграмі flowchart рисунок 4.6. Перед початком аналізу система перш за все ділить увесь вхідний

текст на окремі частини (чанки), які мають обмежений обсяг – достатній для стабільної роботи моделі. Поділ здійснюється логічно, з урахуванням меж речень, аби зберегти змістовність кожного фрагменту. Це дозволяє аналізувати великі тексти (наприклад, документи, статті, звернення, книги, довгі пости), не порушуючи обмежень самих моделей.

Агрегація результатів

Після обробки кожної частини (чанка) окремо система проводить агрегацію – об'єднання отриманих результатів в один загальний висновок. Для кожного типу аналізу це відбувається по-різному:

- У випадку аналізу тональності (емоційності) система узагальнює результати всіх частин і визначає, яка емоція переважає серед всіх – позитивна, нейтральна чи негативна.
- Для токсичності оцінюється загальний рівень – або середнє значення, або найвищий показник, якщо хоча б один фрагмент містить неприйнятний вміст.
- У класифікації тематичності визначається, які теми найчастіше зустрічаються серед усіх фрагментів тексту та формується список основних категорій.

Цей підхід дозволяє: обробляти довгі тексти без втрати даних, уникати помилок, які пов’язані з лімітом на довжину вхідного запиту, підвищити точність аналізу, оскільки кожна частина оцінюється окремо.

Поділ тексту на частини і агрегація працюють автоматично, непомітно для користувача, та є необхідною технічною частиною інтелектуального аналізу тексту.

4.6 Генерація резюме та переформулювання тексту

У межах інтелектуальної системи реалізовано два функціонали на основі генеративної моделі GPT: створення резюме тексту та переформулювання вмісту з урахуванням бажаного рівня токсичності, який може обрати користувач. Обидва процеси виконуються за допомогою моделі GPT-3.5, доступної через API від OpenAI.

Одним із завдань аналізу є стислий виклад основного змісту тексту – тобто формування резюме. Це особливо актуально для довгих текстів, які складно швидко оцінити мануально. Система автоматично надсилає текст на обробку до генеративної моделі, яка повертає короткий, узагальнений опис змісту.

Запит до моделі формулюється з відповідною інструкцією (prompt), у якій зазначається, що потрібно створити коротке, зрозуміле резюме. Далі сформований результат відображається на фронтенді й доступний для копіювання або збереження у PDF файлі.

Другим функціоналом, побудованим на GPT, є переформулювання тексту з урахуванням рівня токсичності. Користувач може обрати один із варіантів:

- зробити текст менш токсичним – для пом’якшення різких або агресивних формулювань;
- зробити текст більш токсичним – наприклад, з експериментальною або сатиричною метою.

Модель отримує відповідну інструкцію (prompt), яка задає бажаний стиль формулювання, та генерує новий варіант на основі оригінального вмісту. Результат відображається у окремому полі на сторінці.

Реалізація обох функцій значно розширює аналітичні можливості системи, дозволяючи не лише розуміти, що сказано та з яким настроєм, а й як це можна сказати інакше або коротше.

5 ІНТЕРФЕЙС КОРИСТУВАЧА ТА РОБОТА ІЗ СИСТЕМОЮ

5.1 Базовий аналіз без авторизації

При переході на головну сторінку користувачу відкривається сторінка базового аналізу (рис. 5.1) для неавторизованих користувачів. Доступне поле для введення тексту, кнопка для аналізу, кнопка реєстрації, ознайомча інструкція, що доступно у повному режимі.

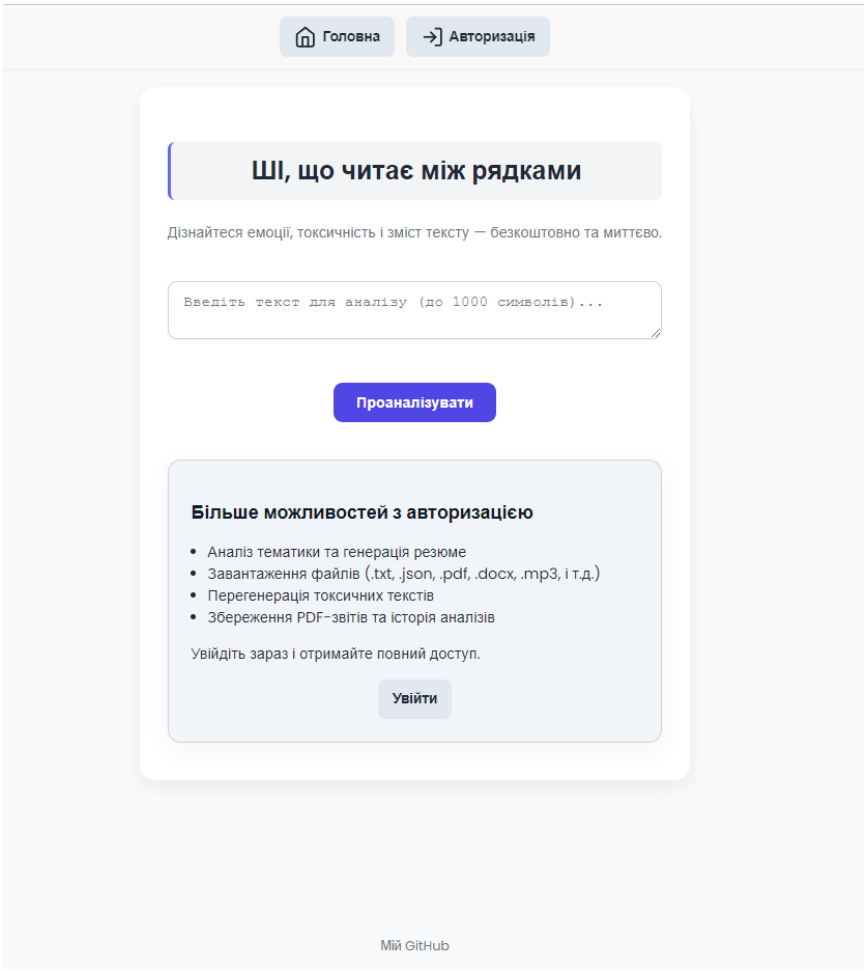


Рисунок 5.1 – Базова сторінка аналізу

Даний режим доступний без входу в обліковий запис і дозволяє швидко перевірити базові характеристики повідомлення.

Після натискання кнопки "Проаналізувати" відбувається:

- надсилання введеного тексту до API маршруту /api/analyze_base;
- обробка тексту двома моделями – для оцінки емоційності та токсичності;
- побудова діаграм Chart.js та ApexCharts для візуального відображення результатів;
- виведення повідомлення, якщо виявлено підвищену токсичність та кнопка авторизації в системі для повної інформації про токсичність.

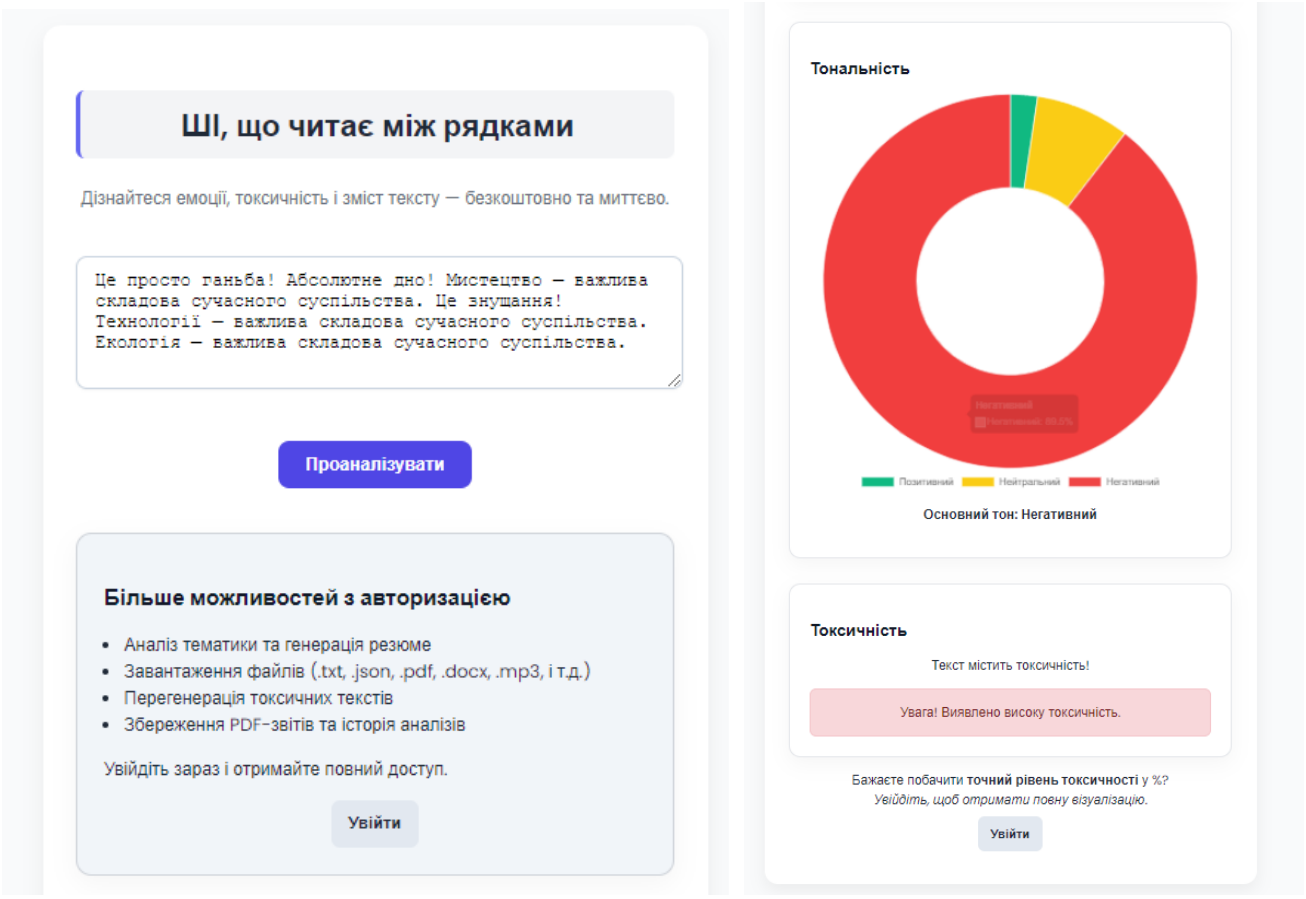


Рисунок 5.2 – Результат базового аналізу

Побудовано діаграму прецедентів (Use Case Diagram) на рисунку 5.3, яка відображає варіанти використання вебсистеми різними типами користувачів. На

схемі зображено дві ролі: гостьовий користувач та авторизований користувач, кожен із яких має доступ до відповідного набору функцій.

Гість може виконати лише базовий аналіз тексту без авторизації, тоді як повний функціонал стає доступним до використання лише після авторизації в системі.

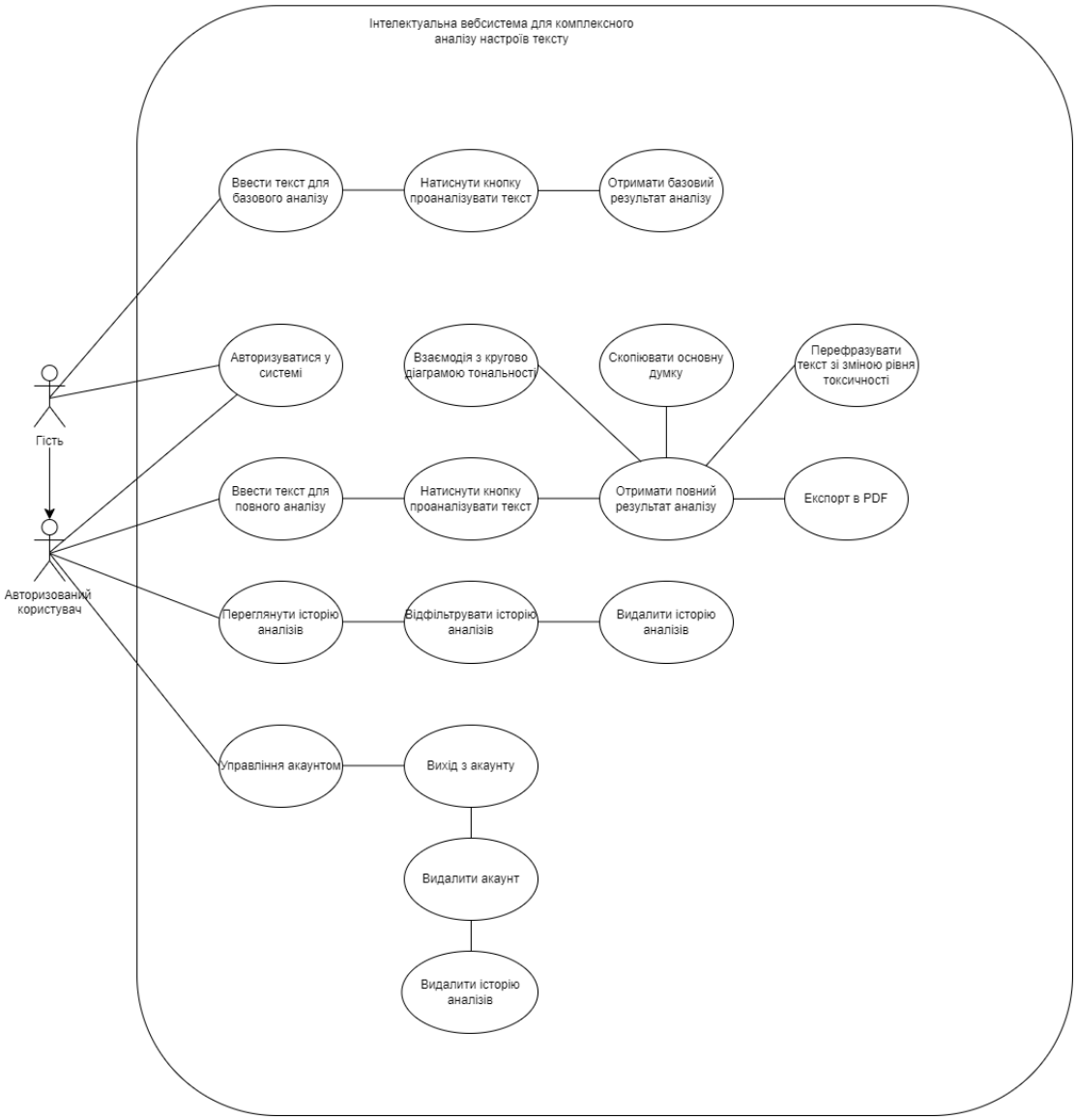


Рисунок 5.3 – Діаграма прецедентів взаємодії користувачів із вебсервісом

Функціонал, який обмежений: тематика, генерація резюме, збереження результатів, історія, експорт даних та робота з файлами недоступні без входу в систему.

5.2 Повний функціонал після авторизації

Після входу в обліковий запис через Firebase Authentication, використовуючи електронну пошту та пароль (рис. 5.4), користувачу відкривається повний функціонал інтелектуальної системи, що реалізовані на окремій сторінці.

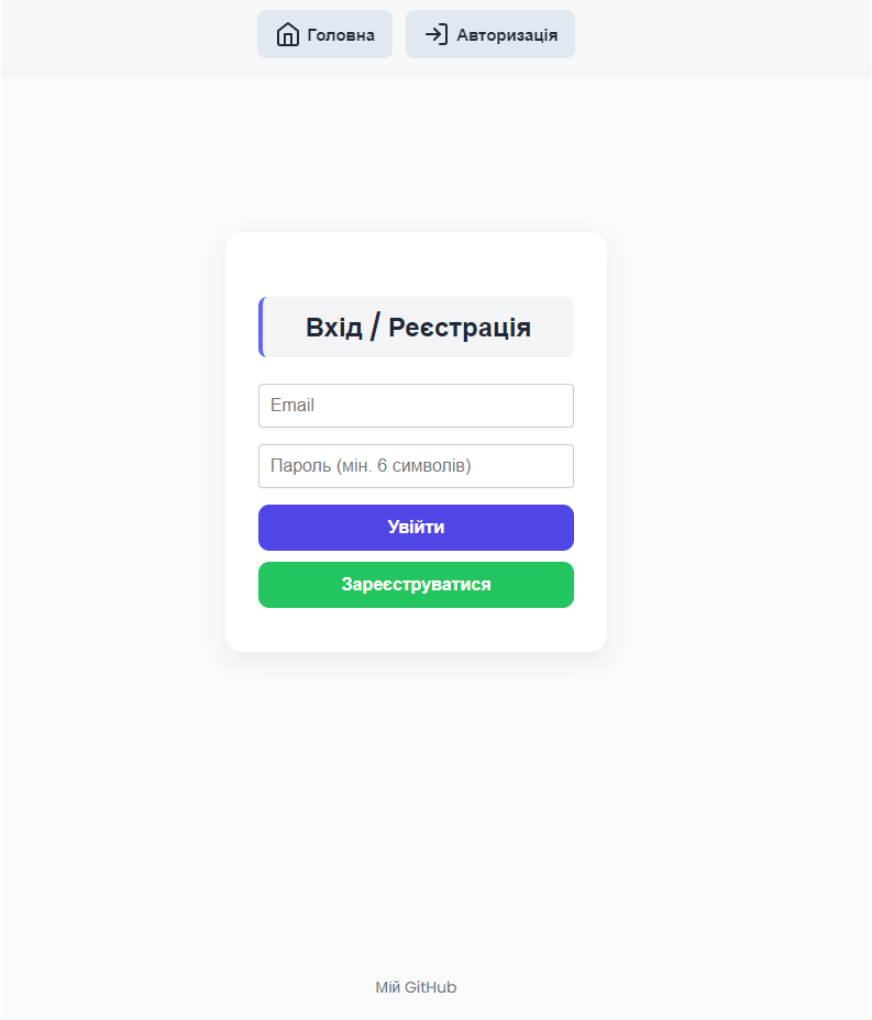


Рисунок 5.4 – Сторінка авторизації в системі

Після авторизації в системі користувач отримує доступ до сторінки повного аналізу, що зображено на рисунку 5.5.

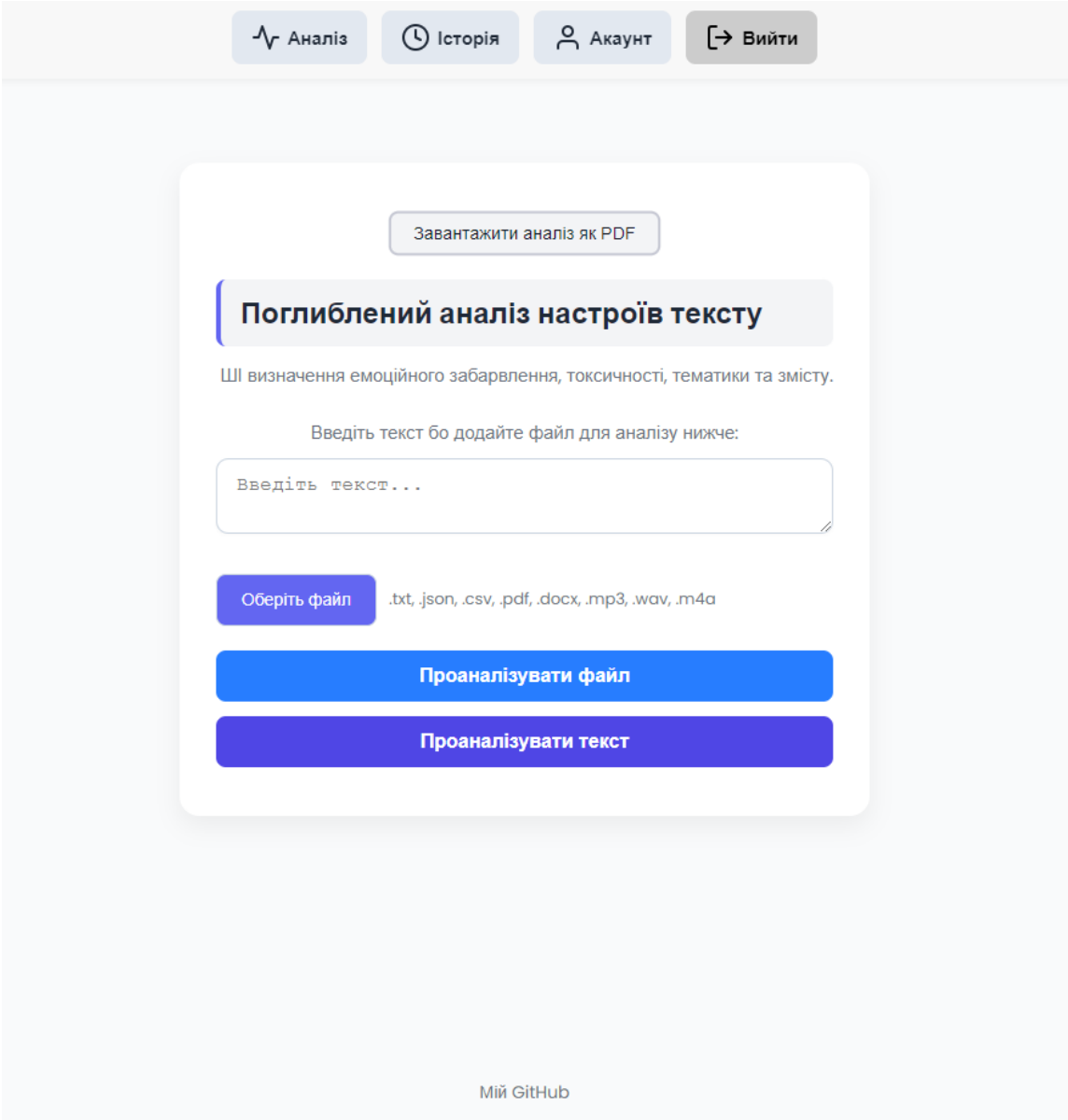


Рисунок 5.5 – Сторінка повного аналізу

Доступна така функціональність:

- Аналіз за 3 напрямами: емоційність, токсичність, тематика (визначення 2–3 тем) та взаємодія з круговою діаграмою.
- Переформулювання тексту з урахуванням бажаного рівня токсичності (менш або більш токсичний варіант).
- Інтерактивна побудова графіків і бейджів тем (рис. 5.6).

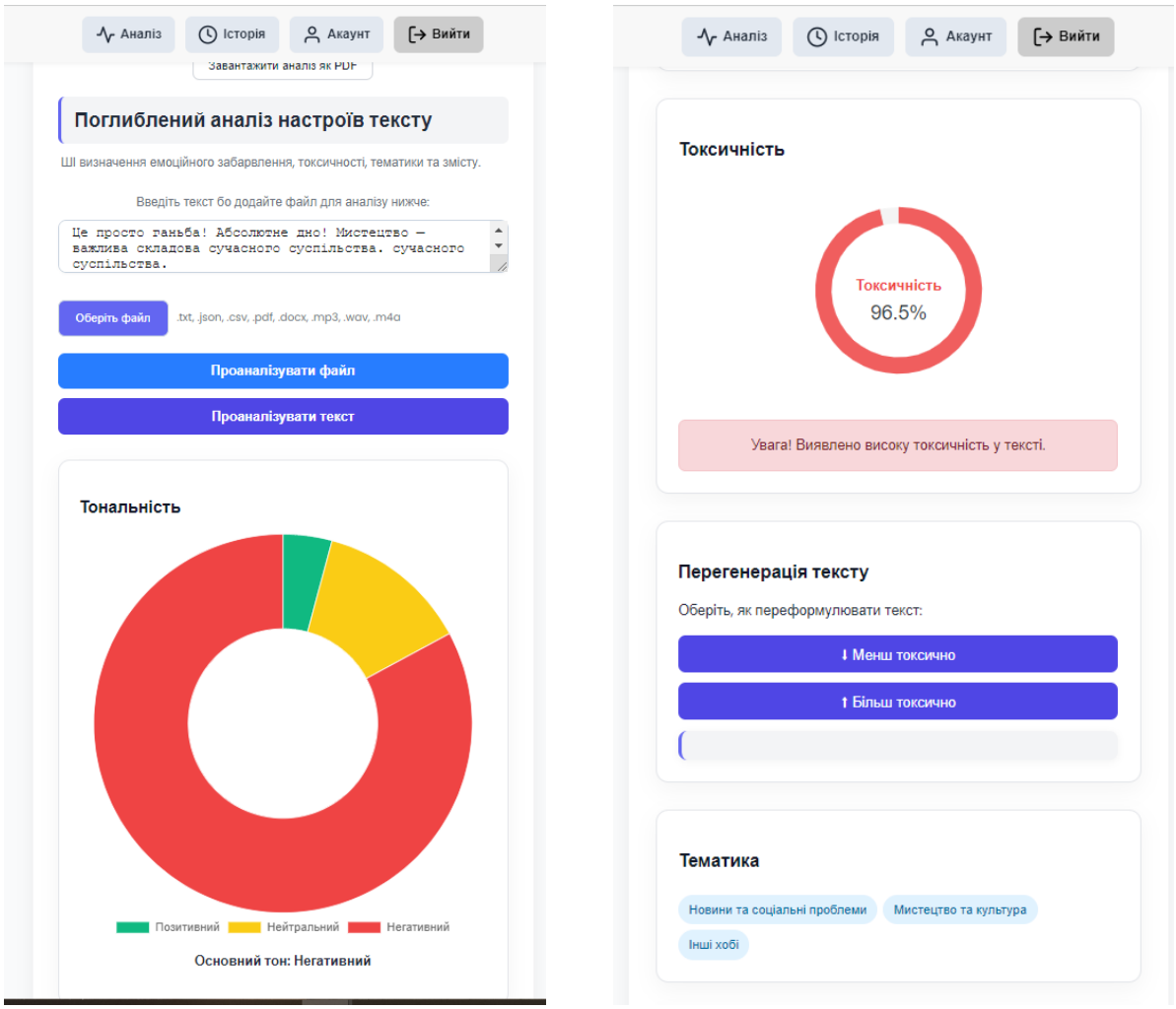


Рисунок 5.6 – Результат повного функціоналу

– Автоматичне збереження результатів у Firestore Database та взаємодія з історією на окремій сторінці (рис. 5.7), яка доступна по кліку в шапці сторінки.

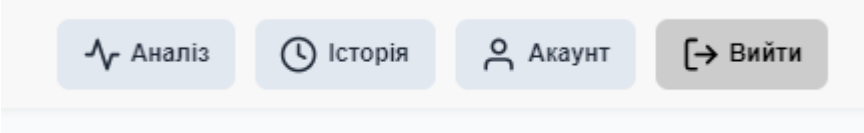


Рисунок 5.7 – Шапка сторінки

– Функціонал для вибору файлу для аналізу зображено на рисунку 5.9.

- Коротке резюме тексту (рис. 5.8) на 1-2 речення та історія трьох останніх аналізів під результатом актуального аналізу.

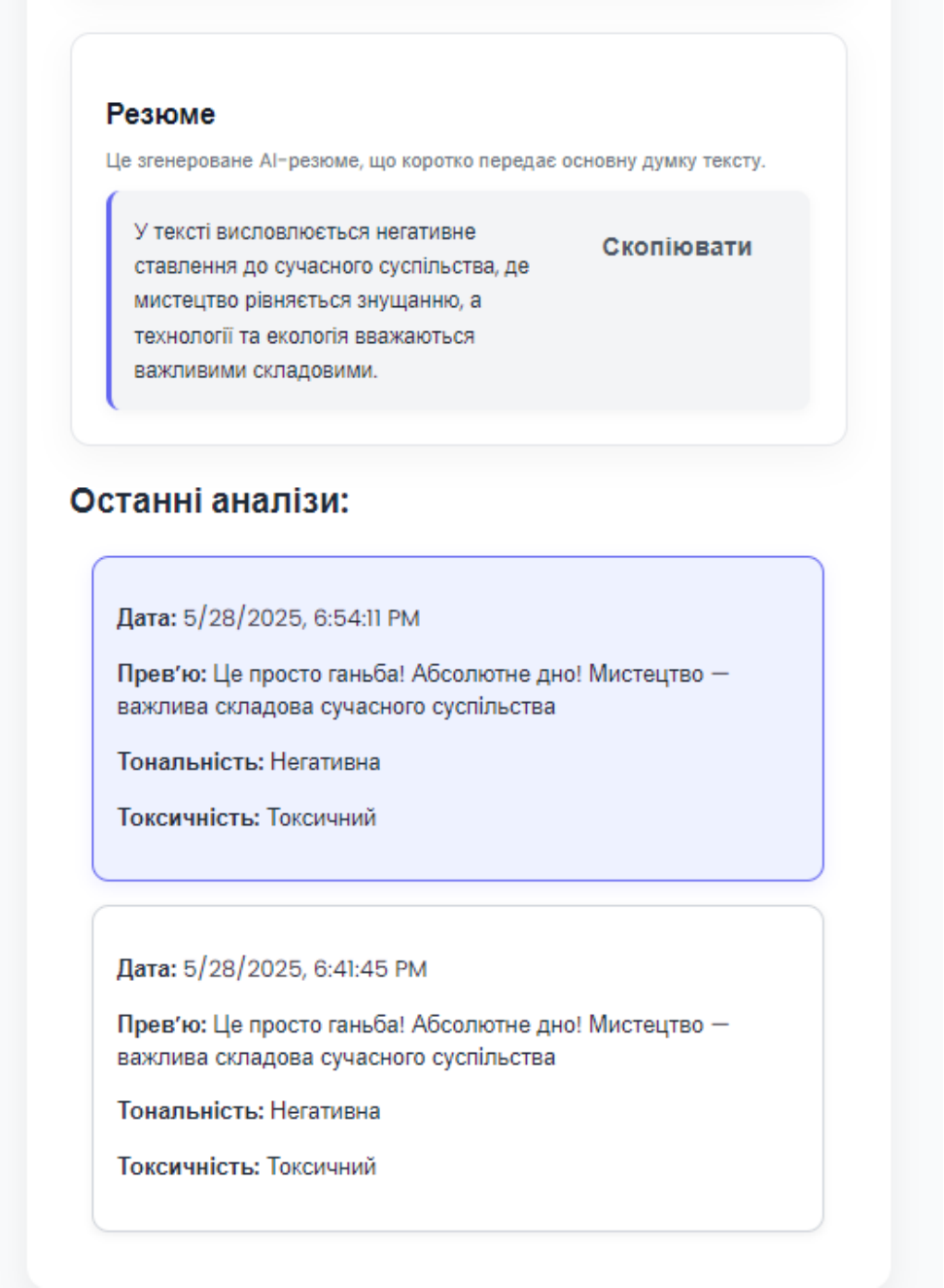


Рисунок 5.8 – Резюме та історія останніх аналізів

Також резюме можна скопіювати та використовувати як скорочений варіант основного тексту адже воно виділяє основну думку в межах 2 речень.

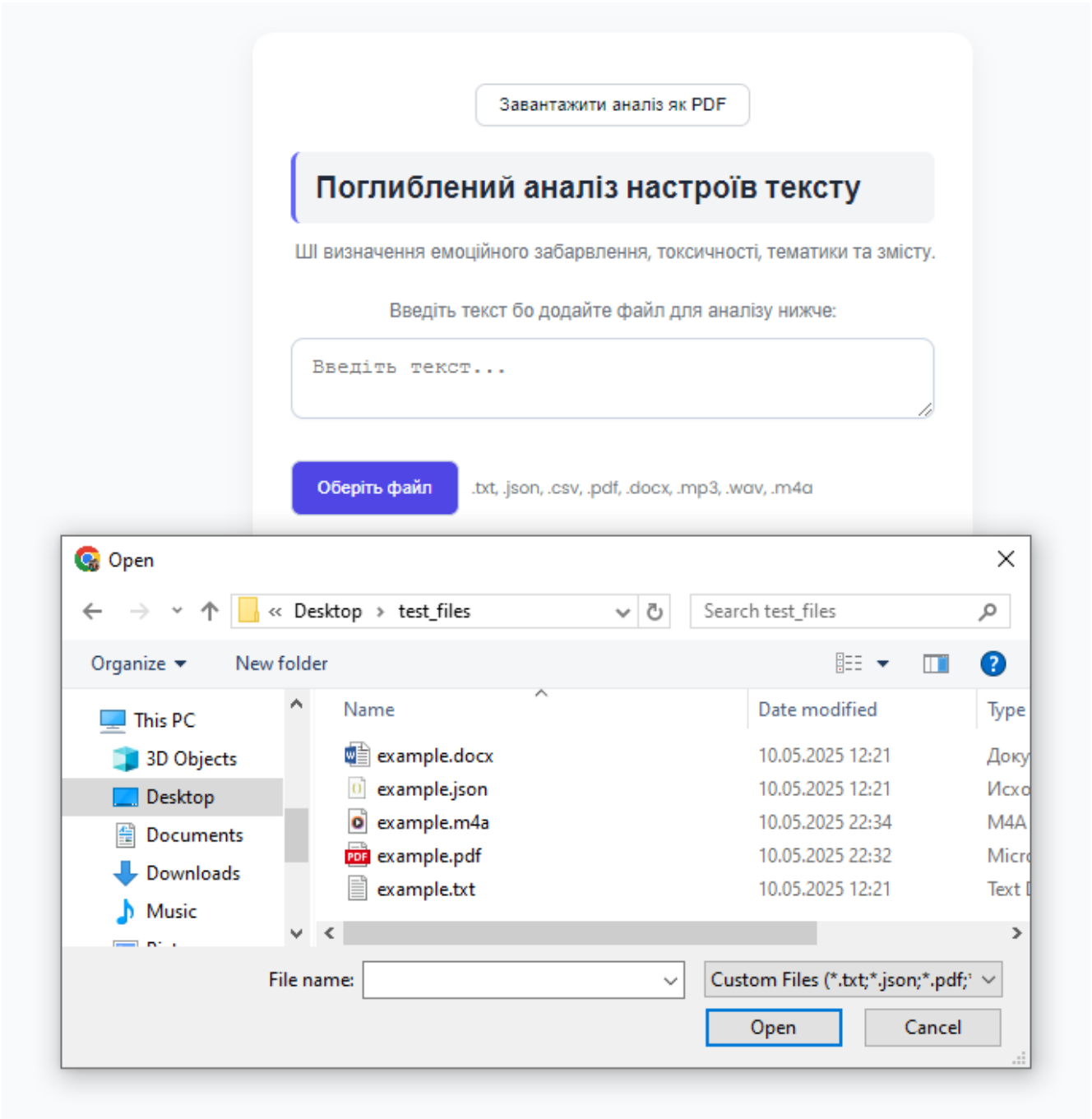


Рисунок 5.9 – Вибір файлу для аналізу

– Експорт результатів у PDF показно на рисунку 5.10 (за допомогою бібліотеки html2pdf.js).

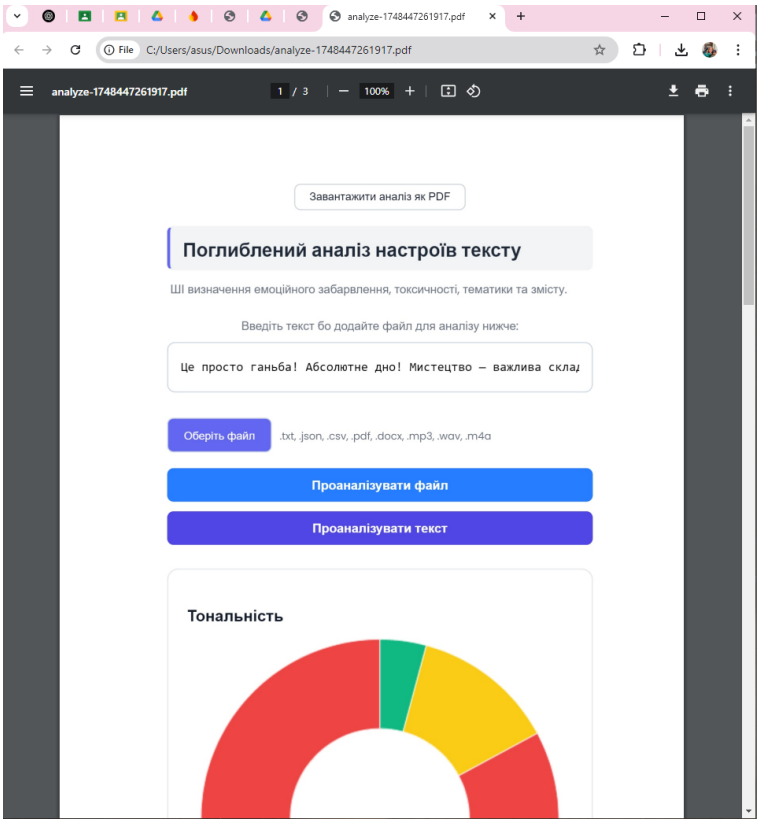


Рисунок 5.10 – Збереження аналізу в PDF файл

- Взаємодія з акаунтом користувача (рис. 5.11)

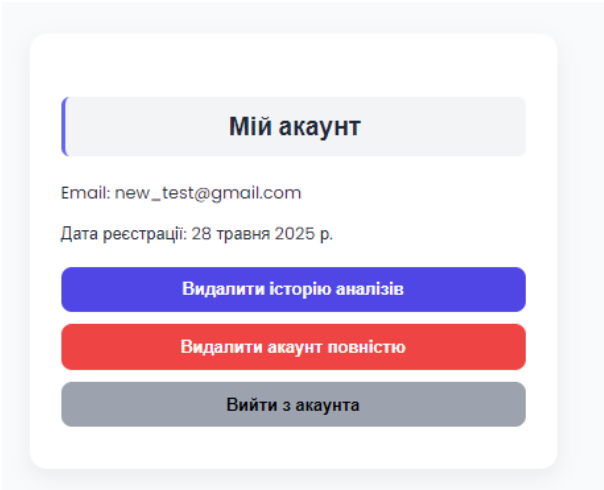


Рисунок 5.11 – Сторінка акаунту користувача

Усі запити виконуються з передачею Firebase-токена для авторизації бекенду.

5.3 Історія аналізів, перегляд та фільтрація

Після входу в систему користувачу надається доступ до повної історії текстових аналізів, які зберігаються у Firebase Realtime Database. Цей функціонал реалізований як окрема сторінка інтерфейсу (рис. 5.12), що дозволяє переглядати список та кожен аналіз детальніше у модальному вікні, фільтрувати та керувати попередніми запитами.

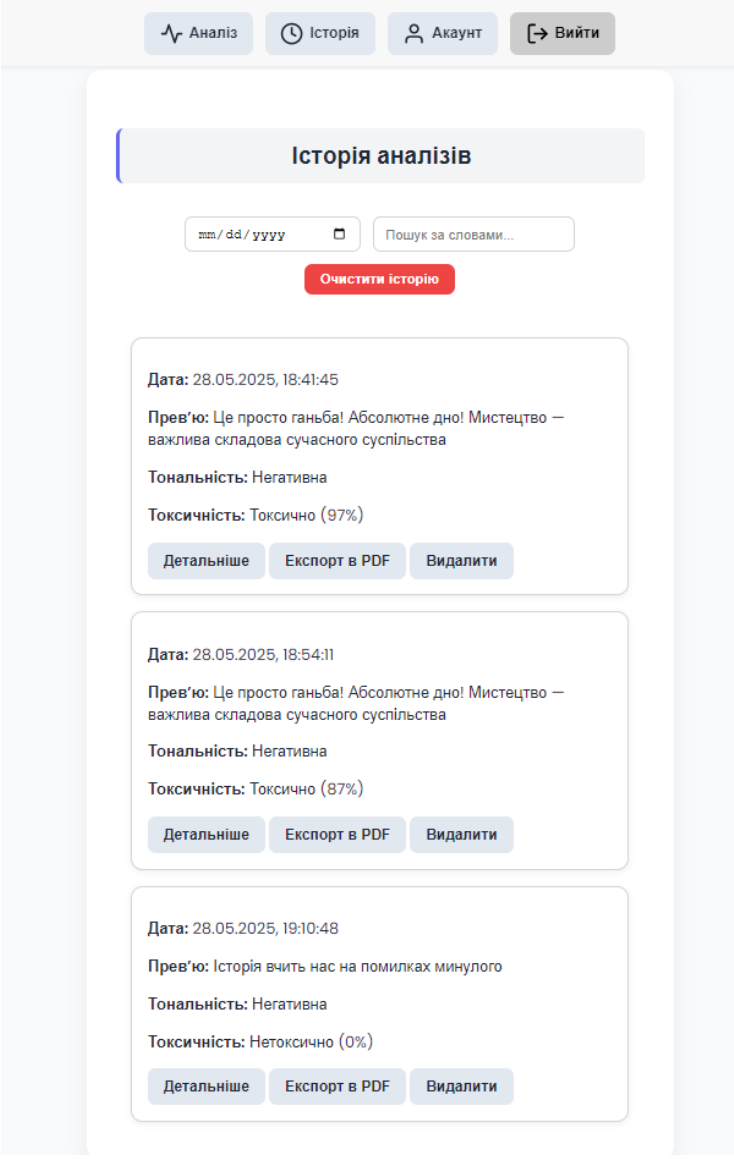


Рисунок 5.12 – Сторінка історії аналізів

Кожен результат аналізу зберігається автоматично після обробки тексту.

Дані записуються у Firebase з прив’язкою до унікального ID користувача (UID), що забезпечує доступність тільки для авторизованих осіб. Кожен запис включає:

- вхідний текст або його частину (preview – передогляд);
- результати аналізу (тональність, токсичність, теми);
- дату та час створення;
- згенероване резюме;
- унікальний ідентифікатор для можливості видалення.

Інтерфейс реалізовано у вигляді адаптивної сітки (grid view), де кожен запис відображається у вигляді картки. Ці картки містять короткий опис аналізу, зокрема:

- дату;
- попередній перегляд тексту;
- визначену тональність (із маркуванням кольором);
- рівень токсичності (із позначкою або іконкою).

За замовчуванням виводяться останні три записи одразу після нового аналізу, але також є окрема сторінка з повною історією.

Перегляд деталей в окремій сторінці та фільтрація історії, користувач має можливість:

- переглянути повні результати аналізу шляхом відкриття модального вікна натиснувши на кнопку “Детальніше” (рис. 5.13);
- для полегшення навігації реалізовано фільтрування через запити за ключовими словами, датою тощо (рис. 5.14);
- видаляти окремі аналізи або повністю очищувати історію за допомогою відповідної кнопки “Видалити”.

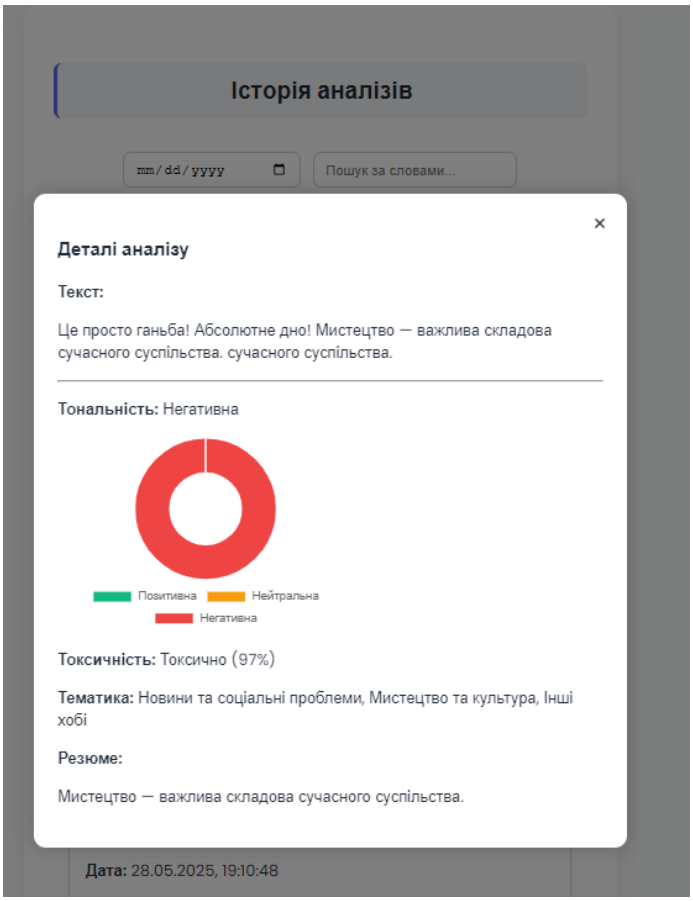


Рисунок 5.13 – Модельне вікно запису з історії

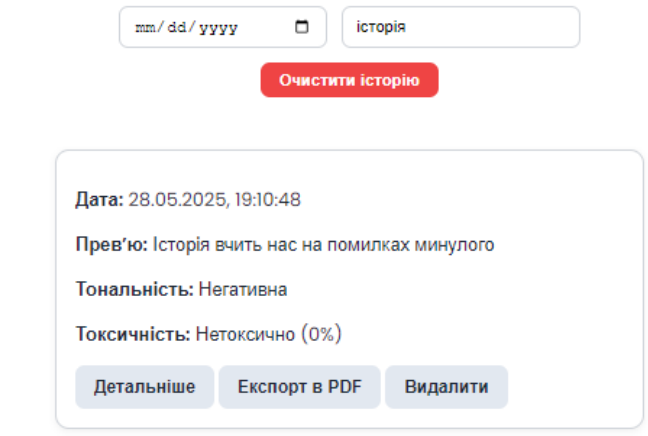


Рисунок 5.14 – Фільтрація записів

Також доступний експорт даних конкретного запису з історії в PDF файл.

Всі операції з історією прив’язані до авторизованого користувача. Сервер перевіряє Firebase-токен перед кожним запитом до /api/analyses, /api/delete_analysis, або /api/clear_history, щоб запобігти доступу до чужих даних.

5.4 Завантаження файлів та транскрипція аудіо

У системі реалізовано можливість аналізу не лише звичайного тексту, який користувач вводить вручну, а й функціонал для завантаження файлів, включаючи аудіозаписи, які автоматично розпізнаються в текстовий формат. Ця функціональність доступна тільки для авторизованих користувачів.

Формати доступні до обробки:

- Текстові файли: .txt, .doc, .csv, .json – після завантаження текст витягується, вставляється у поле аналізу та обробляється як звичайне повідомлення, якщо кількість токенів у тексті перевищує допустиму норму (480 токенів), то підключається модуль chunking та агрегація результатів після його використання.
- Аудіофайли: .mp3, .wav, .m4a – виконуються транскрипція аудіо в текст і подальший аналіз отриманого вмісту.

Інтерфейс автоматично розпізнає тип файлу. Якщо виявлено аудіо – з’являється випадаючий список для вибору мови (у тому числі опція "автоматично", для визначення мови системою), приклад зображено на рисунку 5.15.

Процес обробки файлів відбувається так:

1. Користувач обирає файл через фронтенд.
2. Файл надсилається на бекенд за маршрутом /api/analyze_file_full як FormData.
3. У випадку з аудіо:

– на сервері виконується транскрипція за допомогою моделі Whisper від OpenAI;
– текст автоматично вставляється у поле введення.

4. Після цього запускається стандартна процедура аналізу.

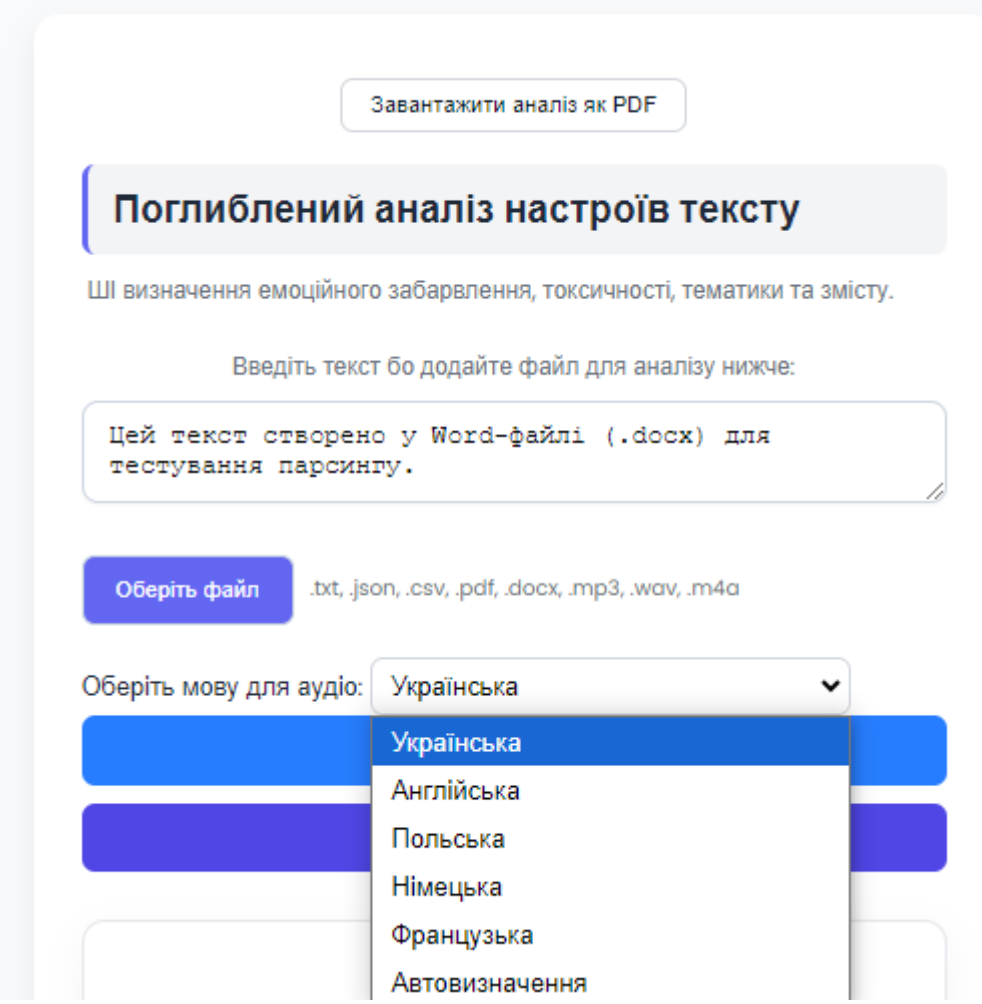


Рисунок 5.15 – Вибір мови для аналізу аудіо

Технічна реалізація: для обробки формату файлу на фронтенді використовується JavaScript API File, з перевіркою MIME-типів. Для передачі – FormData, що дозволяє прикріплювати файли разом з додатковими параметрами (наприклад, вибраною мовою). Аудіо-розпізнавання відбувається на основі інтегрованої моделі Whisper, що дозволяє зчитувати мовлення з високою точністю.

Текст, отриманий із файлу, підставляється у стандартне поле аналізу, і весь процес виглядає для користувача єдиним, без потреби додаткових дій.

Переваги використання функціоналу завантаження файлів для користувача:

- Можливість проаналізувати файл, не копіюючи вручну вміст.
- Підтримка аудіо форматів – зручно для інтерв’ю, подкастів, озвучених коментарів.

ВИСНОВКИ

В межах цього дослідження розроблено інтелектуальну вебсистему для комплексного аналізу текстів, що поєднує в собі сучасні підходи до обробки природної мови та актуальний інтерфейс взаємодії з користувачем. Актуальність реалізованого рішення обумовлена широким використанням текстової інформації в інформаційному середовищі та необхідністю її автоматизованої оцінки за критеріями тональності, токсичності, тематики та змістової насиченості.

Результати реалізації дають підстави зробити такі висновки:

По-перше, завдяки використанню трансформерних моделей, зокрема моделей архітектури XLM-RoBERTa, забезпечено точний багатомовний аналіз тексту, що дозволяє застосовувати систему для широкої вибірки користувачів.

По-друге, розроблена архітектура на базі Flask у поєднанні з Firebase та OpenAI API продемонструвала гнучкість і стабільність у роботі. Серверна частина здатна обробляти запити, взаємодіяти з моделями, виконувати генерацію текстів і зберігати історію аналізів у хмарному середовищі.

По-третє, реалізований адаптивний фронтенд забезпечує інтуїтивну взаємодію з користувачем, підтримує як базовий аналіз без авторизації, так і розширений функціонал після входу в систему. Це дозволяє зробити сервіс доступним як для швидких перевірок, так і для глибокого дослідження тексту та користування сервісом на постійній основі.

Четвертим важливим елементом є підтримка обробки файлів і транскрипції аудіо. Завдяки інтеграції з Whisper користувачі можуть отримувати текстовий аналіз навіть із голосових повідомлень чи аудіозаписів, що значно розширює практичну користь системи.

Крім того, застосовано механізм розбиття великих текстів на частини з агрегацією результатів, що забезпечує стабільність і точність навіть при роботі з довгими документами.

Таким чином, створений вебсервіс довів свою функціональність, точність та зручність, поєднавши у собі елементи штучного інтелекту та сучасної веброзробки. Він може бути корисним як дослідникам, так і звичайним користувачам для швидкої оцінки інформації в інтернеті, соціальних мережах або месенджерах.

У перспективі система може бути розширена через:

1) додавання нових мов та напрямів аналізу, таких як, виявлення маніпуляцій чи фейкових новин;
2) інтеграцію з популярними платформами Twitch, Telegram, Discord, WhatsApp;
3) створення мобільної версії або браузерного розширення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bird S., Klein E., Loper E. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. O'Reilly Media, 2009. 504 p.

2. Бішоп К. та ін. Pattern Recognition and Machine Learning: підручник. Нью-Йорк : Springer, 2006. 738 с.

3. Бьорн Х., Лю Й., Кернс Е., Крузі М. Natural Language Processing in Action: навч. посіб. Нью-Йорк : Manning Publications, 2019. 391 с.

4. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 2nd ed. O'Reilly Media, 2019. 819 p.

5. Firebase. Cloud Firestore documentation. [Електронний ресурс] – Режим доступу до ресурсу: URL: https://firebase.google.com/docs/firestore (дата звернення: 29.05.2025).

6. Firebase. Firebase Authentication. [Електронний ресурс] – Режим доступу до ресурсу: URL: https://firebase.google.com/docs/auth (дата звернення: 29.05.2025).

7. Flask. Flask Documentation. [Електронний ресурс] – Режим доступу до ресурсу: URL: https://flask.palletsprojects.com/ (дата звернення: 29.05.2025).

8. Hugging Face. Transformers documentation. [Електронний ресурс] – Режим доступу до ресурсу: URL: https://huggingface.co/docs/transformers (дата звернення: 29.05.2025).

9. Hugging Face. XLM-RoBERTa. [Електронний ресурс] – Режим доступу до ресурсу: URL: https://huggingface.co/xlm-roberta-base (дата звернення: 29.05.2025).

10. OpenAI. Text completion and prompting with GPT models. [Електронний ресурс] – Режим доступу до ресурсу: URL: https://platform.openai.com/docs/guides/gpt(дата звернення: 29.05.2025).

11. OpenAI. Whisper Speech Recognition. [Електронний ресурс] – Режим доступу до ресурсу: URL: https://platform.openai.com/docs/guides/speech-to-text (дата звернення: 29.05.2025).

12. Python Software Foundation. Python 3.11 Documentation. [Електронний ресурс] – Режим доступу до ресурсу: URL: https://docs.python.org/3.11/(дата звернення: 29.05.2025).

13. Рассел С. Дж. Штучний інтелект: сучасний підхід : навч. посіб. 4-те вид. Бостон: Пірсон, 2021. 1136 с.

14. Хеберт Б., Грін С. Applied Natural Language Processing with Python: навч. посіб. Бірмінгем : Packt Publishing, 2020. 334 с.

15. Чоллет Ф., Аллана Г., Браун Д., Воллес А. Deep Learning with Python: навч. посіб. 2-ге вид. Нью-Йорк : Manning Publications, 2021. 504 с.

ДОДАТОК А

Комплексний аналіз настроїв тексту за допомогою штучного інтелекту

Текст програми

Аркушів 16

Київ – 2025

__init__.py

```
from flask import Blueprint, jsonify, request, g
from app.services.sentiment_service import analyze_sentiment
from app.services.topic_service import classify_topic
from app.services.toxicity_service import detect_toxicity
from app.services.openai_service import generate_summary
from app.utils.auth_decorator import auth_required
from app.services.firebase_service import get_analyses
from app.services.firebase_service import delete_user_analyses
from app.services.firebase_service import delete_user_analysis_by_id
import os
from openai import OpenAI
import fitz
import json
import tempfile
import os
import docx
import whisper
import pandas as pd
bp = Blueprint('api', __name__)
@bp.route('/ping')
def ping():
    return jsonify({"status": "ok"})


@bp.route('/auth/protected', methods=['GET'])
@auth_required
def protected_route():
    return jsonify({
        "message": "Access granted",
        "user_id": g.user.get("uid"),
```

```
        "email": g.user.get("email")
    })

@bp.route('/analyses', methods=['GET'])
@auth_required
def get_analyses_route():
    user_id = g.user.get("uid")
    try:
        analyses = get_analyses(user_id)
        return jsonify(analyses), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@bp.route('/delete_history', methods=['DELETE'])
@auth_required
def delete_history():
    user_id = g.user.get("uid")
    try:
        delete_user_analyses(user_id)
        return jsonify({"message": "History deleted successfully"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@bp.route('/analyze_base', methods=['POST'])
def analyze_base():
    data = request.get_json()
    text = data.get("text", "").strip()
    if not text:
        return jsonify({"error": "Text is required"}), 400
    try:
```

```
        sentiment_result = analyze_sentiment(text)
        toxicity_result = detect_toxicity(text)
        return jsonify({
            "results": {
                "sentiment": sentiment_result,
                "toxicity": toxicity_result
            }
        }), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@bp.route('/delete_analysis/<analysis_id>', methods=['DELETE'])
@auth_required
def delete_single_analysis(analysis_id):
    user_id = g.user.get("uid")
    try:
        success = delete_user_analysis_by_id(user_id, analysis_id)
        if success:
            return jsonify({"message": "Аналіз видалено"}), 200
        else:
            return jsonify({"error": "Аналіз не знайдено або не належить користувачу"}), 404
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@bp.route('/regenerate_text', methods=['POST'])
def regenerate_text():
    data = request.get_json()
    prompt = data.get("prompt", "")
    if not prompt.strip():
        return jsonify({"error": "Prompt is empty"}), 400
```

```
    try:
        client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))
        response = client.chat.completions.create(
            model="gpt-3.5-turbo",
            messages=[{"role": "user", "content": prompt}],
            temperature=0.9,
            max_tokens=512
        )
        generated = response.choices[0].message.content
        return jsonify({"text": generated})
    except Exception as e:
        print("OpenAI error:", e)
        return jsonify({"error": str(e)}), 500

@bp.route('/analyze_file', methods=['POST'])
@auth_required
def analyze_file():
    file = request.files.get('file')
    if not file:
        return jsonify({"error": "No file provided"}), 400

    filename = file.filename.lower()
    text = ""
    try:
        if filename.endswith('.txt'):
            text = file.read().decode('utf-8')
        elif filename.endswith('.json'):
            obj = json.load(file)
            text = obj.get('text', '') or json.dumps(obj)
        elif filename.endswith('.pdf'):
```

```
            doc = fitz.open(stream=file.read(), filetype='pdf')
            text = "\n".join([page.get_text() for page in doc])
        elif filename.endswith('.docx'):
            doc = docx.Document(file)
            text = "\n".join([p.text for p in doc.paragraphs])
        elif filename.endswith(('.mp3', '.wav', '.m4a')):
            try:
                suffix = os.path.splitext(filename)[1]
                with tempfile.NamedTemporaryFile(delete=False, suffix=suffix) as tmp:
                    file.save(tmp.name)
                model = whisper.load_model("base")
                result = model.transcribe(tmp.name)
                text = result["text"]
                os.remove(tmp.name)
            except Exception as e:
                print("Whisper error:", e)
                return jsonify({"error": f"Whisper transcription error: {str(e)}"}), 500
        else:
            return jsonify({"error": "Unsupported file format"}), 400

        if not text.strip():
            return jsonify({"error": "Файл не містить тексту"}), 400
        return jsonify({"text": text})
    except Exception as e:
        return jsonify({"error": str(e)}), 500


@bp.route('/analyze_file_full', methods=['POST'])
@auth_required
def analyze_file_full():
```

```
file = request.files.get('file')
if not file:
    return jsonify({"error": "No file provided"}), 400
filename = file.filename.lower()
text = ""
try:
    if filename.endswith('.txt'):
        text = file.read().decode('utf-8')
    elif filename.endswith('.json'):
        obj = json.load(file)
        text = obj.get('text', '') or json.dumps(obj)
    elif filename.endswith('.csv'):
        df = pd.read_csv(file)
        text = " ".join(map(str, df.values.flatten()))
    elif filename.endswith('.pdf'):
        doc = fitz.open(stream=file.read(), filetype='pdf')
        text = "\\n".join([page.get_text() for page in doc])
    elif filename.endswith('.docx'):
        doc = docx.Document(file)
        text = "\\n".join([p.text for p in doc.paragraphs])

    elif filename.endswith(('.mp3', '.wav', '.m4a')):
        language = request.form.get('language', 'auto')
        suffix = os.path.splitext(filename)[1]
        tmp = tempfile.NamedTemporaryFile(delete=False, suffix=suffix)
        tmp_path = tmp.name
        tmp.close()
        file.save(tmp_path)
        model = whisper.load_model("base")
        result = model.transcribe(tmp_path, language=language if language != "auto" else None)
```

```
    text = result["text"]
    os.remove(tmp_path)
else:
    return jsonify({"error": "Unsupported file format"}), 400
if not text.strip():
    return jsonify({"error": "Файл не містить тексту"}), 400

user_id = g.user.get("uid")
from app.services.sentiment_service import analyze_sentiment
from app.services.topic_service import classify_topic
from app.services.toxicity_service import detect_toxicity
from app.utils.text_utils import get_preview_text
from app.services.openai_service import generate_summary
from app.services.firebase_service import save_analysis
sentiment = analyze_sentiment(text)
topics = classify_topic(text)
toxicity = detect_toxicity(text)
summary = generate_summary(text)
preview = get_preview_text(text)

results = {
    "sentiment": sentiment,
    "topics": topics,
    "toxicity": toxicity
}

payload = {
    "preview_text": preview,
    "summary": summary,
    "input_text": text,
```

```
            "language": "uk",
            "results": results
        }

        save_analysis(user_id, payload)
        return jsonify(payload), 200
    except Exception as e:
        print("Full file analyze error:", e)
        return jsonify({"error": str(e)}), 500
```

full_analysis.py

```
from flask import Blueprint, request, jsonify, g
from app.utils.auth_decorator import auth_required
from app.services.sentiment_service import analyze_sentiment
from app.services.topic_service import classify_topic
from app.services.toxicity_service import detect_toxicity
from app.services.firebase_service import save_analysis
from app.utils.text_utils import get_preview_text
from app.services.openai_service import generate_summary

full_analysis_bp = Blueprint('full_analysis', __name__)

@full_analysis_bp.route('/full-analysis', methods=['POST'])
@auth_required
def full_analysis():
    data = request.get_json() or {}
    user_id = g.user.get("uid")
    input_text = data.get("text", "").strip()
    language = data.get("language", "uk")
    if not input_text:
```

```
        return jsonify({"error": "Empty text"}), 400
    try:
        sentiment = analyze_sentiment(input_text)
        topics = classify_topic(input_text)
        toxicity = detect_toxicity(input_text)
        summary = generate_summary(input_text)
        preview = get_preview_text(input_text)
        results = {
            "sentiment": sentiment,
            "topics": topics,
            "toxicity": toxicity
        }

        payload = {
            "preview_text": preview,
            "summary": summary,
            "input_text": input_text,
            "language": language,
            "results": results
        }

        save_analysis(user_id, payload)
        return jsonify(payload), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500
```

firebase_service.py

```
import firebase_admin
from firebase_admin import credentials, auth, firestore
import os
```

```
from datetime import datetime
from firebase_admin import firestore
from google.cloud.firestore import SERVER_TIMESTAMP
# ініціалізація тільки один раз
if not firebase_admin._apps:
    cred = credentials.Certificate("instance/nlp-project-ai-firebase-adminsdk-fbsvc-bf5453db79.json")
#  service account JSON
    firebase_admin.initialize_app(cred)
# токен перевірки
def verify_id_token(id_token):
    try:
        decoded_token = auth.verify_id_token(id_token)
        return decoded_token
    except Exception as e:
        print("Token verification failed:", e)
        return None
# firestore клієнт
db = firestore.client()
#  збереження аналізу
def save_analysis(user_id, analysis_data):
    doc_ref = db.collection("users").document(user_id).collection("analyses").document()
    doc_ref.set({
        "timestamp": firestore.SERVER_TIMESTAMP,
        **analysis_data
    })
def get_analyses(user_id):
    docs = db.collection("users").document(user_id).collection("analyses")\
        .order_by("timestamp", direction=firestore.Query.DESCENDING).stream()
    return [{**doc.to_dict(), "id": doc.id} for doc in docs]


def delete_user_analyses(user_id):
```

```
    analysis_ref = db.collection("users").document(user_id).collection("analyses")
    docs = analysis_ref.stream()
    for doc in docs:
        doc.reference.delete()

def delete_analysis(user_id, analysis_id):
    try:
        doc_ref = db.collection("analyses").document(analysis_id)
        doc = doc_ref.get()
        if doc.exists and doc.to_dict().get("user_id") == user_id:
            doc_ref.delete()
            return True
        return False
    except Exception as e:
        print(f"Помилка при видаленні аналізу: {e}")
        return False

def delete_user_analysis_by_id(user_id, analysis_id):
    try:
        doc_ref = db.collection("users").document(user_id).collection("analyses").document(analysis_id)
        doc = doc_ref.get()
        if doc.exists:
            doc_ref.delete()
            return True
        return False
    except Exception as e:
        print(f"Помилка при видаленні аналізу користувача: {e}")
        return False
```

sentiment_service.py

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch
import torch.nn.functional as F
from app.utils.text_chunker import chunk_text

MODEL_PATH = "app/models/sentiment_model"
tokenizer = AutoTokenizer.from_pretrained(MODEL_PATH)
model = AutoModelForSequenceClassification.from_pretrained(MODEL_PATH)
model.eval()

label_map = {0: "negative", 1: "neutral", 2: "positive"}
def analyze_sentiment(text: str) -> dict:
    text = text.replace('\n', ' ').strip()
    chunks = chunk_text(text, tokenizer)
    scores = []
    for chunk in chunks:
        inputs = tokenizer(chunk, return_tensors="pt", truncation=True)
        with torch.no_grad():
            outputs = model(**inputs)
            probs = F.softmax(outputs.logits, dim=1).squeeze()
        scores.append(probs)
    avg_probs = torch.stack(scores).mean(dim=0)
    predicted_label = torch.argmax(avg_probs).item()
    return {
        "label": label_map[predicted_label],
        "scores": {label_map[i]: float(avg_probs[i]) for i in range(len(label_map))}
    }
```

topic_service.py

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
```

```
import torch
from app.utils.text_chunker import chunk_text

MODEL_PATH = "app/models/topic_model"
tokenizer = AutoTokenizer.from_pretrained(MODEL_PATH)
model = AutoModelForSequenceClassification.from_pretrained(MODEL_PATH)
model.eval()

label_list = [
    'arts_&_culture', 'business_&_entrepreneurs', 'celebrity_&_pop_culture',
    'diaries_&_daily_life', 'family', 'fashion_&_style', 'film_tv_&_video',
    'fitness_&_health', 'food_&_dining', 'gaming', 'learning_&_educational',
    'music', 'news_&_social_concern', 'other_hobbies', 'relationships',
    'science_&_technology', 'sports', 'travel_&_adventure', 'youth_&_student_life'
]
ukr_topic_labels = {
    "arts_&_culture": "Мистецтво та культура",
    "business_&_entrepreneurs": "Бізнес і підприємництво",
    "celebrity_&_pop_culture": "Знаменитості та поп-культура",
    "diaries_&_daily_life": "Щоденники та повсякденне життя",
    "family": "Сім'я",
    "fashion_&_style": "Мода та стиль",
    "film_tv_&_video": "Фільми, телебачення та відео",
    "fitness_&_health": "Фітнес і здоров'я",
    "food_&_dining": "Їжа та харчування",
    "gaming": "Ігри",
    "learning_&_educational": "Освіта та навчання",
    "music": "Музика",
    "news_&_social_concern": "Новини та соціальні проблеми",
    "other_hobbies": "Інші хобі",
```

```
    "relationships": "Стосунки",
    "science_&_technology": "Наука і технології",
    "sports": "Спорт",
    "travel_&_adventure": "Подорожі та пригоди",
    "youth_&_student_life": "Студентське життя та молодь"
}
def classify_topic(text):
    if not isinstance(text, str) or not text.strip():
        return ["INVALID_INPUT"]
    try:
        chunks = chunk_text(text, tokenizer)
        all_probs = []
        for chunk in chunks:
            inputs = tokenizer(chunk, return_tensors="pt", truncation=True, padding=True,
max_length=512)
            with torch.no_grad():
                outputs = model(**inputs)
                probs = torch.softmax(outputs.logits, dim=1)
                all_probs.append(probs.squeeze())
        avg_probs = torch.stack(all_probs).mean(dim=0)
        top_indices = torch.topk(avg_probs, k=3).indices.tolist()
        top_labels = [label_list[i] for i in top_indices]
        return [ukr_topic_labels.get(label, label) for label in top_labels]
    except Exception as e:
        print("classify_topic ERROR:", str(e))
        return ["ERROR"]


toxicity_service.py
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch
import torch.nn.functional as F
```

```
from app.utils.text_chunker import chunk_text

MODEL_PATH = "app/models/toxicity_model"
tokenizer = AutoTokenizer.from_pretrained(MODEL_PATH)
model = AutoModelForSequenceClassification.from_pretrained(MODEL_PATH)
model.eval()

labels = {0: "non-toxic", 1: "toxic"}

def detect_toxicity(text: str) -> dict:
    text = text.strip().replace("\n", " ")
    chunks = chunk_text(text, tokenizer)
    toxic_detected = False
    all_probs = []

    for chunk in chunks:
        inputs = tokenizer(chunk, return_tensors="pt", truncation=True)
        with torch.no_grad():
            outputs = model(**inputs)
            probs = F.softmax(outputs.logits, dim=1).squeeze()
            all_probs.append(probs)
            if torch.argmax(probs).item() == 1:
                toxic_detected = True

    avg_probs = torch.stack(all_probs).mean(dim=0)
    predicted = 1 if toxic_detected else 0

    return {
        "label": labels[predicted],
        "confidence": float(avg_probs[predicted]),
```

```
        "scores": {labels[i]: float(avg_probs[i]) for i in range(len(labels))}
    }
```

text_chunker.py

```
from transformers import PreTrainedTokenizer
from typing import List

def chunk_text(text: str, tokenizer: PreTrainedTokenizer, max_tokens: int = 480) -> List[str]:
    words = text.strip().split()
    chunks = []
    current_chunk = []
    for word in words:
        current_chunk.append(word)
        tokens = tokenizer(" ".join(current_chunk), truncation=False,
add_special_tokens=False)["input_ids"]
        if len(tokens) > max_tokens:
            current_chunk.pop()
            chunks.append(" ".join(current_chunk))
            current_chunk = [word]
    if current_chunk:
        chunks.append(" ".join(current_chunk))

    return chunks
```

ДОДАТОК Б

Комплексний аналіз настроїв тексту за допомогою штучного інтелекту

Презентація

Аркушів 9

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститутімені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

Комплексний аналіз настроїв тексту за допомогою штучного інтелекту

Шарабура Еліна Дмитрівна, ТВ-з11

Професор кафедри ІПЗЕ Мусієнко Андрій Петрович, д.т.н., доцент

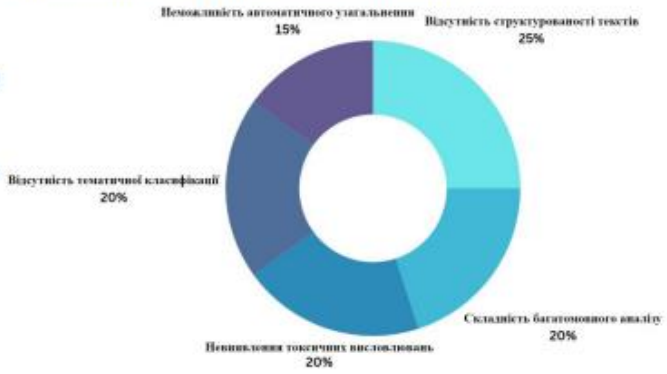
Київ-2025

Актуальність теми

Обсяг неструктурованого тексту стрімко зростає, що унеможливлює його ручний аналіз. Існуючі рішення не враховують ключові моменти токсичність, тематику багатомовність чи суть. Потрібна система для швидкого та комплексного аналізу тексту.

Розробка рішення показує, як штучний інтелект можна ефективно інтегрувати в реальні застосунки. Це важливо для сучасної інженерії програмного забезпечення, особливо в напрямку аналізу тексту та в обробці природної мови.

- Мета – створити вебсервіс для комплексного аналізу тексту.
- Об'єкт – система для аналізу тексту.
- Предмет – моделі штучного інтелекту для визначення тональності, токсичності, тематики й узагальнення.



Постановка задачі

Розробити вебсервіс для комплексного аналізу тексту з використанням моделей штучного інтелекту.

Основні задачі:

1. Провести аналіз предметної області та аналогів
2. Обрати архітектуру вебзастосунку
3. Інтегрувати моделі для визначення:
   - тональності
   - токсичності
   - тематики
   - резюме тексту
4. Забезпечити підтримку багатомовності
5. Реалізувати зручний інтерфейс користувача
6. Реалізувати збереження історії аналізів (для авторизованих користувачів)
7. Провести тестування функціональності

Аналіз предметної області

Огляд існуючих рішень та їх недоліків

Інструмент	Недолік
VADER	Лише англійська, базова тональність
TextBlob	Низька точність, не підтримує токсичність
Perspective API	Працює лише з токсичністю, закритий API
HuggingFace Demo	Без інтеграції, не зберігає історію
GPT API (як основа)	Платна модель, залежність від інтернету, недоцільно використовувати як базу
Django / FastAPI	Складніші для легкої інтеграції REST API у порівнянні з Flask
MySQL / PostgreSQL	Не підходять для гнучкої структури збереження JSON-даних, складніші в налаштуванні

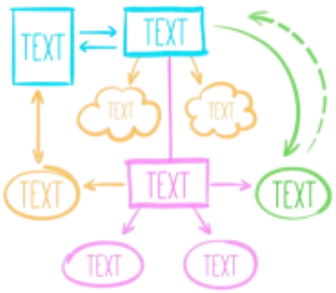
Аналіз предметної області

Обґрунтування вибору технологій та інструментів

Інструмент	Недолік
XLM-RoBERTa (3 моделі)	Локальні, точні під конкретні потреби, підтримують багатомовність,
GPT-3.5 (через API)	Ефективна генерація резюме й переформулювання
Whisper (OpenAI)	Якісна транскрипція аудіо, підтримка української мови
Chart.js, ApexCharts	Інформативна та гнучка візуалізація результатів, легко інтегрувати
HTML / CSS / JavaScript	Просте створення адаптивного й інтерактивного інтерфейсу
Flask	Простий у розгортанні REST API, легкий та гнучкий
Firebase	Швидка авторизація та зберігання історії в реальному часі

Логіка обробки тексту в системі:

Усі текстові запити проходять чітку послідовність дій — від попередньої обробки до багаторівневого аналізу та формування відповіді.
Така структура дозволяє точно обробляти великі тексти з підтримкою кількох мов.



Використані моделі у програмному забезпеченні

Модель	Задача	Особливість
cardiffnlp/twitter-xlm-roberta-base-sentiment	Аналіз емоційності	Базується на XLM-RoBERTa, багатомовна
textdetox/xlmr-large-toxicity-classifier-v2	Визначення токсичності	XLM-RoBERTa, натренована на токсичних прикладах
cardiffnlp/tweet-topic-21-multi	Класифікація тем	XLM-RoBERTa, повертає кілька тем
GPT-3.5 API	Генерація резюме	Використовується через API, працює онлайн
Whisper	Транскрипція аудіо	Локальна модель розпізнавання мовлення

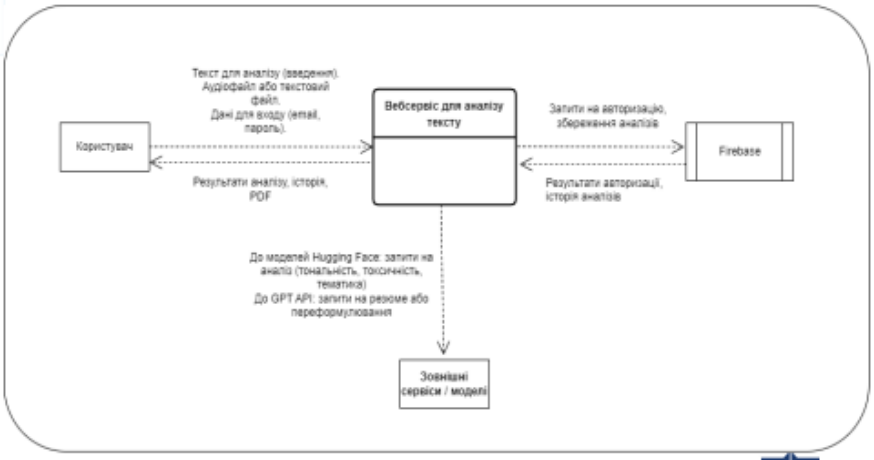
XLM-RoBERTa → трансформерна багатомовна модель, працюють локально

GPT-3.5 API → модель генерації тексту, працює через зовнішній API

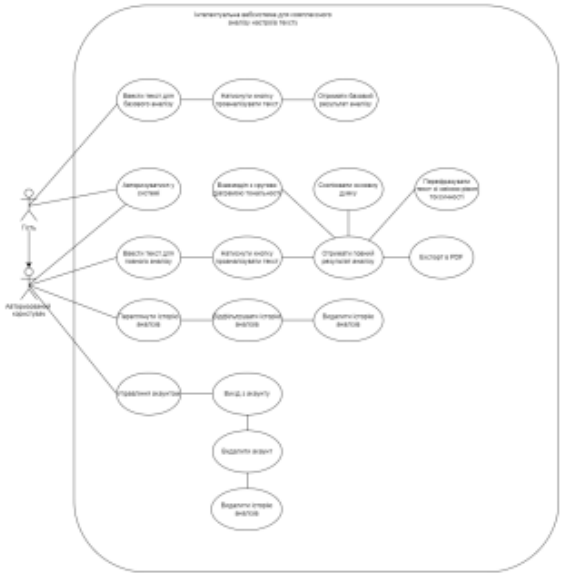
Whisper → модель розпізнавання мовлення, працює через підключення

Архітектура програмної системи. Dfd діаграма

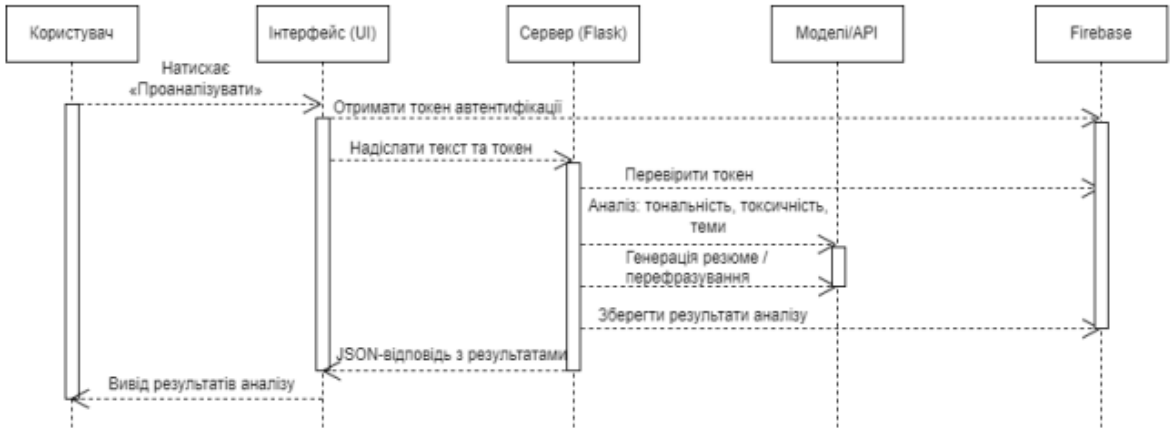
Архітектура побудована за принципом клієнт–сервер з окремими модулями для аналізу тексту, генерації та зберігання даних.



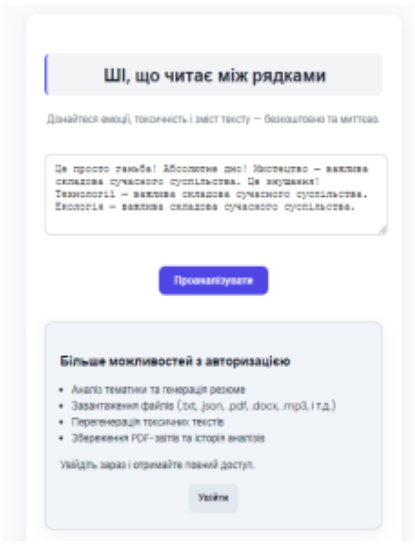
Use Case/Діаграма прецедентів взаємодії користувачів із вебсервісом

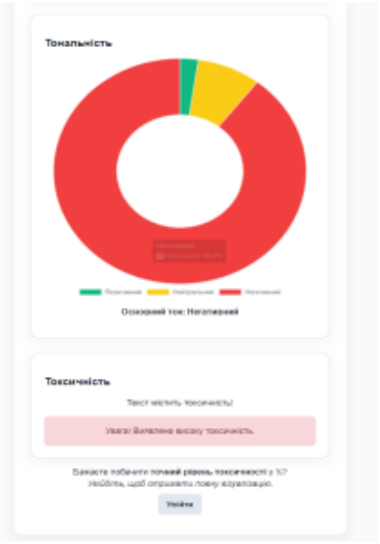


Sequence diagram/Діаграма послідовностей взаємодії користувача через інтерфейс з вебсистемою під час повного аналізу тексту

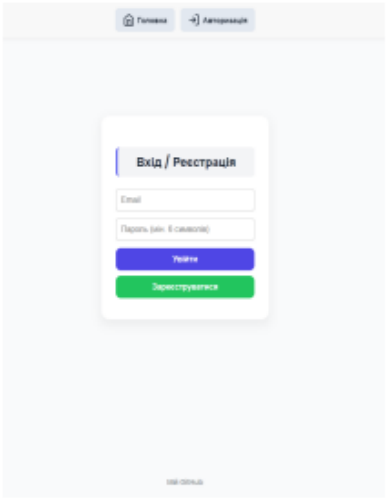


Реалізація системи. Базовий аналіз. Авторизація



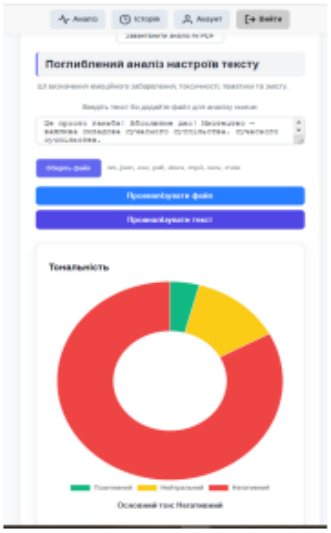


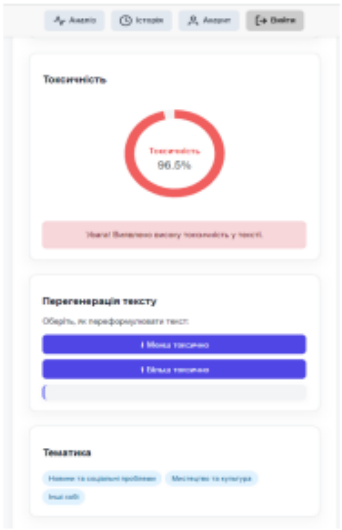
Сторінка сервісу з базовим аналізом (без авторизації)

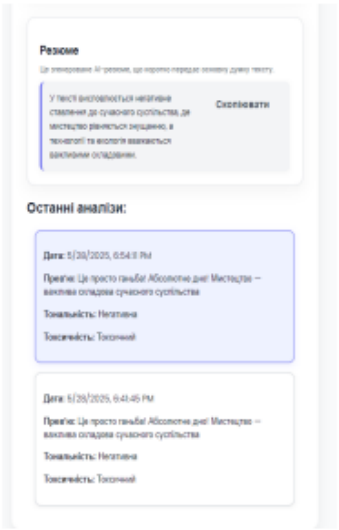


Сторінка входу в систему

Реалізація системи. Повний аналіз

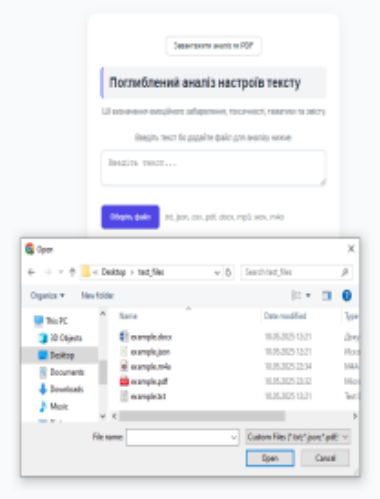




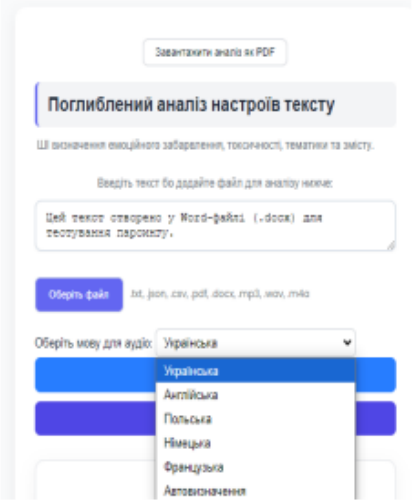


Головна сторінка сервісу з повним аналізом (з авторизацією)

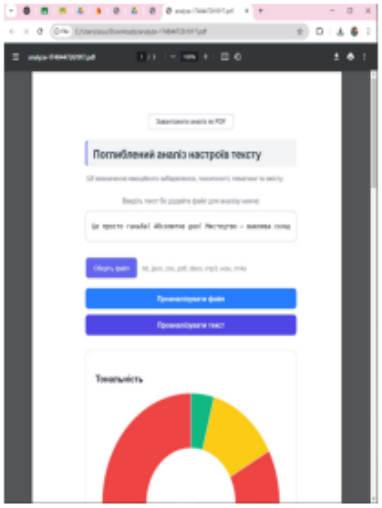
Реалізація системи. Вхід в систему. Додатковий функціонал



Вибір файлу для аналізу

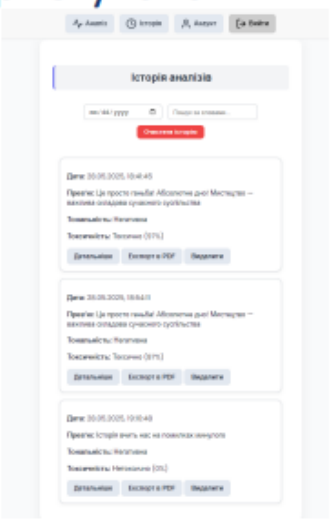


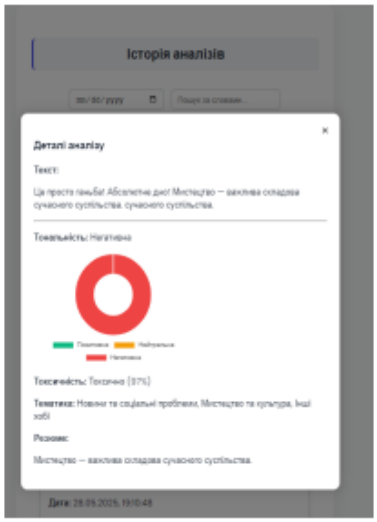
Вибір мови для аналізу файлу

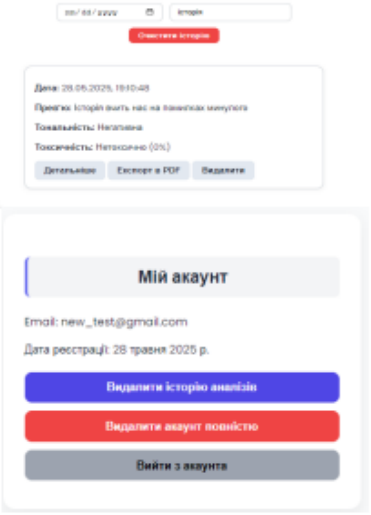


Експорт результату аналізу в PDF

Реалізація системи. Історія аналізів. Акаунт користувача





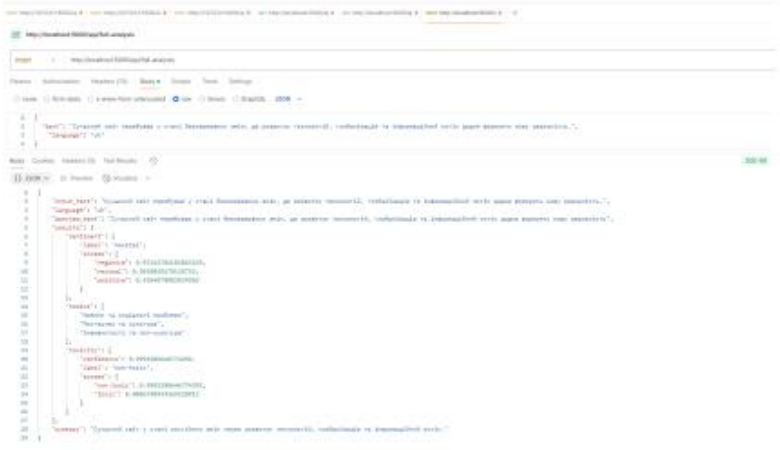


Сторінка історії аналізів та акаунт користувача

Тестування системи

- ✓ Модульне тестування: toxicity_service.py, sentiment_service.py, topic_service.py
- ✓ Інтеграційне: взаємодія Flask API з моделями
- ✓ Системне: повний аналіз через запити в Postman

Test Case

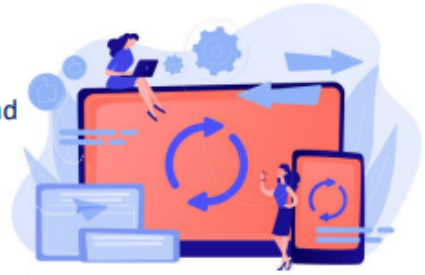


Запит /full-analysis в Postman

Впровадження та супровід

Процес розгортання системи:
Сервіс розгортається локально або на хмарному сервері.
Моделі класифікації завантажуються заздалегідь і зберігаються локально. Генеративна модель підключається через API. Запуск через Flask (Python), доступ через браузер.

Документація:
Коротка інструкція для користувача вбудована в інтерфейс.
Опис API-поведінки задокументовано в окремому файлі README.md
Короткий опис моделей і коду - через коментарі в модулях.



Плани розвитку:
Можливість роботи офлайн без інтернету навіть для резюме .
Розширення типів файлів (наприклад, презентації, зображення, відео).
Розгортання на віддаленому сервері з CI/CD.

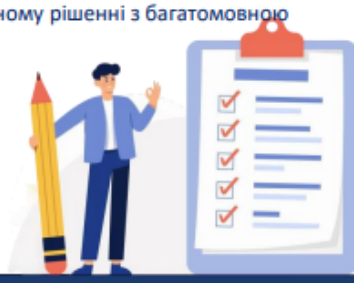
Висновки

Основні результати роботи

1. Розроблено вебсервіс для повного аналізу тексту
2. Реалізовано визначення емоційності, токсичності, тематики та генерацію резюме, експорт результатів, взаємодія з історією
3. Здійснюється збереження історії для авторизованих користувачів
4. Підтримується робота з файлами та аудіо

Мету досягнуто: створено функціональну систему з використанням моделей ШІ. Виконано всі поставлені задачі (аналіз, інтерфейс, збереження, авторизація).

Розроблена система має практичну цінність як інструмент для автоматизованого аналізу текстів різного типу (новини, відгуки, освітній та користувацький контент) з метою виявлення тональності, токсичності, тематики та генерації стислого змісту, а також наукову — як приклад застосування сучасних трансформерних моделей у реальному програмному рішенні з багатомовною підтримкою, модульною структурою та можливістю масштабування.



Перспективи подальших розробок/вдосконалень/досліджень

- Реалізувати повну офлайн-роботу системи без використання зовнішніх API
- Розширити перелік форматів файлів (PPTX, зображення , відео)
- Впровадити CI/CD для автоматичного розгортання та оновлення системи
- Інтегрувати статистику: трекінг змін у тематиці, тональності, активності користувача
- Вивести систему як окремий продукт з публічним доступом через домен